Alternative Proposal for Traversal Using Relays around NAT (TURN) Extensions for TCP Allocations
draft-petithuguenin-turn-tcp-variant-01

Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on September 10, 2009.

Copyright Notice

Copyright (c) 2009 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 in effect on the date of publication of this document (http://trustee.ietf.org/license-info).
Please review these documents carefully, as they describe your rights and restrictions with respect to this document.

Abstract

This document proposes to use a shared TCP connection between a Traversal Using Relays around NAT (TURN) client and a TURN server.
instead of the multiple TCP connections proposed by
[I-D.ietf-behave-turn-tcp]

Table of Contents

1.  Introduction ............................................ 3
2.  Shared Connection vs Multiple Connections Comparison ...... 3
   2.1.  Unified Mechanism ................................. 3
   2.2.  TCP Connection Overhead ............................ 3
   2.3.  Multiple Connections Advantages ..................... 4
3.  Terminology ............................................. 4
4.  Client and Server Processing ............................. 5
   4.1.  Sending a Connect Request .......................... 5
   4.2.  Receiving a Connect Request ........................ 5
   4.3.  Receiving a Connect Response ....................... 5
   4.4.  Receiving a TCP Connection on an Allocation .......... 5
   4.5.  Receiving a ConnectAttempt Request ................ 5
   4.6.  Receiving a ConnectAttempt Response ................ 5
   4.7.  Sending Data ....................................... 6
   4.8.  Sending an AdjustWindow Indication .................. 6
   4.9.  Receiving an AdjustWindow Indication ............... 6
5.  Security Considerations ................................ 6
6.  IANA Considerations .................................... 6
7.  Running Code Considerations ............................. 6
8.  Acknowledgements ..................................... 6
9.  References ............................................. 7
   9.1.  Normative References .............................. 7
   9.2.  Informative References ............................. 7
Appendix A.  Release notes .................................. 8
   A.1.  Modifications between -01 and -00 .................... 8
Author’s Address .......................................... 8
1. Introduction

[I-D.ietf-behave-turn-tcp] proposes to create a separate TCP connection between the TURN client and the TURN server for each TCP connection between the TURN server and a peer. This document proposes to reuse the multiplexing mechanism defined in [I-D.ietaf-behave-turn].

With this proposal, the data received and sent between the TURN server and the peer are multiplexed on the TCP connection between the TURN client and the TURN server by using either the Send/Data indications or by using channels. A window mechanism similar to the one described in SSH [RFC4254] is used to manage the flow of data over the shared TCP connection.

2. Shared Connection vs Multiple Connections Comparison

2.1. Unified Mechanism

The main question behind this proposal is why not reusing the existing multiplexing design in [I-D.ietaf-behave-turn], but one can ask the opposite question: Why not apply the same multiple connections mechanism proposed in [I-D.ietaf-behave-turn-tcp] to [I-D.ietaf-behave-turn]?

This would greatly simplify the TURN specification because the TURN client IP address and port of a data connection would uniquely identify the peer so channels, Send and Data indications would become unnecessary. Data connections simply forward data in both direction after the end of the ConnectionBind transaction so when is used UDP both between the TURN client and the TURN server and between the TURN server and the peer the packets can be sent and received without overhead. When TCP is used between the TURN client and the TURN server and UDP between the TURN server and the peer, the [RFC4571] framing can be used.

In any case, having only one mechanism for carrying data between the TURN client and TURN server is better than having two mechanisms. Note that it is unlikely that TURN will be modified this late to support the TURN TCP mechanism.

2.2. TCP Connection Overhead

NATs create per-stream state and so can cause other streams to fail once they run out of space [I-D.iab-ip-model-evolution], thus preventing additional peer connections from the same allocation. A shared TCP connection does not create additional per-stream state in
TCP connection establishment is relatively slow. This is the reason why HTTP 1.1 [RFC2616] has a persistent connections feature and SSH [RFC4254] has a multiplexing mechanism.

The impact of TCP connection establishment can be significant when TURN TCP is used with ICE TCP [I-D.ietf-mmusic-ice-tcp]. ICE TCP will open a number of TCP connections for connectivity check and then close all of them excepted one. This behavior fits well with the multiplexing mechanism, where no additional TCP connections will be created for the connectivity checks.

Multiple TCP connections between the same endpoints do not share congestion state [1]. (Is it still true?) Using a multiplexed TCP connection can eliminate the slow start delay for subsequent connections and improve congestion control.

2.3. Multiple Connections Advantages

A shared TCP connection can suffer from Head-Of-Line blocking, preventing a stream to forward data because a segment carrying data for another stream was lost. This cannot happen with multiple TCP connections. Note that the same problem exists in TURN when TCP is used between the TURN client and the TURN server and UDP between the TURN server and the peers.

The multiple TCP connections mechanism permits some optimizations, either in userspace, kernel or hardware, that are difficult to use with the shared connection mechanism. Shared connections can also prevent using ECN or new congestion algorithms and make the implementation of an eventual "preserving behavior" difficult.

The shared connection mechanism reuses the multiplexing mechanism from TURN, so there is no additional complexity added by this in an implementation that already supports TURN. The only complexity added is the management of the window. The mechanism is directly inspired by the SSH mechanism and so can reuse the experience [2] acquired from the OpenSSH implementation.

3. 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].
4. Client and Server Processing

4.1. Sending a Connect Request

To initiate a TCP connection to a peer, a TURN client MUST send a Connect request to the TURN server that include a WINDOW-SIZE attribute containing how many bytes of data can be sent to the TURN client without adjusting the window, and a MAX-SIZE attribute containing the maximum size of the buffer allocated.

4.2. Receiving a Connect Request

If the connection is successful, the TURN server sends back to the TURN client a Connect response containing a WINDOW-SIZE attribute containing how many bytes of data can be sent to the TURN server without adjusting the window, and a MAX-SIZE attribute containing the maximum size of the buffer allocated. The TURN server associates the current window size in the WINDOW-SIZE attribute to the TCP connection to the peer.

4.3. Receiving a Connect Response

The TURN client associates the current window size in the WINDOW-SIZE attribute to the IP address and port of the peer TCP connection.

4.4. Receiving a TCP Connection on an Allocation

After accepting the connection, the TURN server sends a ConnectionAttempt request to the client that include a WINDOW-SIZE attribute containing how many bytes of data can be sent to the TURN server without adjusting the window, and a MAX-SIZE attribute containing the maximum size of the buffer allocated. The TURN server associates the current window size in the WINDOW-SIZE attribute to the TCP connection to the peer.

4.5. Receiving a ConnectAttempt Request

The TURN client associates the current window size in the WINDOW-SIZE attribute to the IP address and port of the peer TCP connection and sends back to the TURN server a ConnectAttempt response containing a WINDOW-SIZE attribute containing how many bytes of data can be sent to the TURN client without adjusting the window, and a MAX-SIZE attribute containing the maximum size of the buffer allocated.

4.6. Receiving a ConnectAttempt Response

The TURN server associates the current window size in the WINDOW-SIZE attribute to the IP address and port of the peer TCP connection.
4.7. Sending Data

When sending data in a ChannelData, Send or Data message the TURN server or client decreases the current window size by the number of bytes sent. The TURN server or client MUST stop sending when the current window size is smaller than the size of the data to send.

4.8. Sending an AdjustWindow Indication

When ready to receive more data, the TURN server or client sends an AdjustWindow indication to the other side. The AdjustWindow indication MUST contain either a XOR-PEER-ADDRESS or a CHANNEL-NUMBER attribute identifying the TCP connection to the peer. The AdjustWindow indication MUST contain a ADD-SIZE attribute containing the value to add to the current window size.

4.9. Receiving an AdjustWindow Indication

When receiving an AdjustWindow indication, a TURN client or server uses the XOR-PEER-ADDRESS or CHANNEL-NUMBER to find the current window size associated to the TCP connection to the peer. The TURN client or server then increases the window size by the value in the ADD-SIZE attribute and can eventually restart sending data.

5. Security Considerations

TBD

6. IANA Considerations

TBD

7. Running Code Considerations

TBD

8. Acknowledgements

Adam Roach proposed to use the SSH algorithm at the microphone in the BEHAVE session in Minneapolis.

Thanks to Remi Denis-Courmont and Simon Perreault for their comments, suggestions and questions that helped to improve this document.
9. References

9.1. Normative References


9.2. Informative References


[I-D.ietf-mmusic-ice-tcp] Rosenberg, J., "TCP Candidates with Interactive Connectivity Establishment (ICE)",

URIs


Appendix A. Release notes

This section must be removed before publication as an RFC.

A.1. Modifications between -01 and -00

- Changed author address.
- Changed the IPR to trust200902.
- Rewrote abstract.
- Rewrote introduction with comparisons between the two mechanisms.
- MAX-SIZE is the size of the allocated buffer.
- Added support for ConnectAttempt.

Author’s Address

Marc Petit-Huguenin
(Unaffiliated)

Email: petithug@acm.org