MMUSIC
Internet-Draft
Intended status: Standards Track
Expires: April 22, 2014

M. Thomson
Microsoft

October 19, 2013

Using Interactive Connectivity Establishment (ICE) in Web Real-Time Communications (WebRTC)
draft-thomson-mmusic-ice-webrtc-01

Abstract

Interactive Connectivity Establishment (ICE) has been selected as the basis for establishing peer-to-peer UDP flows between Web Real-Time Communication (WebRTC) clients. Using an unmodified ICE implementation in this context enables the use of the web platform as a denial of service platform. The risks and complications arising from this choice are discussed. A modified algorithm for sending ICE connectivity checks from the web platform is described.

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 April 22, 2014.

Copyright Notice

Table of Contents

1. Introduction

ICE [RFC5245] describes a process whereby peers establish a bi-directional UDP flow. This process has been adopted for use in Web Real-Time Communications (WebRTC) for establishing flows to and from web browsers ([I-D.ietf-rtcweb-overview]).

Properties of ICE are also critical to the security of WebRTC (see Section 4.2.1 of [I-D.ietf-rtcweb-security]).

The design of RFC 5245 does not fully consider the threat models enabled by the web environment. In particular, the following assumptions are not valid in a web context:

- A one-time consent to communicate is sufficient, and revocation of consent is not necessary.
- Signaling and control originates from actors that always operate in good faith.
- Only one ICE processing context operates at the one time.

Implementations of ICE that are technically compliant with the algorithm described in RFC 5245 potentially expose controls to web applications that can be exploited.

In the web context, an attacker is able to provide code (usually JavaScript) that is executed by those hosts in a sandbox. The protections of the sandbox are critical, both for protecting the host running the sandbox, and for protecting the Internet as a whole from bad actors.

The exposure of ICE features in the web browser could allow attackers to generate denial of service (DoS) traffic far in excess of the bandwidth needed to deploy the JavaScript. A small (1KB) file can potentially generate many megabytes of connectivity checks in a short period, representing an amplication factor far greater than other similar amplification attacks (for instance, DNS reflection attacks).

Mounting this sort of DoS attack does not rely on anything other than inducing a host to download and execute JavaScript. This is generally very easy to accomplish, making it very easy to conscript large number of traffic sources.

The issue regarding the one-time consent to communicate has already been identified as a serious problem for WebRTC. [I-D.muthu-behave-consent-freshness] describes a limit on the time that consent remains valid, requiring that communications consent be continuously refreshed.

This document first describes the characteristics of ICE as they relate to the web and the way that these characteristics can be exploited. In order to address the issues arising from allowing web application to initiate and control ICE processing, a modified algorithm is described, plus additional measures that can be employed to reduce the amount of traffic an attacker can produce.

1.1. Conventions and Terminology

In cases where normative language needs to be emphasized, this document falls back on established shorthands for expressing interoperability requirements on implementations: the capitalized words "MUST", "MUST NOT", "SHOULD" and "MAY". The meaning of these is described in [RFC2119].

2. ICE in a Web Browser

A web browser provides an API that applications can use to instantiate and control an ICE agent. The web application is responsible for providing the ICE agent with signaling that it might need to operate successfully, as well as configuration information regarding TURN [RFC5766] or STUN [RFC5389] servers.

In the web context, a browser treats the web application as being potentially hostile, providing access to features in a controlled fashion. Therefore, some of the information that an ICE agent might depend on in other contexts has to be regarded as potentially suspect when provided by a web application.

2.1. Factors Influencing DoS Capacity

There are several parameters that affect the characteristics of DoS attacks that can be mounted using ICE. These include:

- The number of candidate pairs that are created. An attacker can add extra remote candidates to inflate this number to tbe maximum supported. RFC 5245 recommends a default maximum of 100 candidate pairs. Reducing this limit directly reduces DoS potential, though it could affect success in some legitimate scenarios (see the calculations in Appendix A).

- The time between consecutive connectivity checks. Pacing of checks is discussed at length in Section 2.1.1.

- The total number and timing of retransmissions for each candidate pair. Section 2.1.2 discusses the implications of retransmissions.

- The size of connectivity check packets. Size considerations are described in Section 2.1.3.

- The number of ICE agents that can be operated concurrently. RFC 5245 does not consider scenarios like WebRTC where it is not only possible for there to be multiple agents. The web security model allows for cases where multiple agents can be created

concurrently, often with a further restriction that a browser not leak information between agents.

2.1.1. Pacing of Connectivity Checks

ICE [RFC5245] describes a scheme for pacing connectivity checks. There are two primary reasons that are cited:

- Pacing the initial connectivity checks for a given candidate pair allows middleboxes sufficient time to establish bindings. Empirical evidence suggests that failing to allow at least 20 milliseconds between initial connectivity checks risks the bindings being dropped at some middleboxes.

- Pacing limits the potential for connectivity checks to generate network congestion. Section 16.1 of [RFC5245] describes a formula for calculating the time between connectivity checks (Ta) that is based on the expected bandwidth of the real-time session that is being established.

In the web context, information about the expected bandwidth used by the session comes from the web application. Since the web application has to be regarded as potentially malicious, information about expected media bandwidth cannot be used to determine the pacing of connectivity checks. A fixed minimum interval between connectivity checks becomes the primary mechanism for limiting the ability of web applications to generate packets that are potentially congestion inducing.

Increasing the pacing interval directly reduces the amount of congestion that connectivity checks can generate, though this only reduces the peak bitrate that can be induced - the same amount of traffic is generated over a longer period. The cost of this is extended session setup times, where recent efforts have been focused on reducing this time.

2.1.2. Retransmission of Connectivity Checks

The initial retransmission timer (RTO) can also be increased with similar effect to increasing the pacing timer. Furthermore, there is a strong desire to reduce the recommended value of the RTO in ICE from 500 milliseconds to values more reflective of common round trip times in well-connected locations, which might be as low as 50 milliseconds.

More relevant is the total number of connectivity check retransmissions that an implementation attempts for each candidate pair. Each additional retransmission directly increases the duration

and magnitude of a DoS attack. Following the exponential backoff recommended by RFC 5245 does extend the time between retransmissions, which could reduce the rate of connectivity checks after several retransmissions, but this depends on the initial retransmission time out (RTO).

Reducing the number of retransmissions has the effect of reducing the probability of the check succeeding. The selection of a total retransmission count is a trade-off of success rates against the potential for abuse.

2.1.3. Connectivity Check Size

As currently specified, an attacker is only able to influence the size of the USERNAME attribute. [RFC5389] restricts USERNAME to a maximum size of 512 octets; the Session Description Protocol (SDP) signaling described in [RFC5245] limits the size of the username fragment an attacker can set to 256 bytes.

A browser could reduce its username fragment to as little as 4 bytes, limiting the overall size of the attribute to 261 bytes. A small username fragment does limit the collision resilience of the field, which is a property that is important for detecting other forms of attack (see Section 5.7.3 of [I-D.ietf-rtcweb-security-arch]).

There is also the potential for new modifications to ICE that increase the packet size. For instance [I-D.martinsen-mmusic-malice] provides an attacker with direct control over the bytes that are included in connectivity checks.

2.2. Denial of Service Magnitude

A malicious application is able to influence connectivity checking by altering the set of remote candidates and by changing the remote username fragment. The default maximum sizes for remote username fragment (256 bytes) and number of candidate pairs (100) described in RFC 5245 can be exploited by an attacker to increase the number and size of packets. Assuming an inter-check timer of the minimum of 20 milliseconds, plus a minimal 28 bytes of IPv4 and UDP overhead, this results in an attacker being able to induce approximately 144kbps for every ICE agent it is able to instantiate.

This rate is significantly higher than the minimal rate of 20kbps that a typical compressed voice stream generates. By comparison, a G.711 audio stream, which cannot be rate limited in response to network congestion, but is generally regarded as safe to send to a willing target, generates about 74kbps.

ICE does not allow for any congestion feedback (other than ECN [RFC3168]), so this rate could conceivably be sustained for some time, though after several seconds the time between retries increases, reducing the check rate unless the application is able to instantiate another ICE agent.

Some existing ICE implementations could generate about 3 or more times the basic rate of connectivity checks over a short period. These implementations do not pace retransmission of connectivity checks, resulting in significantly higher connectivity check rates during early rounds of retransmission.

> These implementations are ignoring the advice on calculating a minimum RTO from Section 16.1 of [RFC5245]. However, the shorter RTO allows ICE to complete much faster, which is a significant advantage.

Implementations that do not limit the number of ICE agents that can be instantiated, and subsequently fail to enforce rate limits globally create a further multiplicative factor on the basic rate.

3. Modified ICE Algorithm

This section describes an algorithm that ensures proper global pacing of connectivity checks. This limits the ability of any single attacker to generate a high rate of connectivity checks. This only limits the peak data rate that results from connectivity checks, reducing the intensity of DoS attacks.

Measures that reduce the overall duration of attacks are described in Section 4.

The modified algorithm for ICE does not alter the way that candidate pairs are selected, prioritized, frozen or signaled. It only affects the generation of connectivity checks. This algorithm affects candidate pairs in either of the "Waiting" or "In-Progress" states only (see Section 5.7.4 of [RFC5245]).

The ICE agent maintains two queues for candidate pairs.

waiting queue: The first is a prioritized list of candidate pairs in the "Waiting" state. The waiting queue is simply a prioritized list of all the candidate pairs in the check list (see Section 5.7 of [RFC5245]) that are in the "Waiting" state. As candidate pairs enter the "Waiting" state, they are added to the waiting queue. As each candidate pair is added, it is prioritized relative to all the other candidate pairs in the waiting queue.

check queue: The second is for outstanding connectivity checks. Each entry in this list represents a connectivity check for a given candidate pair. Each entry also includes a counter representing the number of connectivity checks that have been sent on this candidate pair.

The ICE agent maintains two types of timer: a pacing timer and a retransmission timer. There is only one pacing timer, though there can be multiple retransmission timers running concurrently.

The first candidate pair that arrives in the waiting queue starts the pacing timer. The pacing timer runs as long as there are items in any queue, ending if the timer expires when there are no entries in either queue. The pacing timer resumes if an entry is added to either queue and the timer is not already running.

Each time the pacing timer expires, the ICE agent performs the following steps:

1. If there are items on the waiting queue, but no items on the check queue, the first candidate pair is taken from the waiting queue.

    a. The candidate pair transitions from "Waiting" to "In-Progress".

    b. A check counter is associted with the candidate pair, initialized with a zero value.

    c. The candidate pair is added to the check queue. This could result in a connectivity check being sent immediately if the check queue is currently empty.

2. If there are items in the check queue, the ICE agent removes the first item and performs a connectivity check on the identified candidate pair.

    a. The check counter associated with the candidate pair is incremented by one.

    b. Based on the value of the check counter, a retransmission timer is scheduled for the candidate pair. The retransmission timer is not scheduled if the check counter exceeds the maximum number of checks configured for the ICE agent.

   c. If the retransmission timer expires without the connectivity check succeeding, the candidate pair is returned to the end of the check queue along with the higher check counter.

   d. The retransmission timer is cancelled if the connectivity check succeeds. The process for handling successful checks in Section 7.1.3.2 of [RFC5245] is followed.

3. If no connectivity checks were sent, the pacing timer is stopped.

An important characteristic of this algorithm is that it - as much as possible - prefers retransmission of connectivity checks over the initiation of new connectivity checks. This ensures that once an initial connectivity check has established any necessary middlebox bindings, subsequent retries are not delayed excessively, which could cause the binding to time out. However, the global pacing can cause the time between retransmission of connectivity checks to be extended as the check queue occasionally fills.

Favoring retransmission over initial checks directly contradicts the guidance on RTO selection in Section 16.1 of [RFC5245]. This is necessary due to the delays induced by potential interactions between multiple ICE agents, which might otherwise cause retries to be significantly delayed. Improvements to candidate prioritization are expected to reduce the impact of this change.

3.1. Trickled and Peer Reflexive Candidates

Trickled ICE candidates [I-D.ivov-mmusic-trickle-ice] generate candidate pairs after connectivity checking has commenced. In order to avoid trickled candidates negatively affecting the chances of a connectivity check succeeding, connectivity checks on newly appearing candidate pairs must be prioritized below any existing connectivity check.

Trickled candidates are in many respects identical to peer reflexive candidates. Both arrive after the algorithm has commenced.

In either case, as new candidates arrive (or are discovered), they are paired as normal (Section 5.7.1 of [RFC5245]), and - if appropriate - entered into the "Waiting" state. This causes the candidate pair to enter the waiting queue. Candidate pairs in the waiting queue are not ordered based on arrival time, they are ordered based on priority alone.

Trickling regular candidates does introduce the potential for a mismatch in the ordering of candidate pairs between peers, since trickled candidates will appear in the sending side well before the

receiving side can act upon them, resulting in the sending peer potentially commencing checks much earlier than the receiving peer. This is particularly important given the possibility that retransmissions of connectivity checks can block the progress of a candidate pair from the "Waiting" state into the "In-Progress" state, resulting in potentially large differences in the commencement time for any given candidate pair.

A trickle ICE implementation MAY choose not to immediately enqueue local candidates as they are discovered to allow some time for trickle signaling to propagate in order to increase the probability that checks remain synchronized.

3.2. Multiple ICE Agents

In a system that has potentially more than one ICE agent, it's important that connectivity checks from any given ICE agent cannot be blocked or starved by other ICE agents. It is also important that an attacker is unable to circumvent any limits by instantiating multiple ICE agents.

To that end, a single pacing timer is maintained globally whenever multiple ICE agents are operated. Each time the pacing timer fires, the global context selects ICE agents in a round-robin fashion. In addition to ensuring a global rate limit, this selection method ensures that no single ICE agent is completely starved.

In a shared context, ICE agents do not stop or start the pacing timer unless they are the first or last ICE agent to be active. The first ICE agent to commence checking starts the global timer, the last ICE agent to cancel the timer causes the global timer to be cancelled. At all other instances, "starting" the pacing timer for an ICE agent simply adds the ICE agent to the set of agents that can be selected; "stopping" the pacing timer removes the ICE agent from the set of ICE agents that are in consideration.

A global pacing timer causes each individual ICE agent to execute checks more slowly than a lone ICE agent would. Where there are many candidate pairs to test, this could have a negative impact on the synchronization of checks between peers. Poor check synchronization can have a negative impact on success rates. Peers with asymmetric contention can have lower priority candidate pairs started on the less contended peer long before the contended peer is able to commence checking, which can result in those checks failing.

Several measures are suggested for mitigating the impact of contention: artificial contention, origin-first distribution, inter-

agent candidate pair freezing, and delayed start. However, it is important to note that similar artificial constraints have classically been quickly circumvented on the web if they have overly negative performance consequences.

3.2.1. Introducing Artificial Contention

In cases where there is zero contention, artificial contention can be introduced to ensure a certain minimum effective pacing timer. In effect, this would increase the basic pacing timer from 20ms by a minimum multiple for any single ICE agent. Artificially contention would result in no checks being sent at all at different phases, spacing genuine connectivity checks.

For instance, contention could be increased to a minimum of 3 ICE agents. Assuming a 20ms basic interval, the first ICE agent would be able to send connectivity checks every 60ms, as though it were contending with two other ICE agents. Adding another ICE agent would have no effect on this rate. It would only be if a fourth ICE agent were added that all ICE agents would be reduced to sending checks at 80ms intervals.

This has the advantage of ensuring that a lightly contended client has the same rate of checking as a client with only a small number of ICE agents so that checks are more likely to be synchronized.

3.2.2. Origin-First Round-Robin

In a system such as a browser, there are potentially competing interests sharing the same limited resources. In this type of context, each competing user - in the browser, this is an origin [RFC6454] - can first be selected using a round-robin or similar allocation scheme.

Thus, as a first step, selection is performed from the set origins that have an active ICE agent. Once an origin is selected, agents are selected from within that origin. This ensures that no single origin can receive more than a proportional share of the access to connectivity checking.

This is particularly important if multiple users (or origins) are each able to create multiple ICE agents. Selecting based on users first prevents a single origin from monopolizing access to connectivity checks.

3.2.3. Inter-Agent Candidate Pair Freezing

In some cases, it might be necessary to instantiate multiple ICE agents from the same application, between the same two peers. An ICE agent MAY place candidate pairs in the "Frozen" state based on candidate pairs with the same foundation being "Waiting" or "In-Progress" on another ICE agent. This reduces the overall demand for connectivity checks without any significant negative effect on the chances that ICE succeeds.

In the browser context, information about the success of connectivity checks cannot leak between different domains. This could allow information about activities on another tab to be leaked, violating the origin security model of the browser. Thus, any inter-agent freezing logic MUST be constrained to ICE agents that operate in the same origin.

3.2.4. Delayed ICE Agent Start

In cases where there is high contention for access to connectivity checking, it might be preferable to delay the start of connectivity checks for an ICE agent rather than have the effective pacing timer increased.

4. Further Reducing the Impact of Attacks

A global pacing timer allows a web application to determine whether another domain is currently establishing an ICE transport, simply by observing the pacing of connectivity checks that it requests. Section 3.2.1 describes a method that allows a limited number of ICE agents to operate without being detectable.

The algorithm and the measures it describes are based on an assumption that ICE agents are created legitimately. Even with these measures, it’s possible to generate a steady amount of bandwidth toward arbitrary hosts. The remainder of this section is dedicated to additional measures that might be employed to reduce the ability of malicious users to generate unwanted connectivity checks over time.

4.1. Bandwidth Rate Limiting

A measure of the bandwidth generated by connectivity checks can be maintained, on both global and a per-origin basis. As this number increases, the browser can reduce the rate of connectivity checks. This reduction might either be gained by increasing the duration of the pacing timer or skipping occasional connectivity checks.

Appendix A includes some simple calculations and recommendations on what might be appropriate limits to set on the bandwidth used by connectivity checks.

4.2. Malicious Application Penalties

An attacker that only wishes to generate traffic is unlikely to provide valid candidates for two reasons:

- a successful connectivity check is likely to cause the ICE agent to terminate further checking
- serving connectivity checks requires the dedication of greater resources by the attacker

A long sequence of unsuccessful connectivity checks is therefore a likely indicator for an attack. An ICE agent could choose to reduce the rate at which connectivity checks are generated for an application that has a large number of failed checks.

Any measure that penalizes for unsuccessful checks will have to allow for some failures. Even legitimate uses of ICE can result in significant numbers of failed connectivity checks. For instance, an implementation that exclusively prioritizes IPv6 over IPv4 on a network with broken IPv6 will legitimately see a large number of failures. Similarly, if a remote peer is behind a NAT, prior to the commencement of checking by that peer all connectivity checks are likely to be discarded by the NAT.

4.3. Limited Concurrent Access to ICE

Setting an absolute maximum on the number of ICE agents that can be instantiated could overly constrain legitimate applications that depend on having multiple active sessions. However, limiting concurrent access to active ICE agents by delaying the start of connectivity checking, as described in Section 3.2.4 might allow an implementation to reduce the ability of a single origin to generate unwanted connectivity checks.

5. Negotiating Algorithm Use

The algorithm defined in Section 3 could cause some ICE agents to perform checks in a very different order to the order of an unmodified ICE agent. Failing to coordinate when checks occur reduces the probability that ICE is successful.

TODO: Determine whether an ice-options token that enables negotiation of this algorithm is appropriate, or whether something more

definitive is required, since an answerer could negotiate an ice-options token away. Note that WebRTC implementations probably won't be able to accept a session that does not use this algorithm.

6. Security Considerations

This entire document is about security.

7. Acknowledgements

The bulk of the algorithm described in this document came out of a discussion with Emil Ivov and Pal-Erik Martinsen. Eric Rescorla and Bernard Aboba provided some feedback regarding the DoS considerations and possible mitigations.

8. References

8.1. Normative References

[I-D.ivov-mmusic-trickle-ice]
Ivov, E., Rescorla, E., and J. Uberti, "Trickle ICE: Incremental Provisioning of Candidates for the Interactive Connectivity Establishment (ICE) Protocol", draft-ivov-mmusic-trickle-ice-01 (work in progress), March 2013.

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

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

8.2. Informative References

[I-D.ietf-rtcweb-overview]
Alvestrand, H., "Overview: Real Time Protocols for Brower-based Applications", draft-ietf-rtcweb-overview-08 (work in progress), September 2013.

[I-D.ietf-rtcweb-security-arch]
Rescorla, E., "WebRTC Security Architecture", draft-ietf-rtcweb-security-arch-07 (work in progress), July 2013.

[I-D.ietf-rtcweb-security]
Rescorla, E., "Security Considerations for WebRTC", draft-ietf-rtcweb-security-05 (work in progress), July 2013.

[I-D.martinsen-mmusic-malice]
Penno, R., Martinsen, P., Wing, D., and A. Zamfir, "Metadata Attribute signaLling with ICE", draft-martinsen-mmusic-malice-00 (work in progress), July 2013.

[I-D.muthu-behave-consent-freshness]
Perumal, M., Wing, D., R, R., and T. Reddy, "STUN Usage for Consent Freshness", draft-muthu-behave-consent-freshness-04 (work in progress), July 2013.

[RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, September 2001.

[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.

[RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, December 2011.

Appendix A. Defining Legitimate Uses of ICE

Limiting the bandwidth generated by connectivity checks depends on knowing how much ICE could use under normal circumstances. This ensures any absolute limit doesn't adversely affect a legitimate use of ICE.

Any calculation should allow for slightly abnormal configurations that might generate higher than average data rates. Otherwise, an average might adversely affect legitimate users. The intent is to avoid having legitimate uses concerned with the limit.

A.1. Candidate Pair Count

Our sample legitimate user has 2 local network interfaces. This can result in as many as 14 candidates, 8 of them IPv4 plus 6 IPv6. Each interface has 1 IPv4 address, an IPv6 address, plus a link-local IPv6 address. Assuming a different public IPv4 NAT address for each interface and IP version (using either NAT4-4 or NAT6-4 as appropriate) other than the link local addresses, this adds another 4 addresses. In addition to this, two TURN servers might be contacted by either IPv4 or IPv6, providing 4 more addresses.

Two peers with this configuration will generate 100 candidate pairs, since only IPv4 candidates are paired with IPv4 candidates.

Assuming that all candidates are checked once before ICE completes on a second round of checks, there are in excess of 100 connectivity checks sent. Even at the fastest permitted pacing, this means that ICE completes in at least 2 seconds, plus the round trip time.

A.2. Connectivity Check Size

The STUN message used for a connectivity check can vary, but making some reasonable assumptions, it is likely to be 149 or 169 bytes on the wire (plus network layer encapsulation). This makes the following assumptions:

IP Header: 20 bytes (IPv4) or 40 bytes (IPv6) with no extensions

UDP Header: 8 bytes

STUN Header: 20 bytes

USE-CANDIDATE Attribute: 4 bytes

CONTROLLED or CONTROLLING Attribute: 4 bytes

PRIORITY Attribute: 4 bytes

MESSAGE-INTEGRITY Attribute: 24 bytes

FINGERPRINT Attribute: 8 bytes

USER Attribute: 49 bytes carries two 20 character username fragments

A.3. Rate Calculations

Assuming a 150 byte connectivity check and a global pacing timer of 20ms, this produces 60kbps at peak (68kpbs for IPv6).

For 100 candidate pairs, with at most 5 connectivity checks on each pair, this peak could be sustained for 10 seconds by a single ICE agent.

The question is: is this a tolerable rate?

A.4. Comparison: G.711 Audio

G.711 audio is commonly used without any congestion feedback mechanisms in place - primarily because it is unflexible and unable

to scale its network usage in response to congestion signals. The theory is that it might be acceptable to generate a similar amount of traffic without congestion controls.

It should be immediately obvious that this theory has a major flaw. Even though the impact on the network might be similar, G.711 is not sent to an unwilling recipient, whereas no such guarantee can be made for connectivity checks.

Assuming 80bit integrity on SRTP, no header extensions and no CSRCs, G.711 produces 84kbps. That would suggest that a single ICE agent with 20ms pacing might be tolerable, at least over short intervals.

A.5. Recommended Rate Limits

Enforcing a limit of 96kbps would allow for a substantial increase in the size of STUN connectivity check messages without affecting legitimate uses.

Over a longer interval, this high rate is likely to be unnecessary. Even with 100 candidate pairs, ICE should complete in between 2 and 5 seconds, especially if candidate pairs are frozen across multiple ICE agents. Providing a lower limit over a 10 to 20 second interval should further limit the damage. Enforcing a longer term limit of 48 kilobytes (every 20 seconds or so) would allow for 6 seconds of continuous checking with the size described above, or 4 seconds of checking at the short term rate limit.

Author's Address

Martin Thomson
Microsoft
3210 Porter Drive
Palo Alto, CA 94304
US

Email: martin.thomson@skype.net