Independent degree project –
first cycle

Computer Engineering

NAT traversal techniques for MediaSense open source platform

Maksym Aryefyev
NAT traversal techniques for MediaSense open source platform
Maksym Aryefyev

MID SWEDEN UNIVERSITY
Department of Information Technology and Media

Examiner: Dr. Ulf Jennehag, ulf.jennehag@miun.se
Supervisor: Stefan Forsström, stefan.forsstrom@miun.se
Author: Maksym Aryefyev, maar1014@student.miun.se
Degree programme: International Bachelor’s Programme in Computer Engineering, 180 credits
Main field of study: Computer Engineering
Semester, year: VT, 2013
Abstract

This thesis project concerns NAT traversal techniques and their application to P2P networking with regard to MediaSense platform. Since MediaSense open source platform, developed by Mid Sweden University, utilizes the benefits of P2P networking, it also suffers from the drawbacks provided by NAT. The issue of NAT traversal is not trivial due to the fact that the behavior of NAT devices is not standardized and vendors are free to provide their own implementations. The common knowledge is, that at least four main types of NATs exist, differing in the filtering and mapping algorithms employed. NAT traversal techniques vary accordingly. No single technique can handle all the cases. Most of the techniques can handle up to three main types of NAT. The last type is usually used in large corporate networks and is called the Symmetric NAT. The most viable, and basically the only available technique for its traversal, is data relaying. This thesis builds a NAT traversal module for the MediaSense platform. The main purpose of this module is to provide seamless NAT traversal capabilities to the platform. The module does this in several steps: UPnP enabled device discovery, NAT type determination and data relaying via the proxy. Firstly the module attempts to discover the presence of a UPnP enabled Internet Gateway Device on the network. If such a device is present on the network, a port mapping can be created, making the node located behind NAT accessible from the public Internet. If a UPnP enabled device was not found, the module will try to determine the type of NAT used. Based on the type of NAT used, the module can transit to either the proxy mode or request assistance of the STUN server to keep the created mapping alive. The resulting chapters provide the reader with the output produced by each step, conclusions the author has made while working on this project and some general ideas on future work within the subject.

Keywords: MediaSense, Java, NAT, traversal, STUN, UPnP.
# Table of Contents

Abstract .................................................................................................................................................. iii  
Terminology ........................................................................................................................................ vi  
1 Introduction ...................................................................................................................................... 1  
1.1 Background .................................................................................................................................. 1  
1.2 Motivation .................................................................................................................................... 1  
1.3 Overall aim ................................................................................................................................... 1  
1.4 Concrete and verifiable goals ......................................................................................................... 1  
1.5 Scope ............................................................................................................................................ 2  
1.6 Ethics ............................................................................................................................................ 2  
1.7 Outline ......................................................................................................................................... 2  
1.8 Contributions ............................................................................................................................... 2  
2 Theory .............................................................................................................................................. 3  
2.1 Network Address Translation (NAT) ............................................................................................ 3  
2.2 NAT types .................................................................................................................................... 4  
2.2.1 Full-cone NAT ......................................................................................................................... 5  
2.2.2 Address-restricted NAT ......................................................................................................... 5  
2.2.3 Port-restricted NAT ............................................................................................................... 6  
2.2.4 Symmetric NAT ..................................................................................................................... 7  
2.3 NAT traversal techniques ............................................................................................................. 8  
2.3.1 Universal Plug and Play (UPnP) ........................................................................................... 9  
2.3.2 TU hole punching .................................................................................................................... 9  
2.3.3 Session Traversal Utilities for NAT (STUN) ...................................................................... 10  
2.3.4 Traversal Using Relay NAT (TURN) ................................................................................... 11  
2.4 MediaSense ............................................................................................................................... 11  
3 Methodology ................................................................................................................................... 12  
3.1 Investigate NAT penetration techniques ...................................................................................... 12  
3.2 Derive possible solutions for NAT penetration ........................................................................... 12  
3.3 Implement NAT penetration techniques for MediaSense .......................................................... 12  
3.4 Implement and enhance platform with load balancing ............................................................... 12  
3.5 Test and evaluate the resulting NAT penetration modules ......................................................... 13  
4 Design ............................................................................................................................................ 14  
4.1 Behind NAT ................................................................................................................................. 14  
4.2 UPnP .......................................................................................................................................... 15  
4.3 STUN .......................................................................................................................................... 19  
4.4 TURN (proxy mode) ..................................................................................................................... 21  
4.4.1 Load balancer ....................................................................................................................... 22  
5 Results ............................................................................................................................................ 23  
5.1 UPnP .......................................................................................................................................... 23  
5.2 STUN .......................................................................................................................................... 26  
5.3 Load balancer ............................................................................................................................. 27
6 Conclusions.................................................................................................................29
6.1 Investigate NAT penetration techniques..........................................................29
6.2 Derive possible solutions for NAT penetration.................................................29
6.3 Implement NAT penetration techniques for MediaSense..............................29
6.4 Implement and enhance platform with load balancing.................................29
6.5 Test and evaluate the resulting NAT penetration modules...........................30
6.6 General conclusion and future work...............................................................30

References.......................................................................................................................32
NAT traversal techniques for MediaSense open source platform
Maksym Aryefyev

Terminology

Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAT</td>
<td>Network Address Translation</td>
</tr>
<tr>
<td>P2P</td>
<td>Peer-to-Peer</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>STUN</td>
<td>Session Traversal Utilities for NAT</td>
</tr>
<tr>
<td>TURN</td>
<td>Traversal Using Relay NAT</td>
</tr>
<tr>
<td>TU</td>
<td>TCP/UDP</td>
</tr>
<tr>
<td>UPnP</td>
<td>Universal Plug and Play</td>
</tr>
<tr>
<td>XML</td>
<td>eXtensible Markup Language</td>
</tr>
</tbody>
</table>
1. **Introduction**

This work is the final thesis for the International Bachelor's Programme in Computer Engineering at Mid Sweden University within a timeframe of four hundred hours.

1.1 **Background**

Everyday computer networks embed themselves deeper and deeper into people’s daily routines. The number of connected devices owned by a single person can reach a dozen, if not more. Devices such as PCs, laptops, tablets, smart-phones, etc., used by everybody nowadays and which are capable of providing end users with different means of network connectivity. Flat screen TVs, for instance, can connect to Wi-Fi networks and allow the user to use a browser to surf the Internet. No wonder that due to high demand the number of connection enabled devices will to continue to grow in future.

1.2 **Motivation**

Some, if not all, of the connection enabled devices carry or can carry different types of sensors or actuators. A special purpose group of dedicated devices whose sole purpose is to aggregate/sense data and to provide it on request exists. Since there are a multitude of devices waiting to supply us with data, the suggestion can be made that it would be of interest to have some means of getting to that data.

Writing a separate program for every device is obviously not an option. Thus a unified platform which can be run on different devices, collect and transfer data from those devices to requesting an instance, is needed. Such a system is currently being developed by Mid Sweden University, and this projects aims to improve certain parts of this system.

1.3 **Overall aim**

The project’s overall aim is to investigate and provide solutions which will allow end nodes connected to the platform and positioned behind NAT to communicate seamlessly with other end nodes connected to the platform and positioned inside or outside NAT. Therefore the problem handled in this thesis is to address seamless NAT traversal for the Internet-of-Things so as to allow ubiquitous access to the information.

1.4 **Concrete and verifiable goals**

In order to achieve the problem, the project is divided into these goals:

1. Investigate NAT penetration techniques
2. Derive possible solutions for NAT penetration
3. Implement NAT penetration techniques for MediaSense
4. Implement and enhance the platform with load balancing
5. Test and evaluate the resulting NAT penetration modules

1.5 Scope
The scope of this thesis is limited to NAT traversal methods investigation and their possible application to the MediaSense open source platform. Due to time limitations and, more importantly hardware limitation, this work will not include in-depth testing of the provided solutions. The scope does not include any security issues that may arise while using NAT traversal solutions.

1.6 Ethics
Ethical aspects are not exactly applicable to this particular work, since it is quite “far” from end users. Nevertheless some parts of this work can be employed for malicious purposes. Thus the author would like to mention that this project was accomplished with the hope that it is going to be used for the benefit of the MediaSense platform. Any malicious use is prohibited and considered ill practice.

1.7 Outline
Chapter 2 provides the reader with some theoretical background, necessary in order to understand the issues associated with this work. Nevertheless, the advanced understanding of networking principles and the platform itself is required of a reader. Chapter 3 describes how the author will tackle milestones mentioned in the concrete and verifiable goals subsection of this chapter. Chapter 4 presents the implementation details. Chapter 5 provides the reader with the results with regard to milestones and overall aim of the project. Chapter 6 is a conclusion, covering the reflections about the thesis work itself and includes some ideas for future work.

1.8 Contributions
The research part of the work is conducted solely by the author. The coding/implementation part of the work is performed by the author. The weUPnP [1] open source library, published under LGPL license, is used as a basis for the UpnP part of the NAT traversal module.
2 Theory

This chapter will provide the reader with some theoretical background necessary to appreciate and understand the aims and goals of this project. This chapter provides only a limited amount of theory, which is highly project specific. In order to understand this chapter, the reader must have a deep knowledge of networking concepts itself, in addition, at least some, basic knowledge and understanding of NAT process and its application.

2.1 Network Address Translation (NAT)

Network address translation [2] functionality provided by routing devices enables computers on a private network to access computers on a public network. NAT is the process of modification of IP addresses and port information of a packet header while in transit in the routing device. In a typical scenario a routing device with enabled NAT functionality provides access to the Internet for an organization that uses private IPv4 addresses.

The following Figure 1 shows an example of a private network that uses a NAT enabled router to provide access to the Internet. Computers on the network have private IP addresses, the router has both public and private IP address, and, obviously, the web server has a public IP address.

![Figure 1: NAT](image)

Several types of NAT exist with the simplest bring a basic NAT or one-to-one NAT. In this type of NAT only the IP address and necessary check sums are changed, the remainder of the packet is left untouched. This type of NAT can be used to interconnect networks with incompatible addressing.
NAT traversal techniques for MediaSense open source platform
Maksym Aryefyev

However, the most common way to use a NAT is to hide the entire private network behind one public IP address or behind a small group of IP addresses. In order to facilitate the correct translation and successful delivery of packets to the requesting host, the NAT must alter TCP/UDP port numbers in outgoing communications and maintain a translation table so that the response packets are correctly translated back. This type of NAT is known as a PAT (port address translation), NAPT (network address and port translation) or one-to-many NAT. Figure 2 shows the mapping in a NAT enabled device.

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Source Address</th>
<th>Source Port</th>
<th>Destination Address</th>
<th>Destination Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP</td>
<td>192.168.0.10</td>
<td>3576</td>
<td>131.107.74.66</td>
<td>80</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Source Address</th>
<th>Source Port</th>
<th>Destination Address</th>
<th>Destination Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP</td>
<td>157.54.35.38</td>
<td>5000</td>
<td>131.107.74.66</td>
<td>80</td>
</tr>
</tbody>
</table>

**Figure 2: Port mapping**

The main problem associated with the NAT with regards to a MediaSense platform involves the communication between instances on a private network (the one behind NAT) and instances on a public network are only possible when the conversation request originates on the private network, since, only in this case, are the translation table records are generated. This makes it difficult for instances behind the NAT to accept incoming communications.

For example, a client computer with a browser located behind a NAT can access a web server located on a public network. At the same time, a client computer located on a public network cannot access a web server located on a private network behind a NAT. The second scenario is only possible if a router was pre-configured with port forwarding, allowing traffic which had originated in the outside networks to reach some hosts behind the NAT.

2.2 **NAT types**

Network Address Translation (NAT) can cause a certain amount of well-known issues, especially if peer-to-peer (P2P) communication is used. Peers involved into such communication may simply not be reachable at any public IP address.

NAT traversal is a general term for techniques that offer seamless traversal capabilities of a network address translation gateway. A network address translation gateway breaks end-to-end connectivity by intercepting and modifying traffic. Various NAT traversal techniques exist, but none can provide a single unified working solution. This is because the NAT behavior is not standardized, and each vendor can provide its own specific implementation.

Traditionally resources available on the Internet provide readers with the following types of NAT [3]:
NAT traversal techniques for MediaSense open source platform
Maksym Aryefyev 2013-06-29

- Full-cone NAT
- Address-restricted cone NAT
- Port-restricted cone NAT
- Symmetric NAT

2.2.1 Full-cone NAT

The private address 192.168.0.2:2210 is mapped to the public address 1.1.1.2:8801. Any packet sent from 192.168.0.2:2210 to a public network will have 1.1.1.2:8801 as the source address in the IP header. Any packet from any public address which arrives to 1.1.1.2:8801 is routed to 192.168.0.2:2210. Figure 3 illustrates full-cone NAT behavior.

![Figure 3: Full cone NAT](image)

2.2.2 Address-restricted NAT

The private address 192.168.0.2:2210 is mapped to the public address 1.1.1.2:8801. Any packet sent from 192.168.0.2:2210 to a public network will have 1.1.1.2:8801 as the source address in the IP header. An external host can send packets to 192.168.0.2:2210 by sending packets to 1.1.1.2:8801 but only if 192.168.0.2:2210 has previously sent a packet to the external host. External host ports are of no importance. Figure 4 illustrates the address-restricted NAT behavior.
NAT traversal techniques for MediaSense open source platform  
Maksym Aryefyev  
2013-06-29

2.2.3 Port-restricted NAT

The private address 192.168.0.2:2210 is mapped to the public address 1.1.1.2:8801. Any packet sent from 192.168.0.2:2210 to a public network will have 1.1.1.2:8801 as the source address in the IP header. The external host 1.1.1.30:1234 can send packets to 192.168.0.2:2210 by sending packets to 1.1.1.2:8801 only if 192.168.0.2:2210 has previously sent a packet to 1.1.1.30:1234. Figure 5 illustrates the port-restricted NAT behavior.
2.2.4 Symmetric NAT

Packets from the same inner host (IP:port) to the same outer host (IP:port) will have a unique public IP:port mapping. Packets from the same inner host (IP:port) but sent to a different outer host (IP:port) will have a different unique IP:port mapping. An outer host can send packets back only from the port on which it has received initial packets from the inner host. Figure 6 illustrates the symmetric NAT behavior.
2.3 NAT traversal techniques

While various NAT traversal techniques exist, the most viable (in the author's opinion) will be mentioned with regard to the MediaSense platform. Some of these techniques use the server only to establish the communication; others use the server to relay all the data or do not require a server at all.

The list of techniques which are briefly discussed in this section are:

- Universal Plug and Play (UPnP)
- TCP/UDP hole punching (TU)
- Session Traversal Utilities for NAT (STUN)
- Traversal Using Relay NAT (TURN)
2.3.1 Universal Plug and Play (UPnP)

Universal Plug and Play (UPnP) [4] is the architecture for peer-to-peer network connectivity of PCs, intelligent devices, etc. UPnP is built on Internet standards and technologies and it aims to enable devices to automatically connect with one another to make networking possible for more people. Networking products which support Universal Plug and Play technology will "just start working" when physically connected to the network. UPnP does not depend on any particular operating system, programming language, or physical medium.

As discussed previously, a usage of NAT can provide certain issues for the hosts positioned behind the NAT. In order to avoid requiring the end users to solve these NAT traversal issues manually, some vendors of Internet gateway device have included an application layer gateway support into their devices. In other words, some Internet gateway devices can automatically enable applications to go through the NAT.

UPnP-enabled NAT traversal devises can automatically solve issues provided by the NAT in the following areas: multi-player gaming, peer-to-peer connections, and real time communications. UPnP can provide the following methods: public IP address discovery, existing port mappings discovery, adding and removing port mappings, assigning lease times to mappings. Currently most of the large gateway vendors implement the UPnP NAT traversal solution in their products.

UPnP networking can be divided into steps. The first step is discovery, when an UPnP enabled device is added to the network it can advertise itself to other devices by means of an UPnP discovery protocol. The same protocol is used by the client devices to discover UPnP enabled devices on the network. The discovery message contains initial pieces of information about the device and the services it provides.

Step two is the description. When the client device has discovered a UPnP enabled device very little is still known about this device. To learn more and to interact with the UPnP enabled device, the client device must retrieve the description of the device and the services it provides. This is performed by retrieving the URL provided in the discovery message. The UPnP device and services description is expressed in XML format.

Step three – control. After the client device has retrieved the UPnP device description [5] it can start sending commands to the device’s services. In order to do this, the client device must send a suitable control message to the control URL of the required service. Control messages are expressed in XML format using SOAP. In response, the message service returns action specific values.

2.3.2 TU hole punching

UDP hole punching [6] is a technique for establishing a UDP connection between hosts located in private networks behind the NAT. Since the NAT char-
characteristics are not standardized, the technique is not applicable in all scenarios. This technique does not modify the NAT configuration, it merely exploits the server on a public network to introduce clients behind the NAT to each other.

Hosts on private networks connect to the server on a public network in order to determine their public IP address and port mapping. When a UDP connection with the server is established and mappings are learned, the host on a private network is required to send periodic keep-alive packets to the server. This is conducted in order to avoid UDP state mapping expiration on the NAT device, which usually occurs within a range of tens seconds to several minutes.

When hosts on private networks want to communicate with each other, they ask the server for the connection parameters of the other host and/or cause the server to inform the other host of its connection intentions (STUN, discussed in the next section), depending on the type of NAT used. Further actions of the hosts and the success of the operation merely depend on the type of NATs used and these were discussed earlier in this chapter. Depending on the filtering and mapping restrictions in relation to the connection for the NATs this can prove to be either extremely easy or almost impossible. However, it is certainly the case that the UDP hole punching technique does not work (nothing is impossible, but in this case a complex educated guessing algorithm must be employed, providing no guarantee of success) for the symmetric NAT, which are often used by large corporations.

The TCP hole punching [7] is a technique for establishing a TCP connection between hosts located in private networks behind the NAT. This technique heavily relies on the type of port allocation used by the NAT, which is either predictable or not predictable. Additionally, it introduces some overhead complexity by requiring both a TCP sequence and ACK number synchronization. Therefore this technique will not be studied and employed in this thesis.

### 2.3.3 Session Traversal Utilities for NAT (STUN)

STUN [8] is a system of probe messages aimed to examine the interchange between a host located behind the NAT and STUN server located on a public network. The STUN server must have two IP addresses and must be able to respond on two UDP ports for each IP address. The protocol itself is rather simple, it compares the private IP address and port number carried in the payload with the header values in order to determine the type of NAT which lies on the path between the host and the server.

In order to determine the type of NAT, the client request message might include a special flag to enable the server to respond using an alternative source address. If the client receives the response, it can be concluded that the client is behind full-cone NAT. If no response was received, further actions might be taken to determine which other type of NAT is being used.
Periodic packet exchange between a client and the server can also assist to determine the timer used by the NAT to maintain its mapping tables. STUN is the technique which works with three primary types of NAT, but fails to traverse a symmetric NAT.

2.3.4 Traversal Using Relay NAT (TURN)

Traversal using relay around NAT [9] is a technique which allows a host behind a NAT to receive incoming data by means of either a TCP or UDP protocol. This technique is the most beneficial for hosts located behind a symmetric NAT, because other available techniques are most likely to fail the symmetric NAT traversal. The TURN technique solely relies on data relaying through the server residing on the Internet. The positive side of this technique is that it can almost always provide connectivity, but it does come at a price, namely for additional traffic overhead and much longer latencies which may be mission critical nowadays.

The process begins when a client wants to connect to a peer for a data exchange, but fails because both client and peer are behind the NATs or a peer is behind a NAT. In this case the instance located behind the NAT must create a constant tunnel to the server and perform send-receive operations via this tunnel. This technique successfully traverses a symmetric NAT since all the data sent to the instances located behind the NAT is delivered (relayed) via a public server.

2.4 MediaSense

The MediaSense platform aims to interconnect sensors and actuators together. In order to accomplish these goals it uses P2P networking and its own implementation of the RUDP protocol as the main means of delivery. Since in P2P networking every node acts as a server and a client simultaneously, it is heavily affected by the presence of the NAT on the network. The issue is that nodes located behind the NAT can lose connectivity with other nodes located on the public Internet or behind other NATs.

Currently, the MediaSense platform has only one way of solving the issue of connectivity loss – data relaying via a proxy. Thus the research is required to provide alternative solutions to this problem. This project aims to research and implement possible solutions, which can be beneficial to the MediaSense platform.
3 **Methodology**

In general the author plans to attack each goal individually in order of appearance. The author also plans to spend a substantial amount of time on overall problem investigation, which will, in the main, include studies of available documentation regarding the matter. In order to test possible solutions in general, the author plans to build one or several prototypes disconnected from the platform itself. After theoretical investigation and initial phase of prototyping the author plans to implement changes to the platform.

3.1 **Investigate NAT penetration techniques**

In order to attack this goal, the author will investigate available online resources regarding existing techniques of NAT penetration: wiki pages, articles, etc. the author will also attempt to read networking books and manuals explaining NAT operational principles. The goal here is to obtain a complete understanding of how NAT operates and what viable methods of penetration exist with regards to MediaSense platform.

3.2 **Derive possible solutions for NAT penetration**

In order to attack this goal, after studying NAT operational principles, existing types of NAT and possible techniques of seamless NAT traversal, the author will attempt to evaluate which solution or solutions are viable and makes sense with regards to the existing MediaSense communication mechanism. The author will also closely study the NAT traversal solution currently present in the platform. Current NAT penetration solution uses a proxy server to relay all the data, both incoming and outgoing. The conclusions will be based on the studies performed to accomplish goal number one.

3.3 **Implement NAT penetration techniques for MediaSense**

In order to attack this goal the author will attempt to implement NAT penetration solutions derived in goal number two. All the new solution will be implemented as a standalong application disconnected from the MediaSense platform. This will be conducted in order to facilitate the testing and evaluation of these solutions with regards to the benefits and viability of these solutions to the MediaSense platform. The general idea is to provide a working solution which will allow the end nodes connected to the platform to communicate with each other, regardless of whether there is or is not any NAT.

3.4 **Implement and enhance platform with load balancing**

The current NAT penetration module implemented in the platform uses the bootstrap as a single available the proxy server. In order to attack this goal the author will remove proxy server functionality from the bootstrap and will
provide a load balancer server which will allow the end nodes to register themselves as proxies with the load balance. End nodes located behind the NAT will gain the ability to request proxy server credentials from the load balancer server.

### 3.5 Test and evaluate the resulting NAT penetration modules

In order to attack this goal the author will perform testing of the NAT penetration and load balancing functionality. Since the NAT traversal system may include several layers and mechanisms, each mechanism will be evaluated separately. All flaws, defects, undefined conditions and general doubts will be noted and included in the report.
4 Design

This chapter provides the reader with an architectural design of the provided solution. Figure 7 demonstrates the logic flow of the NAT traversal model, designed for the MediaSense platform.

![NAT Traversal module](image)

### 4.1 Behind NAT

First of all, when an end node is being deployed it must establish whether it is located behind a NAT enabled device and thus whether or not the communication is limited by this NAT. Since the main purpose of the node is to provide and/or collect data from other nodes, it must have the ability of seamless communication with other nodes.
On a platform startup, it iterates thought all the interfaces, checking whether any has a public IP address configured. If none of the interfaces is configured with a public IP address, the system concludes that it is located behind a NAT enabled device and proceeds with the NAT traversal module.

If the system finds a public IP address configured on any network interface it initiates a normal connection to the bootstrap or server depending on a current configuration.

4.2 UPnP

If the module has determined that it is located behind a NAT enabled device it can switch to the next step. By means of the UPnP protocol, the module must firstly check if any UPnP enabled devices are present on the network.

Thus, according to UPnP-arch-DeviceArchitecture-v1.1-20081015.pdf available on upnp.org if a device wants to search for other devices, it is required to send a multicast request using method M-SEARCH in the format shown on Figure 8:

```
M-SEARCH * HTTP/1.1
HOST: 239.255.255.250:1900
MAN: "ssdp:discover"
MX: seconds to delay response
ST: search target
USER-AGENT: OS/version UPnP/1.1 product/version
```

Figure 8: M-SEARCH header

The UPnP enabled device will respond with a unicast UDP packet to the source IP address and the port that sent the multicast request, only if the ST header field of M-SEARCH request holds one of the next values: “ssdp:all”, “upnp:rootdevice” or “uuid:” plus UUID that must exactly match the one advertised by the device. It will also respond if the M-SEARCH request matches a device type or service type provided by the device.

The response is sent in the format shown on Figure 9:
NAT traversal techniques for MediaSense open source platform
Maksym Aryefyev

After the UPnP enabled device has been discovered, the module has only a very limited knowledge about the device and the services it provides. The most important piece of information provided by the device discovery response message is the LOCATION header field, which supplies the URL to the device’s UPnP description.

In order to learn more about the device and to interact with the device, the module must retrieve a description from the provided URL. The UPnP device description is partitioned into two sections: device description and service description.

The device description includes manufacturer specific information about the device and the service description section holds information about each service provided by the device, which usually has the following fields: service type, service name, a service description URL, a control URL, and a eventing URL.

All descriptions are presented in XML syntax and are usually based on the UPnP Device and Service Templates produced by the UPnP forum working committee. The UPnP service description includes a list of commands to which the service must respond, and arguments for each of these commands.

The retrieval of the UPnP device and service descriptions is conducted by means of a GET request from the HTTP protocol. The UPnP device description is shown in Figure 10 below:

```
HTTP/1.1 200 OK
CACHE-CONTROL: max-age = seconds until advertisement expires
DATE: when response was generated
EXT:
LOCATION: URL for UPnP description for root device
SERVER: OS/version UPnP/1.1 product/version
$T: search target
$US: composite identifier for the advertisement
TOOTID.UPNP.ORG: number increased each time device sends an initial announce or an update message
CONFIGID.UPNP.ORG: number used for caching description information
SEARCHPORT.UPNP.ORG: number identifies port on which device responds to unicast M-SEARCH
```

Figure 9: Search response header
Now when the NAT traversal module has knowledge of the device and services it provides, it can ask these services to invoke actions and obtain the feedback from the device indicating the results of the actions. If the module wants to invoke an action on the device’s service, it sends a suitable control message to the control URL obtained from the service description, providing all the required
arguments inside this control message. When the action has succeeded or failed, the service returns results or errors to the requesting instance. An action invocation message template is shown in Figure 11 and an action response message template is shown in Figure 12.

The UPnP module developed in this thesis uses next actions for WANPPPConnection service described in the UpnP-gw-InternetGatewayDevice-v2-Device.pdf [10]: GetSpecificPortMappingEntry to check whether a mapping exists on the device, AddPortMapping to add a port mapping, DeletePortMapping to delete a port mapping, GetExternalIPAddress to determine the public IP address.

AddPortMapping action, for instance, has the list of arguments shown in Table 1 and described in the UpnP-gw-WANIPConnection-v2-Service.pdf [11]:

---

**Figure 11: Action invocation template**

```xml
POST path control URL HTTP/1.0
HOST: hostname:portNumber
CONTENT-LENGTH: bytes in body
CONTENT-TYPE: text/xml; charset="utf-8"
USER-AGENT: OS/version UPnP/1.1 product/version
SOAPACTION: "urn:schemas-upnp-org:service:serviceType:v#actionName"

<?xml version="1.0"?>
<s:Envelope
 xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:soapBody="http://schemas.xmlsoap.org/soap/encoding/"  
<s:Body>
 <u:actionName
 xmlns:u="urn:schemas-upnp-org:service:serviceType:v"
><argumentName>in arg value</argumentName>
</u:actionName>
</s:Body>
</s:Envelope>
```

---

**Figure 12: Action response template**

```xml
HTTP/1.0 200 OK
CONTENT-TYPE: text/xml; charset="utf-8"
DATE: when response was generated
SERVER: OS/version UPnP/1.1 product/version
CONTENT-LENGTH: bytes in body
<?xml version="1.0"?>
<s:Envelope
 xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:soapBody="http://schemas.xmlsoap.org/soap/encoding/"  
<s:Body>
 <u:actionNameResponse
 xmlns:u="urn:schemas-upnp-org:service:serviceType:v"
><argumentName>out arg value</argumentName>
</u:actionNameResponse>
</s:Body>
</s:Envelope>
```
### Table 1: Action arguments list

<table>
<thead>
<tr>
<th>Argument</th>
<th>Direction</th>
<th>relatedStateVariable</th>
</tr>
</thead>
<tbody>
<tr>
<td>NewRemoteHost</td>
<td>IN</td>
<td>RemoteHost</td>
</tr>
<tr>
<td>NewExternalPort</td>
<td>IN</td>
<td>ExternalPort</td>
</tr>
<tr>
<td>NewProtocol</td>
<td>IN</td>
<td>PortMappingProtocol</td>
</tr>
<tr>
<td>NewInternalPort</td>
<td>IN</td>
<td>InternalPort</td>
</tr>
<tr>
<td>NewInternalClient</td>
<td>IN</td>
<td>InternalClient</td>
</tr>
<tr>
<td>NewEnabled</td>
<td>IN</td>
<td>PortMappingEnabled</td>
</tr>
<tr>
<td>NewPortMappingDescription</td>
<td>IN</td>
<td>PortMappingDescription</td>
</tr>
<tr>
<td>NewLeaseDuration</td>
<td>IN</td>
<td>PortMappingLeaseDuration</td>
</tr>
<tr>
<td>NewReservedPort</td>
<td>OUT</td>
<td>ExternalPort</td>
</tr>
</tbody>
</table>

### 4.3 STUN

If the module attempts to discover an UPnP enabled device on the network and fails, it will proceed to the next step. In order to determine the further course of actions the module must determine the type of NAT that it is behind.

A node, which must determine the type of NAT that it is located behind, can achieve this by communicating with the STUN server. The STUN server is a computer which has at least two public IP addresses (see Figure 13) assigned and is running dedicated software.

The STUN server solution developed within this thesis uses three UDP sockets configured using two public IP addresses. The main socket listens for incoming initial packet and when such a packet is received, the STUN server creates a client record and stores the connection information in this record. Figure 14 shows the logic of the STUN server.
The STUN server responds to the initial packet with two packets. The first carries the STUN server’s socket configuration data and is sent from the main socket while the second is sent using the socket configured to use the second IP address. This is conducted to determine whether the client is behind a full-cone
NAT traversal techniques for MediaSense open source platform
Maksym Aryefyev 2013-06-29

NAT, meaning it can receive packet from IP addresses that it has never initiated any communication.

If the client receives the second packet it responds back to the STUN server with a confirmation. The client now knows that it is behind a full-cone NAT, meaning that as long as the “hole” is open, a mapping exists in the router and thus anyone can access it using the public IP and port.

If the client does not send a confirmation within a second, the STUN server sends a message to check whether the client is behind an address restrictive NAT. This message is sent from the socket configured with the same IP address as the main socket but with a different port number.

If the client can receive this message then it means that it is behind an address restrictive NAT. In this case, as well as in all further cases, the only option is to use a STUD (proxy) solution, because this type of NAT can be traversed by the technique called TU hole punching. This technique is not applicable to the MediaSense platform because there is no means of guaranteeing that the node will fully participate in DHT.

If the client does not respond to the server with a confirmation message then the STUN server will send yet another message to determine whether the client is behind a port restrictive or a symmetric NAT. All three last three cases will lead the module to the final option available – data relaying by means of a proxy server.

Address and port restrictive NATs can be traversed by the TU technique, but unfortunately this technique is not applicable in the case of the MediaSense platform due to the internal mechanism of communication between the end nodes.

4.4 TURN (proxy mode)
If there are no UPnP devices or the detected NAT type is not full-cone, the only option remaining is to use a relaying server. A node located behind the NAT establishes a permanent TCP connection with the dedicated proxy server. When the connection is established, the proxy server creates a normal MediaSense UDP connection and acts like a virtual interface, accepting and sending all the packets destined to the node located behind the NAT. All the received packets are passed by to the node via the TCP tunnel.

The development of this solution was not a part of this thesis.

The main goal for this thesis with regards to proxy functionality was to remove the Base64 serialization. So now the solution employs a Java native serialization and uses ObjectInputStream/ObjectOutputStream for object delivery.
4.4.1 Load balancer

Having only one proxy server in the system, which can consist of hundreds if not thousands of nodes, is obviously a bottleneck. Thus the system requires the ability to have numerous proxy servers and a way to distribute the load among them.

Since the proxy server solution is implemented as an addin in the MediaSense platform, any node which is not placed behind the NAT can be a proxy server. While several nodes can act as a proxy, the means to distribute the connection credentials of these proxies among the nodes located behind the NAT and which is willing to use proxy’s services is required.

As a part of this thesis, a load balancer was developed. The load balancer is a server running on a bootstrap. This server behaves as a middle man. Whenever a node decides to act as a proxy server, it sends a message to the load balancer in order to register itself. The load balancer responds with a confirmation. Hence, from now on, the proxy server can expect incoming connections.

If a node located behind the NAT reaches the point when the only option left is to use a proxy server, it can send a proxy discovery request to the load balancer. The load balancer will respond to such a request with a message carrying the connection credentials of a proxy server.

When the load balancer receives a proxy discovery request it traverses through the collection which holds the connection credentials of every registered proxy, and the number of times a particular set of credentials was returned to a proxy discovery request. The load balancer finds the credentials which were returned the least number of times and returns them to the requesting node.
5 Results

This chapter describes the results of the project. Each piece of the NAT traversal module has been designed as a standalone application, so this chapter will provide the reader with screen shots of the console outputs of every designed module.

5.1 UPnP

The UPnP functionality is dedicated to UPnP enabled device discovery on the network and if such a device was found, to port mapping, so that the hosts on the Internet can access the services provided by a machine located behind the NAT.

The UPnP device discovery request is sent as a UDP multicast, and all the devices receiving such a request and which can identify themselves as a target device for this request must answer within a timeframe of 1 to 3 seconds. The timeframe is randomly generated by the device.

Figure 15 illustrates the output if no UPnP enabled device was found on the network. That means that no device has answered the discovery request within the required timeframe.

![Output: UPnP device was not found](image)

When an UPnP enabled device is found, the description information about the device itself can be retrieved (marked as A on Figure 16), all existing port map-
pings are retrieved (marked as B on Figure 16) and an attempt to map new ports is made (marked as C on Figure 16). Figure 16 shows that the new mapping has failed because a mapping already exists in the router’s table.

Figure 16: Existing mappings and new mapping attempt

Figure 17 illustrates the successful new mapping.
The UPnP part of the NAT traversal module does function as intended and serves its purpose. This part of the module clearly provides the best and easiest way in relation to NAT traversal, which requires the minimal effort from the MediaSense platform and does not require a “middle man”. Similar traversal systems are heavily employed by most P2P applications: Skype, all types of torrents, online games.

Unfortunately this approach is mostly applicable to home grown networks, since the UPnP is enabled by default in home segment network equipment. Middle to large scale networks, in general, do not provide UPnP interfaces, due to absence of such functionality in the network equipment or as they have been disabled by the network administrator.
Provided functionality is derived and partially relies on the weUPnP open source library published under LGPL. The author of this thesis completely re-wrote the device discovery functionality, while retaining the part which uses the SOAP protocol as the main means of command handling and execution.

5.2 STUN

The goal of STUN server is to assist a node located behind a NAT to determine its type and to proceed in accordance with that knowledge.

The type of NAT is determined in a series of packet exchanges, which follow the logic of the NAT behavior described in Chapter 2. According to that behavior, if certain packets are received or are not received, a corresponding conclusion of the type of NAT enabled device present in the network is made.

Figure 18 shows the output of the client if it receives no response to the initial discovery packet within the timeframe of 3 seconds.

```
54  clientSocket.receive(packetR);
55  STUNMessage messageReceived = STUNMessage.deserializeMessage(packetR.getData());
56  if (messageReceived.getSessionId().equals(sessionID)) {
57       if (messageReceived instanceof STUNInitialResponse) {
58           System.out.println("Received initial response from " + packetR.getSocketAddress());
59           STUNInitialResponse message = (STUNInitialResponse) messageReceived;
60       }
61       server1 = message.getServer1();
62       server2 = message.getServer2();
```

Figure 18: STUN discovery failure

Figure 19 shows the series of steps performed by the STUN server to determine the type of NAT that the client is behind.
Figure 19: NAT determination, STUN server side

Figure 20 shows the series of steps performed by the client to determine the type of NAT it is behind.

Figure 20: NAT determination, client side

The developed system behaves in accordance with design. In order to function STUN server must have at least two public addresses assigned, multi-homing can be used. Figures 18 to 20 show the test case when the STUN server is located elsewhere and the client is located behind Mid Sweden University’s NAT in Sundsvall. The type of NAT used by the university is port restricted NAT.

5.3 Load balancer

The purpose of the load balancer is to distribute the traffic among the nodes, which chose to provide proxy functionality to the other nodes located behind the NAT.

So the node which chooses to be a proxy must register itself with the load balancer and start waiting for an incoming connection. Figure 21 shows the proxy registration.
The load balancer works as designed, it can provide the requesting instance with actual proxy connection credentials. Load balancing logic is relatively straightforward and the proxy with the minimum number of clients is returned upon the request.
6 Conclusions

This chapter will provide the reasons why this thesis project should be considered completed and successful.

6.1 Investigate NAT penetration techniques

This goal was successfully completed; author has studied, gained knowledge and understanding of the operational principles of different types of NATs available as of today. This was conducted by an in-depth examination of the available online resources: wiki pages, articles as well as existing books and manuals describing the issue. Based on the knowledge about different types of NATs, the author has investigated the available traversal techniques and solutions with regards to every major type of NAT. This gained knowledge has been successfully applied during the goal execution time.

6.2 Derive possible solutions for NAT penetration

This goal was successfully completed; the author has closely evaluated the existing penetration techniques for viability and general common sense with regards to usefulness and the ability to benefit the MediaSense open source platform. The rule of thumb used by the author while evaluating the possible techniques was the need to retain MediaSense's existing communication mechanism, which requires constant communication between the end nodes in order to maintain DHT. Knowing this, the author has excluded several possible techniques widely employed in P2P networking. The author has also examined existing MediaSense NAT traversal solution. The derived solutions were implemented during the following goal execution.

6.3 Implement NAT penetration techniques for MediaSense

This goal was successfully completed; author has implemented all the solutions derived in the previous goal. All the solutions are implemented as standalone applications, each of which can be easily embedded into the existing MediaSense platform. All the provided solutions are beneficial to the MediaSense platform, since they may greatly simplify the NAT traversal for the end nodes, located behind the NAT enabled device, and may assist to avoid the extensive overhead introduced by data relaying. While executing this goal, the author has also removed the Base64 serialization from the existing MediaSense's NAT traversal solution and substituted it with native Java's serialization.

6.4 Implement and enhance platform with load balancing

This goal was successfully completed; The idea involved in this goal is that any node which is running can act as a proxy server for the end nodes located behind the NAT. Thus the author has implemented the load balancer, which is run
on the same machine with the bootstrap. All the nodes which want to provide proxy services to other nodes can advertise themselves by registering with the load balancer. Those nodes, which must use the proxy services can request connection credentials from the load balancer. Such an enhancement is very beneficial to the platform since it removes the bottleneck introduced by a single proxy server.

6.5 Test and evaluate the resulting NAT penetration modules

All the provided solutions were successfully tested in the university environment. All the solutions, behave as intended and have proved their usefulness to the platform. Unfortunately, testing may not be considered complete, since some of the tests required at least several nodes with assigned public IP addresses. All the performed tests had at most one node/server/bootstrap with an assigned public IP address. However, this should not be considered as a serious flaw. The overall evaluation should be considered as having been successful.

6.6 General conclusion and future work

In general, it is the authors' opinion that the provided solutions may greatly simplify NAT traversal under certain conditions. For instance, the UPnP part of the module can assist to avoid unnecessary overheads in data delivery delays and bottlenecks introduced by data relaying. The UPnP part of the module simply commands the router to accept incoming connections on certain ports and routes packets to the machine located on the private network.

Implementation of the UPnP module from scratch is somewhat onerous as it requires a substantial time and effort. Thus, the author of this thesis chose to use the weUPnP open source library written in Java as a base or starting point for the UPnP part of the module. The author suggest the UPnP module/library implementation in Java as a project for future work. This project could be standalone or as a part of the MediaSense platform development. Competence in the following areas is required: HTTP protocol, since device advertisement and discovery messages are basically HTTP headers delivered over UDP; XML and SOAP since this technology set is used to issue commands on the UPnP enabled device.

The STUN server part of the module can successfully determine the type of NAT used. However, currently, due to particular issues of DHT maintenance, the MediaSense platform can benefit only if the node is behind a full-cone NAT. The outgoing packet creates the mapping in the routing device, which accepts any incoming UDP connection. Address and port restrictive NATs can be penetrated by the set of techniques called TU hole punching. This technique does not appear to work for the platform, since in order to communicate, the end nodes must coordinate their actions via the STUN server. The node behind the NAT must be the instance initializing the connection. Thus, as a future work, the author suggest, further investigation of the matter, and possible changes to the platform.

30
The load balancing functionality introduces a significant improvement to the platform, which helps to distribute the proxy effort among the nodes. As a future work, the author suggests further improvement to the load balancer by the introduction of functionality which will check the current state of the proxies: the number of connected instances, bandwidth consumed, etc. With all that data the balancer will be able to make more intelligent decisions regarding which proxy connection credentials to return upon the proxy discovery request. Although this offers benefits, such functionality also has a drawback in that it introduces extra traffic into the network.
References


