Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 1, 2013

E. Ivov
Jitsi
E. Rescorla
RTFM, Inc.
J. Uberti
Google

January 28, 2013

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

draft-ivov-mmusic-trickle-ice-00

Abstract

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

The above mechanism is also referred to as "trickle ICE".

Status of this Memo

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

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

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

This Internet-Draft will expire on August 1, 2013.

Copyright Notice

Table of Contents

1. Introduction

The Interactive Connectivity Establishment (ICE) protocol [RFC5245] describes mechanisms for gathering, candidates, prioritizing them, choosing default ones, exchanging them with the remote party, pairing them and ordering them into check lists. Once all of the above have been completed, and only then, the participating agents can begin a phase of connectivity checks and eventually select the pair of candidates that will be used in the following session.

While the above sequence has the advantage of being relatively straightforward to implement and debug once deployed, it may also prove to be rather lengthy. Gathering candidates or candidate harvesting would often involve things like querying STUN [RFC5389] servers, discovering UPnP devices, and allocating relayed candidates at TURN [RFC5766] servers. All of these can be delayed for a noticeable amount of time and while they can be run in parallel, they still need to respect the pacing requirements from [RFC5245], which is likely to delay them even further. Some or all of the above would also have to be completed by the remote agent. Both agents would next perform connectivity checks and only then would they be ready to begin streaming media.

All of the above could lead to relatively lengthy session establishment times and degraded user experience.

The purpose of this document is to define an alternative mode of operation for ICE implementations, also known as "trickle ICE", where candidates can be exchanged incrementally. This would allow ICE agents to exchange host candidates as soon as a session has been initiated. Connectivity checks for a media stream would also start as soon as the first candidates for that stream have become available.

Trickle ICE allows reducing session establishment times in cases where connectivity is confirmed for the first exchanged candidates (e.g. where the host candidates for one of the agents are directly reachable from the second agent). Even when this is not the case, running candidate harvesting for both agents and connectivity checks all in parallel allows to considerably reduce ICE processing times.

It is worth pointing out that before being introduced to the IETF, trickle ICE had already been included in specifications such as XMPP Jingle [XEP-0176] and it has been in use in various implementations and deployments.

In addition to the basics of trickle ICE, this document also describes how support for trickle ICE needs to be discovered, how

regular ICE processing needs to be modified when building and updating check lists, and how trickle ICE implementations should interoperate with agents that only implement [RFC5245] processing.

This specification does not define usage of trickle ICE with any specific signalling protocol, contrary to [RFC5245] which contains a usage for ICE wht SIP. Such usages would have to be specified in separate documents.

Trickle ICE does however reuse and build upon the SDP syntax defined by vanilla ICE.

2. Terminology

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

This specification makes use of all terminology defined by the protocol for Interactive Connectivity Establishment in [RFC5245].

Vanilla ICE: The Interactive Connectivity Establishment protocol as defined in [RFC5245].

Candidate Harvester: A module used by an ICE agent to obtain local candidates. Candidate harvesters use different mechanisms for discovering local candidates. Some of them would typically make use of protocols such as STUN or TURN. Others may also employ techniques that are not referenced within [RFC5245]. UPnP based port allocation and XMPP Jingle Relay Nodes [XEP-0278] are among the possible examples.

3. Incompatibility with Standard ICE

The ICE protocol was designed to be fairly flexible so that it would work in and adapt to as many network environments as possible. It is hence important to point out at least some of the reasons why, despite its flexibility, the specification in [RFC5245] would not support trickle ICE.

[RFC5245] describes the conditions required to update check lists and timer states while an ICE agent is in the Running state. These conditions are verified upon transaction completion and one of them stipulates that:

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

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

- Alice and Bob are both located in different networks with Network Address Translation (NAT). Alice and Bob themselves have different address but both networks use the same [RFC1918] block.
- Alice sends Bob the candidate 10.0.0.10 which also happens to correspond to an existing host on Bob's network.
- Bob creates a check list consisting solely of 10.0.0.10 and starts checks.
- These checks reach the host at 10.0.0.10 in Bob's network, which responds with an ICMP "port unreachable" error and per [RFC5245] Bob marks the transaction as Failed.

At this point the check list only contains Failed candidates and the valid list is empty. This causes the media stream and potentially all ICE processing to Fail.

A similar race condition would occur if the initial offer from Alice only contains candidates that can be determined as unreachable (per [I-D.keranen-mmusic-ice-address-selection]) from any of the candidates that Bob has gathered. This would be the case if Bob's candidates only contain IPv4 addresses and the first candidate that he receives from Alice is an IPv6 one.

Another potential problem could arise when a non-trickle ICE implementation sends an offer to a trickle one. Consider the following case:

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

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

4. Determining Support for Trickle ICE

According to [RFC5245] every time an agent supporting trickle ICE generates an offer or an answer, it MUST include the "trickle" token in the ice-options attribute. Syntax for this token is defined in Section 5.1.

Additionally, in order to avoid interoperability problems such as those described in Section 3, it is important that trickle ICE negotiation is only attempted in cases where the remote party actually supports this specification. Agents that receive offers or answers can verify support by examining them for the "trickle" ice-options token. However, agents that are about to send a first offer, have no immediate way of doing this. This means that usages of trickle for specific protocols would need to either:

- Provide a way for agents to verify support of trickle ICE prior to initiating a session. XMPP's Service discovery [XEP-0030] is an example for one such mechanism;
- Make support for trickle ICE mandatory so that support could be assumed the agents.

Alternately, for cases where a protocol provides neither of the above, agents may either rely on provisioning/configuration, or use the half trickle procedure described in Section 4.1.

Note that out-of-band discovery semantics and half trickle are only necessary prior to session initiation, or in other words, when sending the initial offer. Once a session is established and trickle ICE support is confirmed for both parties, either agent can use full trickle for subsequent offers.

4.1. Unilateral Use of Trickle ICE (Half Trickle)

The idea of using half trickle is about having the caller send a regular, vanilla ICE offer, with a complete set of candidates. This offer still indicates support for trickle ice, so the answerer is able to respond with an incomplete set of candidates and continue trickling the rest. Half trickle offers will typically contain an end-of-candidates indication.

The 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 it contains a full set of candidates, its first offer can thus be handled by a regular vanilla ICE agent, while still allowing a trickle one to use the optimisation defined in this specification. This prevents negotiation from failing in the former case while still giving roughly half the trickle ICE benefits in the latter (hence the

name of the mechanism).

Use of half trickle is only necessary during an initial offer/answer exchange. Once both parties have received a session description from their peer, they can each reliably determine trickle ICE support and use it for all subsequent offer/answer exchanges.

It is worth pointing out that using half trickle may actually bring more than just half the improvement in terms of user experience. This can happen in cases where an agent starts gathering candidates upon user interface cues that a call is pending, such as activity on a keypad or the phone going off hook. This would mean a part or all candidate harvesting could have completed before the agent actually needs to send the offer. Given that the answerer will be able to trickle candidates, both agents will be able to start connectivity checks and complete ICE processing earlier than with vanilla ICE and potentially even as early as with full trickle.

However, such anticipation is not 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. Still, even in these cases, using half trickle would be an improvement over vanilla ICE as it would optimize performance for answerers.

5. Sending the Initial Offer

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

Trickle ICE agents MAY include any set of candidates in an offer. This includes the possibility of generating one with no candidates, or one that contains all the candidates that the agent is planning on using in the following session.

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

checks.

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

Prior to actually sending an initial offer, agents MAY verify if the remote party supports trickle ICE, where such mechanisms actually exist. If absence of such support is confirmed agents MUST fall back to using vanilla ICE or abandon the entire session.

All trickle ICE offers and answers MUST indicate support of this specification, as explained in Section 5.1.

Calculating priorities and foundations, as well as determining redundancy of candidates work the same way they do with vanilla ICE.

5.1. Encoding the SDP

The process of encoding the SDP [RFC4566] is mostly the same as the one used by vanilla ICE. Still, trickle ICE does require a few differences described here.

Agents MUST indicate support for Trickle ICE by including the "trickle" token for the "a=ice-options" attribute:

```
a=ice-options:trickle
```

As mentioned earlier in this section, Offers and Answers can contain any set of candidates, which means that a trickle ICE session description MAY contain no candidates at all. In such cases the agent would still need to place an address in the "c=" line(s). If the use of a host address there is undesirable (e.g. for privacy reasons), the agent MAY set the connection address to 0.0.0.0. In this case it MUST also set the port number to 1. There is no need to include a fictitious 0.0.0.0 candidate when doing so.

6. Receiving the Initial Offer

When an agent receives an initial offer, it will first check if it indicates support for trickle ICE as explained in Section 4. If this is not the case, the agent MUST process the offer according to the [RFC5245] procedures or standard [RFC3264] processing in case no ICE support is detected at all.

It is worth pointing out that in case support for trickle ICE is confirmed, an agent will automatically assume support for vanilla ICE as well even if the support verification procedure in [RFC5245] indicates otherwise. Specifically, such verification would indicate lack of support when the offer contains no candidates. The 0.0.0.0 address present in the c= line in that case would not "appear in a candidate attribute". Obviously, a fallback to [RFC3264] is not required when this happens.

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

6.1. Sending the Initial Answer

An agent can respond to an initial offer at any point while gathering candidates. The answer can again contain any set of candidates including none or all of them. Unless it is protecting host addresses for privacy reasons, the agent would typically construct this initial answer including only them, thus allowing the remote party to also start forming checklists and performing connectivity checks.

The answer MUST indicate support for trickle ICE as described by Section 4.

6.2. Forming check lists and beginning connectivity checks

After exchanging offer and answer, and as soon as they have obtained local and remote candidates, agents will begin forming candidate pairs, computing their priorities and creating check lists according to the vanilla ICE procedures described in [RFC5245]. Obviously in order for candidate pairing to be possible, it would be necessary that both the offer and the answer contained candidates. If this was not the case agents will still create the check lists (so that their Active/Frozen state could be monitored and updated) but they will only populate them once they actually have the candidates.

Initially, all check lists will have their Active/Frozen state set to Frozen.

Trickle ICE agents will then also attempt to unfreeze the check list for the first media stream (i.e. the first media stream that was reported to the ICE implementation from the using application). If this checklist is still empty however, agents will continue examining media streams in the order they were reported and will unfreeze the

first non-empty checklist.

Respecting the order in which lists have been reported to an ICE implementation, or in other words, the order in which they appear in SDP, is helpful so that checks for the same media stream is more likely to be performed simultaneously by both agents.

6.3. Encoding the SDP

The process for encoding the SDP at the answerer is identical to the process followed by the offerer for both full and lite implementations, as described in Section 5.1.

7. Receiving the Initial Answer

When receiving an answer, agents will follow vanilla ICE procedures to determine their role and they would then form check lists (as described in Section 6.2) and begin connectivity checks .

8. Performing Connectivity Checks

For the most part, trickle ICE agents perform connectivity checks following vanilla ICE procedures. Of course, the asynchronous nature of candidate harvesting in trickle ICE would impose a number of changes described here.

8.1. Check List and Timer State Updates

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

- all of the pairs in the check list are either in the Failed or Succeeded state;
- if at least one of the components of the media stream has no pairs in its valid list.

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

- all candidate harvesters have completed and the agent is not expecting to learn any new candidates;

- the remote agent has sent an end-of-candidates indication for that check list as described in Section 9.2.

Vanilla ICE requires that agents then update all other check lists, placing one pair in each of them into the Waiting state, effectively unfreezing the check list. Given that with trickle ICE, other check lists may still be empty at that point, a trickle ICE agent SHOULD also maintain an explicit Active/Frozen state for every check list, rather than deducing it from the state of the pairs it contains. This state should be set to Active when unfreezing the first pair in a list or when that couldn't happen because a list was empty.

9. Learning and Sending Additional Local Candidates

After an offer or an answer have been sent, agents will most likely continue discovering new local candidates as STUN, TURN and other non-host candidate harvesting mechanisms begin to yield results. Whenever such a new candidate is learned agents will compute its priority, type, foundation and component id according to normal vanilla ICE procedures.

The new candidate is then checked for redundancy against the existing list of local candidates. If its transport address and base match those of an existing candidate, it will be considered redundant and will be ignored. This would often happen for server reflexive candidates that match the host addresses they were obtained from (e.g. when the latter are public IPv4 addresses). Contrary to vanilla ICE, trickle ICE agents will consider the new candidate redundant regardless of its priority. [TODO: is this OK? if not we need to check if the existing candidate was already used in conn checks, cancel them, and then restart them with the new candidate ... and in this specific case there's probably no point to do that].

Then, if no remote candidates are currently known for this same stream, the new candidate will simply be added to the list of local candidates.

Otherwise, if the agent has already learned of one or more remote candidates for this stream and component, it will begin pairing the new local candidates with them and adding the pairs to the existing check lists according to their priority. Forming candidate pairs will work the way it is described by the vanilla ICE specification. Actually adding the new pair to a check list however, will happen according to the rules described below.

If the new pair's local candidate is server reflexive, the server reflexive candidate MUST be replaced by its base before adding the

pair to the list. Once this is done, the agent examines the check list looking for another pair that would be redundant with the new one. If such a pair exists and its state is:

Succeeded: the newly formed pair is ignored.

Frozen or Waiting: the agent chooses the pair with the higher priority local candidate, places it in the state that the old pair was in (i.e. Frozen or Waiting) and removes the other one as redundant.

Failed: the agent chooses the pair with the higher priority local candidate, places it in the Waiting state and removes the other one as redundant.

In-Progress: The agent cancels the in-progress transaction (where cancellation happens as explained in Section 7.2.1.4 of [RFC5245]), then it chooses the pair with the higher priority local candidate, places it in the Waiting state and removes the other one as redundant.

For all other pairs, including those with a server reflexive local candidate that were not found to be redundant:

- if this check list is Frozen then the new pair will also be assigned a Frozen state.
- else if the check list is Active and it is either empty or contains only candidates in the Succeeded and Failed states, then the new pair's state is set to Waiting.
- else if the check list is non-empty and Active, then the new pair state will be set to

  Frozen: if there is at least one pair in the list whose foundation matches the one in the new pair and whose state is neither Succeeded nor Failed (eventually the new pair will get unfrozen after the the on-going check for the existing pair concludes);

  Waiting: if the list contains no pairs with the same foundation as the new one, or, in case such pairs exist but they are all in either the Succeeded or Failed states.

9.1. Encoding the SDP for Additional Candidates

To facilitate interoperability an ICE agent will encode additional candidates using the vanilla ICE SDP syntax. For example:

```
a=candidate:2 1 UDP 1658497328 198.51.100.33 5000 typ host
```

Given that such lines do not provide a relationship between the candidate and the m line that it relates to, signalling protocols using trickle ICE MUST establish that relation themselves. This can

be done in one of two ways: using the m-line index, or an MID [RFC3388]. The m-line index is a zero-based index, referring to the Nth m-line in the SDP. The MID uses "media stream identification", as defined in [RFC3388], to identify the m-line. When creating candidate lines usages of trickle ICE SHOULD use the MID if possible, or the m-line index if not. Agents MUST NOT send individual candidates any prior to generating the corresponding SDP session description.

The exact means of transporting additional candidates to a remote agent is left to the protocols using trickle ICE. It is important to note, however, that these candidate exchanges are not part of the offer/answer model. [TODO: or should we pick one of the two? Is there any reason not to mandate MID?]

9.2. Announcing End of Candidates

Once all candidate harvesters for a specific media stream complete, or expire, the agent MUST generate an "end-of-candidates" indication for that stream and send it to the remote agent via the signalling channel. The SDP representation for end-of-candidates following form:

```
a=end-of-candidates
```

The end-of-candidates notifications can be sent as part of an offer, which would typically be the case with half trickle initial offers, they can accompany the last candidate an agent can send for a stream, and they can also be sent alone (e.g. after STUN Binding requests or TURN Allocate requests to a server timeout and the agent has no other active harvesters).

Receiving an end-of-candidates notification allows an agent to update check list states and, in case valid pairs do not exist for every component in every media stream, determine that ICE processing has failed.

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

Half trickle agents SHOULD send end-of-candidates together with their initial offer unless they are planning on potentially sending additional candidates in case the remote party turns out to actually

support trickle ICE. (TODO: can this be useful?)

Once the agent sends the end-of-candidates event, it will update the state of the corresponding check list as explained in section Section 8.1

9.3. Receiving an End Of Candidates Notification

When an agent receives an end-of-candidates notification for a specific check list, they will update its state as per Section 8.1. In case the list is still in the Active state after the update, the agent will persist the the fact that an end-of-candidates notification has been received for and take it into account in future list updates.

[TODO would we like to say anything about nomination? in general this would be up to implementers but is there a need for some basic guidelines?]

10. Receiving Additional Remote Candidates

At any point of ICE processing, a trickle ICE agent may receive new candidates from the remote agent. When this happens and no local candidates are currently known for this same stream, the new remote candidates are simply added to the list of remote candidates.

Otherwise, the new candidates are used for forming candidate pairs with the pool of local candidates.

Once the remote agent has completed candidate harvesting, it will send an end-of-candidates event. Upon receiving such an event, the local agent MUST update check list states as per Section 8.1. This may lead to some check lists being marked as Failed.

11. Concluding ICE Processing

This specification does not directly modify the procedures ending ICE processing described in Section 8 of [RFC5245], and trickle ICE implementations will follow the same rules.

12. Subsequent Offer/Answer Exchanges

Either agent MAY generate a subsequent offer at any time allowed by [RFC3264]. When this happens agents will use [RFC5245] semantics to determine whether or not the new offer requires an ICE restart. If

this is the case then agents would perform trickle ICE as they would in an initial offer/answer exchange.

The only differences between an ICE restart and a brand new media session are that:

- during the restart, media can continue to be sent to the previously validated pair.
- both agents are already aware whether or not their peer supports trickle ICE, and there is no longer need for performing half trickle or confirming support with other mechanisms.

13. Interaction with ICE Lite

Behaviour of Trickle ICE capable ICE lite agents does not require any particular rules other than those already defined in this specification and [RFC5245]. This section is hence added with an informational purpose only.

A Trickle ICE capable ICE Lite agent would generate offers or answers as per [RFC5245]. Both will indicate support for trickle ICE (Section 5.1) and given that they will contain a complete set of candidates (the agent's host candidates) these offers and answers would also be accompanied with an end-of-candidates notification.

14. Example Flow

A typical successful trickle ICE exchange with an Offer/Answer protocol would look this way:

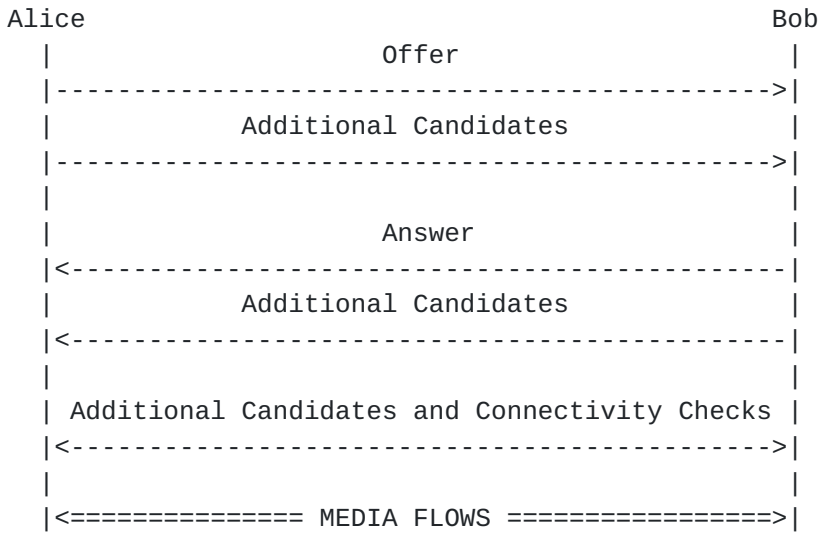

Figure 1: Example

15. Security Considerations

[TODO]

16. Acknowledgements

The authors would like to thank Bernard Adoba, Christer Holmberg, Enrico Marocco, Jonathan Lennox and Martin Thomson for their reviews and suggestions on improving this document.

17. References

17.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3264] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, June 2002.

[RFC4566] Handley, M., Jacobson, V., and C. Perkins, "SDP: Session Description Protocol", RFC 4566, July 2006.

[RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, April 2010.

17.2. Informative References

[I-D.keranen-mmusic-ice-address-selection] Keraenen, A. and J. Arkko, "Update on Candidate Address Selection for Interactive Connectivity Establishment (ICE)", draft-keranen-mmusic-ice-address-selection-01 (work in progress), July 2012.

[RFC1918] Rekhter, Y., Moskowitz, R., Karrenberg, D., Groot, G., and E. Lear, "Address Allocation for Private Internets", BCP 5, RFC 1918, February 1996.

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E.

Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[RFC3388] Camarillo, G., Eriksson, G., Holler, J., and H. Schulzrinne, "Grouping of Media Lines in the Session Description Protocol (SDP)", RFC 3388, December 2002.

[RFC3840] Rosenberg, J., Schulzrinne, H., and P. Kyzivat, "Indicating User Agent Capabilities in the Session Initiation Protocol (SIP)", RFC 3840, August 2004.

[RFC4787] Audet, F. and C. Jennings, "Network Address Translation (NAT) Behavioral Requirements for Unicast UDP", BCP 127, RFC 4787, January 2007.

[RFC5389] Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for NAT (STUN)", RFC 5389, October 2008.

[RFC5766] Mahy, R., Matthews, P., and J. Rosenberg, "Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)", RFC 5766, April 2010.

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

[XEP-0115] Hildebrand, J., Saint-Andre, P., Troncon, R., and J. Konieczny, "XEP-0115: Entity Capabilities", XEP XEP-0115, February 2008.

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

[XEP-0278] Camargo, T., "XEP-0278: Jingle Relay Nodes", XEP XEP-0278, June 2011.

Appendix A. Open issues

At the time of writing of this document the authors have no clear view on how and if the following list of issues should be addressed.

A.1. MID/Stream Indices in SDP

This specification does not currently define syntax for candidate-to-stream bindings although it says that they should be implemented with MID or a stream index. Yet, it is reasonable to assume that most usages would need to do this within the SDP and it may make sense to agree on the format. Here's one possible way to do this:

```
a=mid:1
a=candidate:1 1 UDP 1658497328 192.168.100.33 5000 typ host
a=candidate:2 1 UDP 1658497328 96.1.2.3 5000 typ srflx
a=mid:2
a=candidate:2 1 UDP 1658497328 96.1.2.3 5002 typ srflx
a=end-of-candidates
```

A.2. ICE Lite and Candidate Signalling

ICE Lite implementations of Trickle ICE do not necessarily need to receive candidates from their peers: it would be enough for them to start getting checks and discover all candidates as peer reflexive. This would allow to reduce signalling traffic but could introduce other issues.

A.3. Handling new candidates after check completion

Basically, do any new candidates discovered after ICE completes (e.g. from a new interface that comes up) need an ICE restart, or not?

Appendix B. Changes From Earlier Versions

Note to the RFC-Editor: please remove this section prior to publication as an RFC.

B.1. Changes From draft-rescorla-01

- Brought back explicit use of Offer/Answer. There are no more attempts to try to do this in an O/A independent way. Also removed the use of ICE Descriptions.
- Added SDP specification for trickled candidates, the trickle option and 0.0.0.0 addresses in m-lines, and end-of-candidates.
- Support and Discovery. Changed that section to be less abstract. As discussed in IETF85, the draft now says implementations and usages need to either determine support in advance and directly use trickle, or do half trickle. Removed suggestion about use of discovery in SIP or about letting implementing protocols do what

they want.
- Defined Half Trickle. Added a section that says how it works. Mentioned that it only needs to happen in the first o/a (not necessary in updates), and added Jonathan's comment about how it could, in some cases, offer more than half the improvement if you can pre-gather part or all of your candidates before the user actually presses the call button.
- Added a short section about subsequent offer/answer exchanges.
- Added a short section about interactions with ICE Lite implementations.
- Added two new entries to the open issues section.

B.2. Changes From draft-rescorla-00

- Relaxed requirements about verifying support following a discussion on MMUSIC.
- Introduced ICE descriptions in order to remove ambiguous use of 3264 language and inappropriate references to offers and answers.
- Removed inappropriate assumption of adoption by RTCWEB pointed out by Martin Thomson.

Authors' Addresses

Emil Ivov
Jitsi
Strasbourg 67000
France

Phone: +33 6 72 81 15 55
Email: emcho@jitsi.org

Eric Rescorla
RTFM, Inc.
2064 Edgewood Drive
Palo Alto, CA 94303
USA

Phone: +1 650 678 2350
Email: ekr@rtfm.com

Justin Uberti
Google
747 6th St S
Kirkland, WA 98033
USA

Phone: +1 857 288 8888
Email: justin@uberti.name