Behavior Engineering for Hindrance Avoidance
Internet-Draft
Intended status: Standards Track
Expires: March 16, 2014

X. Chen
Huawei
S. Garcia Murillo
Medooze
O. Moskalenko
Yahoo
V. Pascual
Quobis
L. Miniero
Meetecho

September 12, 2013

WebSocket Protocol as a Transport for Traversal Using Relays around NAT (TURN)

draft-chenxin-behave-turn-websocket-01

Abstract

This document defines an extension to the Traversal Using Relays around NAT (TURN) protocol, in order to allow it to run over a WebSocket channel. This will allow clients in restrictive networks to traverse them and effectively exchange and relay media or data over WebSockets.

Status of This Memo

Copyright Notice

1. Introduction

Traversal Using Relays around NAT (TURN) [RFC5766], which assigns a transport address allocation for clients and relays data between the address and the clients, is an extension to the Session Traversal Utilities for NAT [RFC5389] protocol. TURN is used for NAT traversal in some complicated types of NAT network by UDP-based media sessions [RFC5766] or TCP-based media sessions [RFC6062]. It is also used in conjunction with the Interactive Connectivity Establishment (ICE) [RFC5245] technique.

In some particularly restrictive networks though, e.g., a web proxy or firewall that only allows HTTP traffic to pass through, TURN UDP-based media sessions and TCP-based media sessions do not work. These types of networks are often deployed in corporations, prisons, hotels, airports and other locations that may need to limit the access, and as such legitimate users trying to set up a real-time multimedia session in such a scenario would find themselves unable to do so. This is a known issue and in fact the RTCWEB specification, which provides the means to realize direct interactive rich communications between two peers by using just their web browsers, has an explicit requirement to allow such peers to use some kind of fallback communication in HTTP-only networks, as specified in [I-D.ietf-rtcweb-use-cases-and-requirements](F37).

That said, this document is aimed at targeting such scenarios, and as such defines an extension to the standard TURN protocol that allows it to run over a WebSocket [RFC6455] channel.

The WebSocket [RFC6455] protocol enables message exchange between clients and servers on top of a persistent TCP connection. Considering that the initial protocol handshake makes use of HTTP [RFC2616] semantics, thus allowing the WebSocket protocol to reuse existing HTTP infrastructure, this means that a client in a restrictive network would be able to exchange media over a WebSocket. Besides solving the HTTP fallback problem, this solution could also be easyly implemented and deployed within the existing RTCWEB framework.

For what concerns the impact of such an extensions on the interaction with legacy peers making use of the services provided by a TURN server, the connection between the server and such peers would still be based on UDP as [RFC5766] or TCP as [RFC6062] in a seamless and transparent fashion.

TURN client to TURN server	TURN server to peer
WebSocket over TCP/TLS	UDP TCP

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].

3. Deployment Topology

Within the context of real-time multimedia communications and considering a scenario that involves two peers, an HTTP fallback mechanism may fall in basically three different network topologies:

3.1. A topology whereas only one of the involved peers needs HTTP fallback for communication

```
                    +-------------+
                    |             |
      +-------------+ TURN Server +----------------+
      |    WS/WSS   |             |    UDP/TCP     |
      |             +-------------+                |
      |                                            |
      |                                            |
  +---+---+                                    +---+---+
  | Alice |                                    |  Bob  |
  +-------+                                    +-------+
```

Figure 1

In Figure 1, only one involved peer (Alice) is in a restrained network, which means Alice needs to make use of a WebSocket connection to traverse the firewall and/or proxy. The situation for Bob is better, he could connect to the TURN server by UDP or TCP using the existing mechanisms.

When Alice wants to communicate with Bob, she needs to request a UDP or TCP allocation in the WebSocket server for Bob, which is then transferred to the WebSocket channel. The WebSocket server will receive the request and handle it like a TURN server. The processing of TURN messages is exactly the same as TURN UDP and TURN TCP, and the WebSocket server will also allocate a UDP or TCP relay address for Bob. The application data between Alice and Bob will be packaged and relayed to each other by the WebSocket server.

3.2. A topology whereas both the involved peers need HTTP fallback for communication, using two different intermediaries

```
                +-------------+         +-------------+
                |             |         |             |
       +--------+ TURN Server +---------+ TURN Server +---------+
       | WS/WSS |             | UDP/TCP |             | WS/WSS  |
       |        +-------------+         +-------------+         |
       |                                                        |
       |                                                        |
   +---+---+                                                +---+---+
   | Alice |                                                |  Bob  |
   +-------+                                                +-------+
```

Figure 2

In Figure 2, both Alice and Bob are in restrictive networks, so both need a fallback mechanism. In this slightly more complex scenario, both Alice and Bob each have been configred to refer to different WebSocket servers. In this scenario, Alice and Bob need to request the TURN allocation in their own WebSocket server using a WebSocket connection.

Again, just as before the processing of TURN messages is exactly the same as TURN UDP and TURN TCP. The only difference with previous sceneario is that, in this case, the involved WebSocket server have to relay the application data to each other by either UDP, TCP or other existing ways, using the existing TURN mechanics for the purpose.

It is of course suggested that Alice and Bob allocate the same type of transport address, so that their reference WebSocket server could connect to each other by this address directly.

The scenario would of course be simpler in case the TURN servers depicted in the figure above happen to be the same TURN server, i.e., if Alice and Bob both referred to the same server. In that case, it may be possible to relay the data internally instead of using an UDP/TCP connection.

```
                        +-------------+
                        |             |
            +-----------+ TURN Server +-----------+
            |  WS/WSS   |             |  WS/WSS   |
            |           +-------------+           |
            |                                     |
            |                                     |
        +---+---+                             +---+---+
        | Alice |                             |  Bob  |
        +-------+                             +-------+
```

Figure 3

However, this is an implementation decision, not affecting the TURN clients interaction with the TURN server and it will not be covered in detail within this specification.

4. The WebSocket TURN Sub-Protocol

The term WebSocket sub-protocol refers to an application-level protocol layered on top of a WebSocket connection. This document specifies the WebSocket TURN sub-protocol for carrying TURN requests and responses through a WebSocket connection.

4.1. Handshake

The TURN Client and TURN Server negotiate usage of the WebSocket TURN sub-protocol during the WebSocket handshake procedure as defined in section 1.3 of [RFC6455]. The Client MUST include the value "turn" in the Sec-WebSocket-Protocol header in its handshake request. The 101 reply from the Server MUST contain "turn" in its corresponding Sec-WebSocket-Protocol header.

Also, the TURN WebSocket Client shall set the Origin header if the TURN connection is createad in a Web context as defined in [RFC6454]. Particularly, for WebRTC, the Origin header shall be set to the value of the URI of the HTML page creating the PeerConnection.

Below is an example of a WebSocket handshake in which the Client requests the WebSocket TURN sub-protocol support from the Server:

```
GET / HTTP/1.1
Host: TURN-ws.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: http://www.example.com
Sec-WebSocket-Protocol: turn
```

```
     Sec-WebSocket-Version: 13
```

The handshake response from the Server accepting the WebSocket TURN sub-protocol would look as follows:

```
     HTTP/1.1 101 Switching Protocols
     Upgrade: websocket
     Connection: Upgrade
     Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
     Sec-WebSocket-Protocol: turn
```

Once the negotiation has been completed, the WebSocket connection is established and can be used for the transport of TURN requests and responses. The WebSocket messages transmitted over this connection MUST conform to the negotiated WebSocket sub-protocol.

4.2. TURN message framing

TURN messages shall be transported in unfragmented binary frames (FIN:1,opcode:%x2).

The WebSocket frame data shall be a valid TURN packet, so the length of the payload of the WebSocket frame shall be lower than the maximum size allowed (2^16 bytes) for a TURN request or response as defined in [RFC5766].

TURN client using TURN over WebSockets should follow the recommendations in section 2.7 of [RFC5766] "Avoiding IP Fragmentation" when sending application data on the client-to-server-leg as messages could be relied over a UDP connection to the peer client.

4.3. TURN Allocation

This document extends both [RFC5766] (TURN UDP relay) and [RFC6062] (TURN TCP relay) with a new type of client-to-server connection, i.e. WebSocket. For TURN allocations, WebSocket is a type of TCP client-to-server connection and is subject to all TURN TCP considerations.

This specification strictly follows the allocation definition in section 5 in [RFC5766]. In the 5-tuple, the transport address is always TCP, of course, when WebSockets are used. All definitions in the section 5 of [RFC5766] are applicable to the WebSockets TURN connections.

4.4. TURN Operation

The operation of the client, server and peer is the same as TURN UDP and TURN TCP, with the difference consisting in the new connection channel - WebSocket.

4.5. TURN and TURNS URI WebSocket Transport Parameter

This document defines the value "ws" as a transport parameter value for a TURN and TURNS URI [I-D.petithuguenin-behave-turn-uris] to be contacted using the TURN WebSocket sub-protocol as transport.

The "turns" URI scheme MUST be used when TURN is run over Secure Websockets (WebSockets over TLS) and the "turn" scheme MUST be used otherwise.

The updated augmented BNF (Backus-Naur Form) for this parameter is the following (the original BNF for this parameter can be found in [I-D.petithuguenin-behave-turn-uris]):

```
  transport     = "udp" / "tcp" / "ws"  / transport-ext
```

4.6. Impact on ICE candidates and SDP signalling

This specification does not have any impact on ICE. In fact, all the related candidates would be allocated at the TURN sever, and as such no modifications are needed in the SDP signaling in order to support the TURN over WebSockets operation.

5. IANA Considerations

RFC Editor Note: Please set the RFC number assigned for this document in the sub-sections below and remove this note.

5.1. Registration of the WebSocket TURN Sub-Protocol

This specification requests IANA to register the WebSocket TURN sub-protocol under the "WebSocket Subprotocol Name" Registry with the following data:

Subprotocol Identifier: turn

Subprotocol Common Name: WebSocket Transport for TURN

Subprotocol Definition: TBD: this document

6. Security Considerations

TBD.

7. Change Summary

Note to RFC Editor: Please remove this whole section.

The following are the major changes between the 00 and the 01 versions of the draft:

- o Removal of multiplexing and references to BCFP and other non related protocols
- o Websocket TURN sub protocol specification
- o TURN message framing inside Websocket
- o Extension to turn and turns URI
- o Impact analisys on ice candidates SDP negotiation

8. Acknowledgements

Paul Kyzivat helped with the formatting of this draft.

9. References

9.1. Normative References

[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.

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

9.2. Informative References

[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.

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

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

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

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[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.

[I-D.petithuguenin-behave-turn-uris]
Petit-Huguenin, M., Nandakumar, S., Salgueiro, G., and P. Jones, "Traversal Using Relays around NAT (TURN) Uniform Resource Identifiers", draft-petithuguenin-behave-turn-uris-06 (work in progress), August 2013.

[I-D.ietf-rtcweb-use-cases-and-requirements]
Holmberg, C., Hakansson, S., and G. Eriksson, "Web Real-Time Communication Use-cases and Requirements", draft-ietf-rtcweb-use-cases-and-requirements-11 (work in progress), June 2013.

Authors’ Addresses

Xin Chen
Huawei

Email: hangzhou.chenxin@huawei.com

Sergio Garcia Murillo
Medooze

Email: sergio.garcia.murillo@gmail.com

Oleg Moskalenko
Yahoo

Email: olegm@yahoo-inc.com

Victor Pascual
Quobis

Email: victor.pascual@quobis.com

Lorenzo Miniero
Meetecho

Email: lorenzo@meetecho.com