Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: July 24, 2014

T. Stach
A. Hutton
Unify
J. Uberti
Google

January 20, 2014

RTCWEB Considerations for NATs, Firewalls and HTTP proxies
draft-hutton-rtcweb-nat-firewall-considerations-03

Abstract

This document describes mechanism to enable media stream establishment for Real-Time Communication in WEB-browsers (WebRTC) in the presence of network address translators, firewalls and HTTP proxies. HTTP proxy and firewall deployed in many private network domains introduce obstacles to the successful establishment of media stream via WebRTC. This document examines some of these deployment scenarios and specifies requirements on WebRTC enabled web browsers designed to provide the best possible chance of media connectivity between WebRTC peers.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

WebRTC is a web-based technique for direct interactive rich communication using audio, video, and data between two peer browsers.

Many organizations, e.g. an enterprise, a public service agency or a university, deploy Network Address Translators (NAT) and firewalls (FW) at the border to the public internet. WebRTC relies on ICE [RFC5245] in order to establish a media path between two WebRTC peers in the presence of such NATs/FWs.

When WebRTC is deployed by the corporate IT department one can assume that the corporate IT configures the corporate NATs, Firewalls, DPI units, TURN servers accordingly. If so desired by the organization WebRTC media streams can then be established to WebRTC peers outside of the organization subject to the applied policies. In order to cater for NAT/FWs with address and port dependent mapping characteristics [RFC4787], the peers will introduce a TURN server [RFC5766] in the public internet as a media relay. Such a TURN server could be deployed by the organization wanting to assert policy on WebRTC traffic.

However, there are also environments that are not prepared for WebRTC and have NAT/FW deployed that prevent media stream establishment although such blocking is not intentional. These environments include e.g. internet cafes or hotels offering their customers access to the web and have opened the well-known HTTP(S) ports but nothing else. In such an environment ICE will fail to establish connectivity. Re-configuration of the NAT/FW is also often impracticable or not possible.

In such an environment a WebRTC user may easily reach its WebRTC server possibly via an HTTP proxy and start establishing a WebRTC session, but will become frustrated when a media connection cannot be established. A corresponding use case and its requirements relating to WebRTC NAT/FW traversal can be found in [draft-ietf-rtcweb-use-cases-and-requirements].

The TURN server in the public internet is not sufficient to establish connectivity for RTP-based media [RFC3550] and the WebRTC data channel [draft-ietf-rtcweb-data-channel] towards external WebRTC peers since the FW policies include blocking of all UDP based traffic and allowing only traffic to the TCP ports 80/443 with the intent to support HTTP(S) [RFC2616].

We explicitly don’t address even more restricted environments, that deploy HTTP traffic validation. This could e.g. be done by means of DPI validation or traffic pattern analysis to determine the contents of the packets that the traffic is, in fact, HTTP or HTTPS-looking or by an HTTP proxy that breaks into the TLS exchange and looks for HTTP in the traffic. However we want to address the case when access to the World Wide Web from inside an organization is only possible via a transparent HTTP Proxy that just tunnels traffic after e.g. enforcing an acceptable use policy.

This document examines impact of NAT/FW policies in Section 2. Additional impacts due to the presence of a HTTP proxy are examined in Section 3.

1.1. Requirements Language

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 RFC 2119 [RFC2119].

2. Considerations for NATs/Firewalls independent of HTTP proxies

This section covers aspects of how NAT/FW characteristic influence the establishment of a media stream. Additional aspects introduced by the presence of a HTTP proxy are covered in Section 3.

If the NATs serving caller and callee both show port and address dependent mapping behavior the need for a TURN server arises in order to establish connectivity for media streams. The TURN server will relay the RTP packet to the WebRTC peer using UDP. How the RTP packets can be transported from the WebRTC client within the private network to the TURN server depends on what the firewall will let pass through.

Other types of NATs do not require using the TURN relay. Nevertheless, the FW rules and policies still affect how media streams can be established.

2.1. NAT/Firewall open for outgoing UDP and TCP traffic

This scenario assumes that the NAT/FW is transparent for all outgoing traffic independent of using UDP or TCP as the transport protocol. This case is used as starting point for introduction of more restrictive firewall policies. It presents the least critical example with respect to the establishment of the media streams.

The TURN server can be reached directly from within the private network via the NAT/FW and the ICE procedures will reveal that media can be sent via the TURN server. The TURN client will send its media to the allocated resources at the TURN server via UDP.

Dependent on the port range that is used for WebRTC media streams, the same statement would be true if the NAT/Firewall would allow UDP traffic for a restricted UDP port range only.

2.2. NAT/Firewall open only for TCP traffic

This scenario assumes that the NAT/FW is transparent for outgoing traffic only using TCP as transport protocol. Theoretically, this gives two options for media stream establishment dependent on the NAT’s mapping characteristics. Either transporting RTP over TCP directly to the peer or contacting a TURN server via TCP that then relays RTP.

In the first case the browser does not use any TURN server to get through its NAT/FW. However, the browser needs to use ICE-TCP [RFC6544] and provide active, passive and/or simultaneous-open TCP candidates. Assuming the peer also provides TCP candidates, a connectivity check for a TCP connection between the two peers should be successful.

In the second case the browser contacts the TURN server via TCP for allocation of an UDP-based relay address at the TURN server. The ICE procedures will reveal that RTP media can be sent via the TURN relay using the TCP connection between TURN client and TURN server. The TURN server would then relay the RTP packets using UDP, as well as other UDP-based protocols. ICE-TCP is not needed in this context.

Note that the second case is not to be confused with using TURN to request a "TCP Allocation" as described in [RFC6062], which deals with how to establish a TCP connection from a TURN server to the peer. For this document we assume that the TURN server can reach the peer always via UDP, possibly via a second TURN server, in case the WebRTC peer is located in a similar environment as described in this section.

We don’t see a need to request TCP allocations at the TURN server since it is preferable that WebRTC media is transported over UDP as far as possible. For the same reason we also prefer using TCP just as transport to the TURN server over using the ICE-TCP with an end-to-end TCP connection

2.3. NAT/Firewall open only for TCP on restricted ports

In this case the firewall blocks all outgoing traffic except for TCP traffic to specific ports, for example port 80 (HTTP) for HTTP or 443 for HTTPS(HTTPS). A TURN server listening to its default ports (3478 for TCP/UDP, 5349 for TLS) would not be reachable in this case. However, the TURN server can still be reached when it is configured to listen to e.g. the HTTP(S) ports.

In addition the browser needs to be configured to contact the TURN server over the HTTP(S) ports and/or the WebRTC client has to provide this information to browser.

3. Considerations for NATs/Firewalls in presence of HTTP proxies

This section considers a scenario where all HTTP(S) traffic is routed via an HTTP proxy. We assume that the HTTP proxy is tranparent and just tunnels traffic after e.g. enforcing an acceptable use policy with respect to domains that are allowed to be reached. We don’t consider cases where the HTTP proxy is used to deploy HTTP traffic validation. This includes DPI validation that the traffic is, in fact, HTTP or HTTPS-looking or a HTTP proxy that breaks into the TLS exchange and looks for HTTP in the traffic.

Note: If both WebRTC clients are located behind the same HTTP proxy, we, of course, assume that ICE would give us a direct media connection within the private network. We don’t consider this case in detail within this document.

3.1. HTTP proxy with NAT/Firewall open for outgoing UDP and TCP traffic

As in Section 2.1 we assume that the NAT/FW is transparent for all outgoing traffic independent of using UDP or TCP as transport protocol. The HTTP proxy has no impact on the transport of media streams in this case. Consequently, the same considerations as in Section 2.1 apply with respect to the traversal of the NAT/FW.

3.2. HTTP proxy with NAT/Firewall open only for TCP traffic

As in Section 2.2 we assume that the NAT/FW is transparent only for outgoing TCP traffic. The HTTP proxy has no impact on the transport of media streams in this case. Consequently, the same considerations as in Section 2.2 apply with respect to the traversal of the NAT/FW.

3.3. HTTP proxy with NAT/Firewall open only to proxy routed traffic

Different from the previous scenarios, we assume that the NAT/FW accepts outgoing traffic only via a TCP connection that is initiated from the HTTP proxy. Currently only the case of an explicit proxy is considered here.

This scenario is the most complex and controversial as it requires the WebRTC media to be tunneled through the proxy. However such techniques are already specified in RFC’s and deployed an example of this is websockets [RFC6455] which uses the HTTP CONNECT mechanism in the presense of HTTP Proxies.

This document discusses some alternative approaches to achieving connectivity for WebRTC media in this environment but does not currently make any firm recommendations as the alternatives are mostly work in progress in other areas of the IETF. Therefore it is not possible to make such a recommendation at this time.

4. Solutions for Further Study

The following sections outline and provide some analysis of various solutions to the issues raised regarding WebRTC media traversing firewalls and proxies. All of these potential solutions require further analysis by the IETF RTCWEB working group and in some cases may require work in other IETF working groups.

It is possible that due to different network environments that WebRTC browsers may need to implement more than one solution.

NOTE - THIS ANALYSIS IS NOT COMPLETE.

4.1. HTTP CONNECT based mechanism

A WebRTC browser could make use of the HTTP CONNECT method [RFC2817] and request that the HTTP proxy establishes a tunnel connection on its behalf in order to get access to the TURN server. The HTTP CONNECT request needs to convey the TURN Server URI or transport address. As a result the HTTP Proxy will establish a TCP connection to the TURN server and when successful the HTTP Proxy will answer the HTTP CONNECT request with a 200OK response. In case of a transparent proxy, the HTTP proxy will now switch into tunneling mode and will transparently relay the traffic to the TURN server.

By using the HTTP CONNECT method, the TURN server only has to handle a standard TCP connection. An update to the TURN protocol or the TURN software is not needed.

Afterwards, the browser could upgrade the connection to use TLS, forward STUN/TURN traffic via the HTTP proxy and use the TURN server as media relay. Note that upgrading in this case is not to be misunderstood as usage of the HTTP UPGRADE method as specified in [RFC2817] as this would require the TURN server to support HTTP. The following is a possible sequence of events:

o the browser opens a TCP connection to the HTTP proxy,

o the browser issues a HTTP CONNECT request to the HTTP proxy with the TURN server address in the Request URI, for example

   * CONNECT turn_server.example.com:5349 HTTP/1.1 Host: turn_server.example.com:5349

- the HTTP proxy opens a TCP connection to the TURN server and "bridges" the incoming and outgoing TCP connections together, forming a virtual end-to-end TCP connection,

- the browser can do a TLS handshake over the virtual end-to-end TCP connection with the TURN server.

Strictly speaking the TLS upgrade is not necessary, but using TLS would also prevent the HTTP proxy from sniffing into the data stream and provides the same flow as HTTPS and might improve interoperability with proxy servers. The WebRTC application has the ability to control whether TLS is used by the parameters it supplies to the TURN URI (e.g. turns: vs. turn:), so the decision to access the TURN server via TCP versus TLS could be left up to the application or possibly the browser configuration script.

In contrast to using UDP or TCP for transporting the STUN messages, the browser would now need to first establish a HTTP over TCP connection to the HTTP proxy, upgrade to using TLS and then switch to using this TLS connection for transport of STUN messages.

Further considerations apply to the default connection timeout of the HTTP proxy connection to the TURN server and the timeout of the TURN server allocation. Whereas [RFC5766] specifies a 10 minutes default lifetime of the TURN allocation, typical proxy connection lifetimes are in the range of 60 seconds if no activity is detected. Thus, if the WebRTC client wants to pre-allocate TURN ressources it needs to refresh TURN allocations more frequently in order to keep the TCP connection to its TURN server alive.

4.2. ALPN - Use of Application Layer Protocol Negotiation

The application layer protocol negotiation (ALPN) [draft-ietf-tls-applayerprotoneg] specifies a TLS extension which permits the application layer to negotiate protocol selection within the TLS handshake. This provides an explicit and visable indication of the application layer protocol associated with the TLS connection allowing the application protocol to be visable without relying on the port number to identify the protocol.

[draft-ietf-tls-applayerprotoneg] could therefore be used to identify that it is WebRTC media that is contained within the TLS connection.

ALPN is effectively an extension to the HTTP CONNECT mechanism decribed in Section 4.1 since the establishment of the TLS connection would require the use of this mechanism in the presence of a proxy as described in [draft-ietf-httpbis-http2].

4.3. TURN server connection via WebSocket

The WebRTC client could connect to a TURN server via WebSocket [RFC6455] as described in [draft-chenxin-behave-turn-WebSocket]. This might have benefits in very restrictive environments where HTTPS is not permitted through the proxy. However, such environments are also likely to deploy DPI boxes which would eventually complain against usage of WebSocket or block WebRTC traffic based on other heuristic means. It is also to be expected that an environment that does not allow HTTPS will also forbid usage of WebSocket over TLS.

In addition, usage of TURN over WebSocket puts an additional burden on existing TURN server implementation to support HTTP and WebSocket.

This is again effectively an extension to the HTTP CONNECT mechanism decribed in Section 4.1 since the establishment of the webcoskets connection would require the use of this mechanism in the presence of a proxy as described in [draft-ietf-httpbis-http2]. Like the ALPN approach the websockets approach also includes that the purpose of the websockets connection is to transport WebRTC media.

4.4. HTTP Fallback for RTP Media Streams

As an alternative to using a TURN server [draft-miniero-rtcweb-http-fallback] proposed to send RTP directly over HTTP. This approach bears some similarities with TURN as it also uses a RTP relay. However, it uses HTTP GET and POST requests to receive and send RTP packets.

Despite a number of open issues, the proposal addreses some corner cases. However, the expected benefit in form of an increased success rate for establishment of a media stream seems rather small.

4.5. Port Control Protocol

As a further alternative, the Port Control Protocol (PCP) [RFC6887] allows the client to communicate with the NAT/FW and negotiate how incoming IPv6 or IPv4 packets are translated and forwarded. However, to be successful such a solution would require the widespread deployment and use of PCP enabled firewalls so this does not appear to be a workable solution at least for early deployments of WebRTC.

4.6. Network Specific TURN Server

If a network specific TURN server under administrative control of the organization is deployed it is desirable to reach this TURN server via UDP. The TURN server could be specified in the proxy configuration script, giving the browser the possibility to learn how to access it. Then, when gathering candidates, this TURN server would always be used such that the WebRTC client application could get UDP traffic out to the internet.

Since the TURN server is under the same administrative control as the NAT/FW then it can be assumed that the NAT/FW allows WebRTC media that traverses the TURN server to traverse the NAT/FW.

The implementation of this solution in WebRTC is actually a requirement specified in [draft-ietf-rtcweb-use-cases-and-requirements].

The implementation of this solution in WebRTC does not remove the need for other solutions for the case when there is no such network specific TURN server.

5. Requirements for RTCWEB-enabled browsers

THIS SECTION IS EVEN MORE WORK IN PROGRESS THAN PREVIOUS SECTIONS.

For the purpose of relaying WebRTC media streams or data channels a browser needs to be able to

- o connect to a TURN server via UDP, TCP and TLS,
- o support a mechanism for connecting to a TURN server in the presence of a firewall that only permits connections that orginate from a HTTP Proxy. The mechanism is for further study.
- o connect to the TURN server via application specified ports other than the default STUN ports including the HTTP(s) ports,
- o use the same proxy selection procedure for TURN as currently done for HTTP (e.g. Web Proxy Autodiscovery Protocol (WPAD) and .pac-files for Proxy-Auto-Config),
- o use a preconfigured or standardized port range for UDP-based media streams or data channels,
- o learn from the proxy configuration script about the presence of a local TURN server and use it for sending UDP traffic to the internet,
- o as an option and if needed, support ICE-TCP for TCP-based direct media connection to the WebRTC peer.

6. Acknowledgements

The authors want to thank Heinrich Haager for all his input during many valuable discussions.

Furthermore, the authors want to thank for comments and suggestions received from Bernard Aboba, Xavier Marjou, Dan Wing, ...

7. IANA Considerations

This memo includes no request to IANA.

8. Security Considerations

In case of using HTTP CONNECT to a TURN server the security consideration of [[draft-ietf-httpbis-p2-semantics], Section-4.3.6] apply. It states that there "are significant risks in establishing a tunnel to arbitrary servers, particularly when the destination is a well-known or reserved TCP port that is not intended for Web traffic. ... Proxies that support CONNECT SHOULD restrict its use to a limited set of known ports or a configurable whitelist of safe request targets."

Consequently when HTTP CONNECT is used to reach a TURN server, the proxy administrator SHOULD configure a whitelist of trusted TURN servers and/or a blacklist of TURN server known to be subject to fraud or other undesired behavior.

With respect to the other discussed alternatives the security considerations of the corresponding RFCs and Internet Drafts apply.

9. References

9.1. Normative References

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

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC2817] Khare, R. and S. Lawrence, "Upgrading to TLS Within HTTP/1.1", RFC 2817, May 2000.

[RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, July 2003.

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

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

[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.

9.2. Informative References

[RFC6062] Perreault, S. and J. Rosenberg, "Traversal Using Relays around NAT (TURN) Extensions for TCP Allocations", RFC 6062, November 2010.

[RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, December 2011.

[RFC6544] Rosenberg, J., Keranen, A., Lowekamp, B., and A. Roach, "TCP Candidates with Interactive Connectivity Establishment (ICE)", RFC 6544, March 2012.

[RFC6887] Wing, D., Cheshire, S., Boucadair, M., Penno, R., and P. Selkirk, "Port Control Protocol (PCP)", RFC 6887, April 2013.

[draft-chenxin-behave-turn-WebSocket] Xin. Chen, "Traversal Using Relays around NAT (TURN) Extensions for WebSocket Allocations", 2013, <http://tools.ietf.org/html/draft-chenxin-behave-turn-WebSocket>.

[draft-ietf-httpbis-http2] M. Belshe, R. Peon, M. Thomson, A. Melnikov, "Hypertext Transfer Protocol version 2.0", 2013, <http://tools.ietf.org/html/draft-ietf-httpbis-http2-09#section-8.3>.

[draft-ietf-httpbis-p2-semantics] R. Fielding, J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", 2013, <http://tools.ietf.org/html/draft-ietf-httpbis-p2-semantics-25#section-4.3.6>.

[draft-ietf-rtcweb-data-channel]
R. Jesup, S. Loreto, M. Tuexen, "RTCWeb Data Channels", 2013, <http://tools.ietf.org/html/draft-ietf-rtcweb-data-channel>.

[draft-ietf-rtcweb-use-cases-and-requirements]
C. Holmberg, S. Hakansson, G. Eriksson, "Web Real-Time Communication Use-cases and Requirements", 2013, <http://tools.ietf.org/html/draft-ietf-WebRTC-use-cases-and-requirements>.

[draft-ietf-tls-applayerprotoneg]
S. Friedl, A. Popov, A. Langley, E. Stephan, "Transport Layer Security (TLS) Application Layer Protocol Negotiation Extension", 2013, <http://tools.ietf.org/html/draft-ietf-tls-applayerprotoneg>.

[draft-miniero-rtcweb-http-fallback]
L. Miniero, "HTTP Fallback for RTP Media Streams", 2012, <http://tools.ietf.org/html/draft-miniero-rtcweb-http-fallback>.

Authors’ Addresses

Thomas Stach
Unify
Dietrichgasse 27-29
Vienna 1030
AT

Email: thomas.stach@unify.com

Andrew Hutton
Unify
Technology Drive
Nottingham NG9 1LA
UK

Email: andrew.hutton@unify.com

Justin Uberti
Google
747 6th Ave S
Kirkland, WA 98033
US

Email: justin@uberti.name