Abstract

This document describes a minimal set of requirements to enable NAT traversal (and satisfy one of the security requirements) for media channels within browser-based real-time communications (RTCWEB).

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 December 8, 2011.

Copyright Notice

Copyright (c) 2011 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.
Table of Contents

1. Introduction .............................................. 3
2. Drawbacks of a Full ICE Implementation .................. 3
   2.1. Limits Adaptability .................................. 3
   2.2. Hampers Innovation .................................. 4
   2.3. Unnecessary Cost in some Cases ...................... 4
3. Proposed Model ........................................... 5
4. Proposed API .............................................. 6
   4.1. Before Test ......................................... 6
   4.2. Send Test ........................................... 6
   4.3. Receipt of Test ...................................... 7
   4.4. Receipt of Response .................................. 7
5. Security Considerations .................................... 7
6. Informative References .................................... 8
Authors’ Addresses ........................................... 8
1. Introduction

RTCWEB clients - including, but not limited to web browsers - should be able to send and receive real-time media directly to and from other RTCWEB clients without sending the media through an application-layer intermediary. This will serve to reduce media latency, decrease packet loss, and reduce the operational cost of deploying the application.

There is general agreement that Interactivity Connectivity Establishment (ICE) RFC 5245 [RFC5245] represents a reasonable choice for meeting this need. ICE provides firewall and NAT traversal, creating direct peer-to-peer connections for media when possible, and falling back to media relays (typically established with TURN) when not possible.

Consequently, the natural inclination is to simply embed a full ICE implementation inside of the browser. However, there are drawbacks to doing so. This document proposes an alternative model, based on the concept of browser minimalism - embedding only the minimum necessary functionality into the browser itself, and then allowing application developers flexibility to use those tools as needed.

Section 2 first discusses the drawbacks of a full ICE implementation in the browser. Section 3 then outlines an alternative model where only STUN is present in the browser, and argues why it addresses the limitations discussed in Section 2. Section 4 then proposes a concrete extension to the PeerConnection API to enable STUN.

2. Drawbacks of a Full ICE Implementation

There are several drawbacks to including a full ICE implementation in the browser.

2.1. Limits Adaptability

ICE was not the IETF's first attempt at techniques for firewall and NAT traversal. Basic STUN [RFC3489] was defined in 2003, and it solved the problem by attempting to characterize NATs. It failed for a variety of reasons. However, one of the key lessons of STUN was that its technique for classifying NATs - breaking them into four different NAT varieties - proved brittle. In reality, the market saw changes in the types of implementations, and NATs appeared which met none of the classifications. For this reason, ICE abandoned the classification approach and instead moved towards a model of connectivity checking.
As a consequence, ICE has greater reliability than pure STUN, but its effectiveness in achieving direct p2p connections is still based on some underlying assumptions around NAT types. Its design is most effective for NATs whose behavior is endpoint-independent mapping, and whose filtering policy is either endpoint-independent or address-dependent [RFC4787].

With the ongoing exhaustion of the IPv4 address space, we can anticipate even further reliance on NAT and the likely appearance of carrier NATs of differing varieties. This is likely to change the nature of NAT behaviors seen in the real world. The right way to deal with this is to adapt ICE’s behavior, using differing allocation techniques and assigning different priorities. For example, ICE currently does not enable direct p2p connections in cases where NATs have mapping policies which are endpoint dependent but utilize sequential port allocation. If, despite the recommendations of RFC4787, such NAT types become increasingly prevalent, ICE’s effectiveness will decline and more connections will be relayed. With ICE literally baked into web browsers, it will become harder to adapt its algorithms to work best under the conditions of the modern Internet.

2.2. Hampers Innovation

One of the benefits of ICE is that it allows local implementation flexibility in the way candidates are gathered, offered and prioritized. However, once ICE is baked into the browser, it is no longer possible for that innovation to take place - or at least, it leaves the hands of the voice application providers. To date, there has been variability in this aspect of implementation, with different providers tuning it to tweak their needs and deployments.

2.3. Unnecessary Cost in some Cases

There is a broad array of use cases for VoIP. It is used for everything from consumer Internet services (like Skype) to small business phone systems. Though clearly global consumer Internet services require the kind of traversal technology provided by full ICE, it is not needed in other cases. One such use case is, in fact, enterprise telephony, where users make calls within the confines of their corporate network, and remote access is supported through VPN. Today, VoIP endpoints in these environments do not generally use ICE.

As such, if an enterprise communications application wanted to utilize browser RTC, it would need to support ICE even though it was not strictly required. Is there a penalty to support of ICE? The enterprise would need to deploy STUN and TURN servers, which would not actually be needed. ICE also typically increases call setup
delay (though the degree to which it does it is dependent on the network conditions the users are in), those increases would be for no benefit in the enterprise deployment scenario.

3. Proposed Model

The model proposed here is that the browser itself support STUN only. APIs are provided which allow for initiation of a STUN transaction. The results of this transaction are then passed to the browser application (notably, the reflexive address). The browser API allows the browser application to set attribute/value pairs in the message. Similarly, on the receive-side, APIs are defined for allowing an application to register callbacks for receipt of a STUN request. Those callbacks provide the application information on the source IP and port, amongst other information.

For security purposes, the browser will refuse to send, or accept, media to or from a peer to which a STUN transaction has not completed successfully. This ensures that the browser cannot be used as a DoS tool to launch a voice hammer attack.

What about TURN? In this model, TURN is mostly implemented on top of the browsers STUN implementation. The Javascript code in the browser can generate Allocate requests, and be informed of the results. The only exception to this is that the browser has to be told whether or not to encapsulate media in Send transactions, or to use an allocated channel. The browser API provides a switch which allows the application to tell the browser which encapsulation to use for media.

In a server-mediated environment, TURN might also be unnecessary. A call setup service can communicate directly with the relay service to establish a transparent UDP tunnel through one or more relays, the STUN connectivity checks may be sent through this tunnel, and no TURN encapsulation support is needed in the browser. The Javascript-initiated STUN connectivity tests may also be used to authenticate the browser to the tunnel service.

With this model, there is now a great deal of flexibility in how NAT traversal can be done. Some of the models which can now be supported are:

ICE in Javascript: A full ICE implementation is possible in Javascript itself. Because the implementation resides in Javascript, it is trivially changed at any time.
Server-Based ICE: A full ICE implementation can execute in the server, using remote-control commands to inform the browser to send STUN transactions, and passing the results from the browser back to the server. In essence - MGCP for ICE.

STUN-Only: For deployments where the peer is always publically reachable from clients - such as enterprises or PSTN termination services - the Javascript can do a single STUN transaction to create a permission in the browser, and then proceed to send media.

Non-ICE: Protocols similar to ICE, but not otherwise compliant, can also be implemented. Negotiation of which NAT traversal mechanism is needed, is done by the application outside of the browser.

This model addresses all of the concerns outlined in Section 2. Now, if changes in NAT types occur over time, new Javascript or server code can be deployed which uses different prioritizations, or even performs new allocation models. For example, port-predictive allocations can be added in this model, without upgrading the browser. Since the browser has the barest minimum necessary for security and functional purposes, innovation is possible to a greater degree. Finally, implementations can be only as complex as is needed for the task at hand.

4. Proposed API

The following makes the assumption that a PeerConnection object exists and is bound to a single local UDP port.

4.1. Before Test

There must be an API which allows the PeerConnection’s local credentials to be determined, and a way to send these via a signaling service to the other party. The browser SHOULD generate the credentials itself and provide an API for read-only access.

4.2. Send Test

A function must be provided in order to initiate a STUN connectivity test. This function MUST allow the specification of a far address and port number, the far username and password, and (if necessary) additional ICE attributes to be included in the STUN Binding Request message. This function causes a single STUN [RFC 3489] (RFC3489) Binding Request with short-term credentials to be sent to the far address from the initiating client. The client MUST enforce a rate limit of the transmission of these requests. Username concatenation
is performed as per ICE 7.1.2.3. The client (in the case of a browser) MUST NOT allow the user of the API to specify or examine the transaction ID for this request, in order to prevent spoofing of successful replies from an attacking host. Using a STUN request ensures that the packet will begin with the STUN magic cookie, and therefore is very unlikely to simulate other traffic. The STUN request is sent from the exact same IP address and port that the PeerConnection object will use for subsequent media traffic.

4.3. Receipt of Test

Upon receipt of a STUN Binding Request with valid credentials, the responding client SHOULD automatically generate and send the STUN transaction response. (If it does not, an API for sending the transaction response MUST be provided.) The responding client MUST also locally call a callback function that delivers the attribute/value pairs received in the Binding Request as well as the locally derived (reflexive) address from which the Binding Request was received.

4.4. Receipt of Response

Upon receipt of a valid STUN transaction response from the responding client, the initiating client MUST call a callback function that delivers the attribute/value pairs received in the response, one of which is the reflexive address. The response MUST be ignored if the receivedSocketAddress does not match the socket address to which the matching transaction ID was sent, as per ICE 7.1.3.2. Upon receipt of a valid response the client also adds the now-verified address to the Transmit Whitelist, a list of socket addresses to which sending of media is now permissible. The client MUST NOT allow media to be sent to any address/port combination that has not been added to the Transmit Whitelist. Note that the client must appropriately time out any state associated with pending tests.

5. Security Considerations

The Transmit Whitelist function serves to prevent a client from sending media to an endpoint which has not properly responded to a STUN request.

The requirement that the client internally generate the transaction ID and not allow it to be explicitly set or read back prevents spoofing of the STUN test replies.
6. Informative References


Authors’ Addresses

Matthew Kaufman
Skype
3210 Porter Drive
Palo Alto, California 94304
US
Phone: +1 831 440 8771
Email: matthew.kaufman@skype.net

Jonathan Rosenberg
Skype
3210 Porter Drive
Palo Alto, California 94304
US
Email: jdrosen@skype.net