XMPP Working Group
Internet-Draft
Intended status: Informational
Expires: January 6, 2010

K. Hartke
C. Bormann
Universitaet Bremen TZI

July 5, 2009

STUN/TURN using PHP in Despair
draft-hartke-xmpp-stupid-00

Status of this Memo

Copyright Notice

Abstract

NAT (Network Address Translator) Traversal may require TURN (Traversal Using Relays around NAT) functionality in certain cases that are not unlikely to occur. There is little incentive to deploy TURN servers, except by those who need them -- who may not be in a position to deploy a new protocol on an Internet-connected node, in particular not one with deployment requirements as high as those of TURN.

"STUN/TURN using PHP in Despair" is a highly deployable protocol for obtaining TURN-like functionality, while also providing the most important function of STUN.

Table of Contents

1. Introduction

NAT (Network Address Translator) Traversal may require TURN (Traversal Using Relays around NAT) [I-D.ietf-behave-turn] functionality in certain cases that are not unlikely to occur. There is little incentive to deploy TURN servers, except by those who need them -- who may not be in a position to deploy a new protocol on an Internet-connected node, in particular not one with deployment requirements as high as those of TURN.

"STUN/TURN using PHP in Despair" is a highly deployable protocol for obtaining TURN-like functionality, while also providing the most important function of STUN [RFC5389].

The high degree of deployability is achieved by making STuPiD a Web service, implementable in any Web application deployment scheme. As PHP appears to be the solution of choice for avoiding deployment problems in the Web world, a PHP-based sample implementation of STuPiD is presented in Figure 5 in Appendix B. (This single-page script has been tested with a free-of-charge web hoster, so it should be deployable by literally everyone.)

1.1. The Need for Standardization

If STuPiD is so easy to deploy, why standardize on it? First of all, STuPiD server implementations will be done by other people than the clients making use of the service. Clearly communicating between these communities is a good idea, in particular if there are security considerations.

Having one standard form of STuPiD service instead of one specific to each kind of client also creates an incentive for optimized implementations.

Finally, where STuPiD becomes part of a client standard (such as a potential extension to XMPP's in-band byte-stream protocol as hinted in Appendix C), it is a good thing if STuPiD is already defined.

Hence, this document focuses on the definition of the STuPiD service itself, tries to make this as general as possible without increasing complexity or cost and leaves the details of any client standards to future documents.

2. Basic Protocol Operation

The STuPiD protocol will typically be used with application instances that first attempt to obtain connectivity using mechanisms similar to those described in the STUN specification [RFC5389]. However, with STuPiD, STUN is not really needed for TCP, as was demonstrated in previous STUN-like implementations [STUNT]. Instead, STuPiD (like [STUNT]) provides a simple Web service that echoes the remote address and port of an incoming HTTP request; in most cases, this is enough to get the job done.

In case no connection can be established with this simple STUN(T)-like mechanism, a TURN-like relay is needed as a final fall-back. The STuPiD protocol supports this, but solely provides a way for the data to be relayed. STuPiD relies on an out-of-band channel to notify the peer whenever new data is available (synchronization signal). See Appendix C for one likely example of such an out-of-band channel. (Note that the out-of-band channel may have a much lower throughput than the STuPiD relay channel -- this is exactly the case in the example provided in Appendix C, where the out-of-band channel is typically throughput-limited to on the order of a few kilobits per second.)

By designing the STuPiD web service in such a way that it can be implemented by a simple PHP script such as that presented in Appendix B, it is easy to deploy by those who need the STuPiD services. The combination of the low-throughput out-of-band channel for synchronization and the STuPiD web service for bulk data relaying is somewhat silly but gets the job done.

The STuPiD data relay is implemented as follows (see Figure 1):

1. Peer A, the source of the data to be relayed, stores a chunk of data at the STuPiD server using an opaque identifier, the "chunk identifier". How that chunk identifier is chosen is local to Peer A; it could be composed of a random session id and a sequence number.

2. Peer A notifies the receiver of the data, Peer B, that a new data chunk is available, specifying the URI needed for retrieval. This notification is provided through an out-out-band channel. (As an optimization for multiple consecutive transfers, A might inform B once of a constant prefix of that URI and only send a varying part such as a sequence number in each notification -- this is something to be decided in the client-client notification protocol.)

3. Peer B retrieves the data from the STuPiD server using the URI provided by Peer A.

Note that the data transfer mechanism is one-way, i.e. to send data in the other direction as well, Peer B needs to perform the same steps using a STuPiD server at the same or a different location.

```
                   STuPiD   ```````````````````````````````,
                   Script   <----------------------------. ,
                                                         | ,
                     ^ ,                                 | ,
                     | ,                                 | ,
               (1)   | ,                                 | ,  (3)
               POST  | ,                                 | ,  GET
                     | ,                                 | ,
                     | v                                 | v

                   Peer A   ------------------------>   Peer B
                                      (2)
                                  out-of-band
                                  Notification
```

Figure 1: STuPiD Protocol Operation

3. Protocol Definition

3.1. Terminology

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in BCP 14, RFC 2119 [RFC2119] and indicate requirement levels for compliant STuPiD implementations.

3.2. Discovering External IP Address and Port

A client may discover its external IP address and the port required for port prediction by performing a HTTP GET request to a STuPiD server. The STuPiD server MUST reply with the remote address and remote port in the following format:

```
host ":" port
```

where 'host' and 'port' are defined as in [RFC3986].

3.3. Storing Data

Data chunks are stored using the POST request of HTTP. The STuPiD server MUST support one URI parameter which is passed as query-string:

'chid': A unique ID identifying the data chunk to be stored. The ID SHOULD be chosen from the characters of the base64url set [RFC4648].

The payload of the POST request MUST be the data to be stored. The 'Content-Type' SHOULD be 'application/octet-stream', although a STuPiD server implementation SHOULD simply ignore the 'Content-Type' as a client implementation may be restricted and may not able to specify a specific 'Content-Type'. (E.g., in certain cases, the peer may be limited to sending the data as multipart-form-encoded -- still, the data is stored as a byte stream.)

STuPiD servers may reject data chunks that are larger than some predefined limit. This maximum size in bytes of each data chunk is RECOMMENDED to be 65536 or more.

As HTTP already provides data transparency, the data chunk SHOULD NOT be encoded using Base64 or any other data transparency mechanism; in any case, the STuPiD server will not attempt to decode the chunk.

The sender MUST wait for the HTTP response before going on to notify the receiver.

3.4. Notification

The sender notifies the receiver of the data chunk by passing via an out-of-band channel (which is not part of the STuPiD protocol):

The full URL from which the data chunk can be retrieved, i.e. the same URL that was used to store the data chunk, including the chunk ID parameter.

The exact notification mechanism over the out-of-band channel and the definition of a session is dependent on the out-of-band channel. See Appendix C for one example of such an out-of-band channel.

3.5. Retrieving Data

The notified peer retrieves the data chunk using a GET request with the URL supplied by the sender. The STuPiD server MUST set the 'Content-Type' of the returned body to 'application/octet-stream'.

4. Implementation Notes

A STuPiD server implementation SHOULD delete stored data some time after it was stored. It is RECOMMENDED not to delete the data before five minutes have elapsed after it was stored. Different client protocols will have different reactions to data that have been deleted prematurely and cannot be retrieved by the notified peer; this may be as trivial as packet loss or it may cause a reliable byte-stream to fail (Appendix B). (TODO: It may be useful to provide some hints in the storing POST request.)

STuPiD clients should aggregate data in order to minimize the number of requests to the STuPiD server per second. The specific aggregation method chosen depends on the data rate required (and the maximum chunk size), the latency requirements, and the application semantics.

Clearly, it is up to the implementation to decide how the data chunks are actually stored. A sufficiently silly STuPiD server implementation might for instance use a MySQL database.

5. Security Considerations

The security objectives of STuPiD are to be as secure as if NAT traversal had succeeded, i.e., an on-path attacker can overhear and fake messages, but an off-path attacker cannot. If a higher level of security is desired, it should be provided on top of the data relayed by STuPiD, e.g. by using XTLS [I-D.meyer-xmpp-e2e-encryption].

Much of the security of STuPiD is based on the assumption that an off-path attacker cannot guess the chunk identifiers. A suitable source of randomness [RFC4086] should be used to generate at least a sufficiently large part of the chunk identifiers (e.g., the chunk identifier could be a hard to guess prefix followed by a serial number).

To protect the STuPiD server against denial of service and possibly some forms of theft of service, it is RECOMMENDED that the POST side of the STuPiD server be protected by some form of authentication such as HTTP authentication. There is little need to protect the GET side.

6. References

6.1. Normative References

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

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC4086] Eastlake, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, June 2005.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.

6.2. Informative References

[I-D.ietf-behave-turn] Rosenberg, J., Mahy, R., and P. Matthews, "Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)", draft-ietf-behave-turn-16 (work in progress), July 2009.

[I-D.ietf-xmpp-3920bis] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Core", draft-ietf-xmpp-3920bis-00 (work in progress), June 2009.

[I-D.meyer-xmpp-e2e-encryption] Meyer, D. and P. Saint-Andre, "XTLS: End-to-End Encryption for the Extensible Messaging and Presence Protocol (XMPP) Using Transport Layer Security (TLS)", draft-meyer-xmpp-e2e-encryption-02 (work in progress), June 2009.

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

[STUNT] Hanson, R., "STUNT & out-of-band channels", September 2007, <http://deusty.blogspot.com/2007/09/stunt-out-of-band-channels.html>.

Appendix A. Examples

This appendix provides some examples of the STuPiD protocol operation.

```
Request:

   GET /stupid.php HTTP/1.0
   User-Agent: Example/1.11.4
   Accept: */*
   Host: example.org
   Connection: Keep-Alive

Response:

   HTTP/1.1 200 OK
   Date: Sun, 05 Jul 2009 00:30:37 GMT
   Server: Apache/2.2
   Cache-Control: no-cache, must-revalidate
   Expires: Sat, 26 Jul 1997 05:00:00 GMT
   Vary: Accept-Encoding
   Content-Length: 17
   Keep-Alive: timeout=1, max=400
   Connection: Keep-Alive
   Content-Type: application/octet-stream

   192.0.2.239:36654
```

Figure 2: Discovering External IP Address and Port

Request:

```
POST /stupid.php?chid=i781hf64-0 HTTP/1.0
User-Agent: Example/1.11.4
Accept: */*
Host: example.org
Connection: Keep-Alive
Content-Type: application/octet-stream
Content-Length: 11

Hello World
```

Response:

```
HTTP/1.1 200 OK
Date: Sun, 05 Jul 2009 00:20:34 GMT
Server: Apache/2.2
Cache-Control: no-cache, must-revalidate
Expires: Sat, 26 Jul 1997 05:00:00 GMT
Vary: Accept-Encoding
Content-Length: 0
Keep-Alive: timeout=1, max=400
Connection: Keep-Alive
Content-Type: application/octet-stream
```

Figure 3: Storing Data

Request:

```
GET /stupid.php?chid=i781hf64-0 HTTP/1.0
User-Agent: Example/1.11.4
Accept: */*
Host: example.org
Connection: Keep-Alive
```

Response:

```
HTTP/1.1 200 OK
Date: Sun, 05 Jul 2009 00:21:29 GMT
Server: Apache/2.2
Cache-Control: no-cache, must-revalidate
Expires: Sat, 26 Jul 1997 05:00:00 GMT
Vary: Accept-Encoding
Content-Length: 11
Keep-Alive: timeout=1, max=400
Connection: Keep-Alive
Content-Type: application/octet-stream

Hello World
```

Figure 4: Retrieving Data

Appendix B. Sample Implementation

```
<?php
header("Cache-Control: no-cache, must-revalidate");
header("Expires: Sat, 26 Jul 1997 05:00:00 GMT");
header("Content-Type: application/octet-stream");

mysql_connect(localhost, "username", "password");
mysql_select_db("stupid");

$chid = mysql_real_escape_string($_GET["chid"]);

if ($_SERVER["REQUEST_METHOD"] == "GET") {
   if (empty($chid)) {
      echo $_SERVER["REMOTE_ADDR"] . ":" . $_SERVER["REMOTE_PORT"];
   } elseif ($result = mysql_query("SELECT `data` FROM `Data` " .
                         "WHERE `chid` = '$chid'")) {
      if ($row = mysql_fetch_array($result, MYSQL_ASSOC)) {
         echo base64_decode($row["data"]);
      } else {
         header("HTTP/1.0 404 Not Found");
      }
      mysql_free_result($result);
   } else {
      header("HTTP/1.0 404 Not Found");
   }
} elseif ($_SERVER["REQUEST_METHOD"] == "POST") {
   if (empty($chid)) {
      header("HTTP/1.0 404 Not Found");
   } else {
      mysql_query("DELETE FROM `Data` " .
                  "WHERE `timestamp` < DATE_SUB(NOW(), INTERVAL 5 MINUTE)");
      $data = base64_encode(file_get_contents("php://input"));
      if (!mysql_query("INSERT INTO `Data` (`chid`, `data`) " .
                       "VALUES ('$chid', '$data')")) {
         header("HTTP/1.0 403 Bad Request");
      }
   }
} else {
   header("HTTP/1.0 405 Method Not Allowed");
   header("Allow: GET, HEAD, POST");
}
mysql_close();
?>
```

Figure 5: STuPiD Sample Implementation

Appendix C. Using XMPP as Out-Of-Band Channel

XMPP [I-D.ietf-xmpp-3920bis] is a good choice for an out-of-band channel.

The notification protocol is closely modeled after XMPP's In-Band Bytestreams (IBB, see http://xmpp.org/extensions/xep-0047.html). Just replace the namespace and insert the STuPiD Retrieval URI instead of the actual Base64 encoded data, see Figure 8. (Note that the current proposal redundantly sends a sid and a seq as well as the chid composed of these two; it may be possible to optimize this, possibly sending the constant prefix of the URI once at bytestream creation time.)

Notifications MUST be processed in the order they are received. If an out-of-sequence notification is received for a particular session (determined by checking the 'seq' attribute), then this indicates that a notification has been lost. The recipient MUST NOT process such an out-of-sequence notification, nor any that follow it within the same session; instead, the recipient MUST consider the session invalid. (Adapted from http://xmpp.org/extensions/xep-0047.html#send)

Of course, other methods can be used for setup and teardown, such as Jingle (see http://xmpp.org/extensions/xep-0261.html).

```
<iq from='romeo@montague.net/orchard'
    id='jn3h8g65'
    to='juliet@capulet.com/balcony'
    type='set'>
  <open xmlns='urn:xmpp:tmp:stupid'
        block-size='65536'
        sid='i781hf64'
        stanza='iq'/>
</iq>
```

Figure 6: Creating a Bytestream: Initiator requests session

```
<iq from='juliet@capulet.com/balcony'
    id='jn3h8g65'
    to='romeo@montague.net/orchard'
    type='result'/>
```

Figure 7: Creating a Bytestream: Responder accepts session

```
<iq from='romeo@montague.net/orchard'
    id='kr91n475'
    to='juliet@capulet.com/balcony'
    type='set'>
  <data xmlns='urn:xmpp:tmp:stupid'
        seq='0'
        sid='i781hf64'
        url='http://example.org/stupid.php?chid=i781hf64-0'/>
</iq>
```

Figure 8: Sending Notifications: Notification in an IQ stanza

```
<iq from='juliet@capulet.com/balcony'
    id='kr91n475'
    to='romeo@montague.net/orchard'
    type='result'/>
```

Figure 9: Sending Notifications: Acknowledging notification using IQ

```
<iq from='romeo@montague.net/orchard'
    id='us71g45j'
    to='juliet@capulet.com/balcony'
    type='set'>
  <close xmlns='urn:xmpp:tmp:stupid'
         sid='i781hf64'/>
</iq>
```

Figure 10: Closing the Bytestream: Request

```
<iq from='juliet@capulet.com/balcony'
    id='us71g45j'
    to='romeo@montague.net/orchard'
    type='result'/>
```

Figure 11: Closing the Bytestream: Success response

Authors' Addresses

Klaus Hartke
Universitaet Bremen TZI

Email: hartke@tzi.org

Carsten Bormann
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63921
Fax: +49-421-218-7000
Email: cabo@tzi.org