NTCP: NAT traversal TCP HOWTO

Luca Gaballo

July 11, 2005

Contents

1 Product name and version number

Name: NTCP (NAT Traversal TCP)
version: 0.1

2 Company name

Company: France Telecom R&D
Department: MAPS/MMC
Project: SOLIPSIS
http://solipsis.netofpeers.net/wiki2/index.php/Main_Page

3 New and special in this release

It's the first release...everything is new!!! :)

4 Download source code and documentation

Anonymous SVN Access

The project's BerliOS Developer SVN repository can be checked out through anonymous (svnserver) SVN with the following instruction set. To see the list of the subdirectories available in the repository use the web-based SVN repository access with ViewCVS or WebSVN.

```
svn checkout svn://svn.berlios.de/ntcp/trunk
```

Developer SVN Access via SSH

Only project developers can access the SVN tree via this method. SSH2 must be installed on your client machine. Substitute developername with the proper value. Enter your site password when prompted.

```
svn checkout \
svn+ssh://developername@svn.berlios.de/svnroot/repos/ntcp/trunk
```

The subdirectories for documentation or source code only are respectively:

- /ntcp/trunk/doc
- /ntcp/trunk/p2pNetwork

5 Hardware and software requirements

General Requirements

- **Python**
  http://www.python.org/
- **Twisted** (at least 1.3)
  http://twistedmatrix.com/projects/core/

Special Unix Requirements

- **libpcap** (at least 0.9.1)
- **Pcapy**: python library for spoofing (modification of address, SYN, ACK, TTL, ...) (http://oss.coresecurity.com/projects/pcapy.html)
- **Impacket**: python library to sniff and listen for outgoing packets
  http://oss.coresecurity.com/projects

Special Windows Requirements

- **Python** (at least 2.3): some socket options are not available in the previous version
- **winpcap** (at least 3.0): the packet capture library for Windows
  http://www.winpcap.org

- **Netwox**: library in C language for spoofing
  http://www.laurentconstantin.com/en/netw/
- **Impacket**: python library to sniff and listen for outgoing packets
  http://oss.coresecurity.com/projects

6 Installation instructions, getting started tips, and documentation

6.1 Installation

There is no installation procedure at now. So download the source code (see download section), put it to the desired folder, and make the PYTHONPATH environment variable to point it. Execute this command or put it in your '*profile*' file.

```
export PYTHONPATH=\$PYTHONPATH:/directory_to_ntcp/ntcp/trunk
```

6.2 Modules' structure

The source code in the /ntct/trunk/p2pNetwork is organized like exposend

```
/ntcp/trunk/p2pNetwork

 |
 |__ discover: to discover NAT presence and configuration (use STUN)
 |
 |__ htcp: Connection Broker and Hole Punching code
 |
 |__ stun: STUN code (client and server)
 |
 |__ testTCP: tests for TCP through NAT connectivity
 |
 |__ solipsisSimulator.py: an application example
```

6.3 Getting started

To know how to use *ntcp* library in your code look at the examples. There are two sections, one for UDP and the other for TCP connectivity.

The UDP connectivity module is complete and allow to make communication between two endhost behind two different NATs. It needs of a STUN server and a Connection Broker in the public network.

The TCP connectivity module is still in the test phase. You test a TCP communication with or without spoofing. This module will remplace the UDP module integrating him.

6.3.1 UDP connectivity

Here the most important code sections, you can find the complete code of *solipsisSimulator.py* in the source code (See Download section).

In the code listing 1 we import all the needed library for:

- stun: for STUN procedure to discover the NAT presence and its configuration.
- punch: for hole punching through the Connection Broker

- defer: to recovery the signal of all asynchronous procedure

Listing 1: Import packets

```
import p2pNetwork.discover.stunDiscover as stun
import p2pNetwork.htcp.puncher as punch
import twisted.internet.defer as defer
```

In the listing 2 we start the procedure for the NAT discovery. This function look at NAT presence and try to find information about it. We set the procedure for the comebacking signal too.

Listing 2: Start discover code

```
deferr = defer.Deferred()

# Start to discover the public network address
d = stun.DiscoverAddress(stunPort, reactor)
d.addCallback(succeed)
d.addErrback(fail)
```

If the discovery procedure fail, because there is a firewall or for a network problem, the function *fail*, showed in listing 3 is called.

Listing 3: Failure callback function code

```
def fail(failure):
    """ Fail in STUN discover """
    print failure
```

If the discovery procedure succeeded the function showed in listing 4 is called. Now we know if there is a NAT and what type of NAT, so the eventually registration at the Connection Broker (CB) can be called. To register at the CB you have to call the *punch.HolePunching(port, reactor, netconf, id)* function. For details see the API documentation.

Listing 4: Succeeded callback function code

```
def succeed(address):
        """The STUN discovery succeeded.
        Registration to the Connection Broker"""

        self.host, self.port = address
        deferr.callback((self.host, self.port))
        print "discovevry found address %s:%d" \\
                % (self.host, self.port)

        # Several methods for network configuration
        #print stun.getNATType()
        #print stun.getPrivateAddress(),
        #print stun.getPublicAddress()
        stun.printConfiguration()
        netconf = stun.getConfiguration()
```

```
        # Registration to the CB for Hole punching
        d, puncher = punch.HolePunching \\
                        (port, reactor, netconf, id)
        d.addCallback(registrationMade)
        d.addErrback(fail)
```

Finally, after the registration at the CB, you can start to communicate, in UDP, with another endpoint. Call the *puncher.connectByURI(peerURI)* function and pass it the id of the other endpoint you want to connect to.

Listing 5: Callback function code after CB registration

```
def registrationMade((transport, puncher)):
        """ Called when the registration
        to the Connection Broker was done.
        Try to connect with the other peer """

        puncher.connectByURI(peerURI)
```

6.3.2 TCP connectivity

You can find the complete code of the examples of this section in *testTCPcl.py* source code file (See Download section).

TCP connectivity module is still in the test phase. You can test a TCP communication with or without spoofing. This module will remplace the UDP module integrating him. The compatibility with previous version is not assured.

The listing code 6 shows the needed packet imported for your TCP connection. There are two modules, one to allow sniffing packets and one to allow UDP communication with the CB or the other endpoint.

Listing 6: Import packets

```
import ConfigParser
import p2pNetwork.testTCP.sniffy as sniffer
import p2pNetwork.testTCP.udpSniffer as udp_sniffer
```

Before start TCP connection you have to load the configuration preset in the *test.conf* file. See the Set Configuration section.

Listing 7: Load configuration

```
# Load configuration
config = ConfigParser.ConfigParser()
config.read("test.conf")

myIP=config.get('myConf', 'myIP')
myPort=int(config.get('myConf', 'myPort'))
peerIP=config.get('peer', 'peerIP')
peerPort=int(config.get('peer', 'peerPort'))
```

Finally the *connect()*. Don't forget to establish an UDP communication with the CB or with the other endpoint and start sniffing in a thread using *udp_sniffer.UDP_factory()* and *sniffer.sniff(argv, udp_obj)* functions.

Listing 8: Start TCP connection

```
# Start to listen for UDP communication
udp_obj = udp_sniffer.UDP_factory()

# Start to sniff packets (run method in thread)
argv = ('', 'eth0', 'tcp port %d'%myPort)
sniffer.sniff(argv, udp_obj)

# Start TCP connection
reactor.connectTCP(peerIP, peerPort, \\
                EchoClientFactory(), 30, (myIP, myPort))
reactor.run()
```

6.4 Configuration files

Before to start use the ntcp library, you have to configure the configuration files for the different two modules.

- UDP module: *p2pNetwork.conf*

Listing 9: p2pNetwork.conf

```
[stun]
# STUN Server configuration
stunPort: 3478
otherStunPort: 3479
otherStunServer: 10.193.161.61:3478
otherStunServerPort: 10.193.161.61:3479

# Well known STUN Server for peer configuration
WellKnownStunServer: p-maul

# ConnectionBroker for peer configuration
[holePunch]
ConnectionBroker: 10.193.163.246:6060
```

- TCP module: *test.conf*

Listing 10: test.conf

```
[CB]
# Connection Broker
CBPort: 9999
CBIP: 10.193.163.246

[peer]
# The peer to connect to
peerIP: 10.193.167.86
peerPort: 50007

[myConf]
# My network configuration
myIP: 192.168.1.109
myPort: 50007

[UDPhole]
# For the UDP communication
udpPort: 9999
```

6.5 API Documentation

For the complete API documentation see the python documentation in /ntcp/trunk-/doc/apidoc/ directory

7 Important known problems

The *ntcp* is still in an initial phase, it isn't stable and dynamically configurable.

8 Version history

Release 0.1: First release of NTCP

- STUN implementation (client and server)
- UDP Hole punching (Puncher and Connection Broker)
- test for TCP NAT traversal

9 Contact information

Gaballo Luca
mail: gaballo.luca@rd.francetelecom.com
Tel: 01 45 29 63 03

10 Legal information

...