

UNIVERSITY OF TARTU  
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE  
Institute of Computer Science

Gert Palok  
**Port Forwarding with  
Universal Plug and Play**  
Bachelor Thesis (6 EAP)

Supervisor: Ulrich Norbistrath, PhD

Author: ..... “.....” June 2010

Supervisor: ..... “.....” June 2010

Allowed to defence

Professor: ..... “.....” June 2010

TARTU 2010

# Contents

<b>Introduction</b>	<b>3</b>
<b>1 Motivation</b>	<b>6</b>
<b>2 Existing work</b>	<b>8</b>
2.1 libupnp . . . . .	8
2.2 CyberLink for Java . . . . .	8
2.3 UPNPLib . . . . .	8
2.4 linux-igd . . . . .	9
2.5 JPunch . . . . .	9
2.6 NAT-PMP . . . . .	9
<b>3 UPnP</b>	<b>10</b>
3.1 Discovery . . . . .	10
3.2 Description . . . . .	12
3.3 Control . . . . .	13
<b>4 Portforward</b>	<b>16</b>
4.1 Prototype . . . . .	16
4.2 The program . . . . .	17
4.3 Issues . . . . .	17
4.4 Usage . . . . .	18
<b>Conclusion</b>	<b>19</b>
<b>Pordisiire universaalse isehäälestumisega</b>	<b>20</b>
<b>Bibliography</b>	<b>21</b>

# Introduction

Internet Protocol version 4 (IPv4) is the fourth revision of the Internet Protocol (IP). IPv4 uses 32-bit addresses, limiting the address space to  $2^{32}$  possible unique addresses. Every host on an IPv4 network is assigned an IPv4 address that is used to communicate with other hosts, either on the same network or globally. Each IPv4 packet has a source and destination address.

User Datagram Protocol (UDP) and Transmission Control Protocol (TCP) are transport layer protocols. UDP and TCP use the notion of port to identify the application endpoint on the host. A port is a 16-bit unsigned integer. Each UDP and TCP packet has a source and destination port.

As there are not enough addresses to uniquely identify every host on the Internet, Network Address Translation (NAT) is used to masquerade a Local Area Network (LAN) with hosts using private addresses into a single host on the Wide Area Network (WAN). Private addresses are assigned from special ranges of addresses and are unique in that LAN, but reusable across different LANs. Hosts behind a NAT can address and therefore send packets to hosts in the WAN, but the hosts on WAN are unable to send packets to hosts behind a NAT, because a private address can belong to any number of hosts. The devices performing NAT are known as Network Address Translators and are also referred to as NATs, the distinction is based on context.

When UDP or TCP is used over IPv4, endpoints are usually described with *address:port* and packets with (*source address:source port, destination address:destination port*). Consider the case where *10.0.0.1* in the LAN is behind a NAT-enabled *1.0.0.3* and *1.0.0.2* is on the WAN. Figure 1 on the following page illustrates the flow of packet from *10.0.0.1:1* to *1.0.0.2:2*. *10.0.0.1* sends a packet to *1.0.0.2*, it arrives at *1.0.0.3* as (*10.0.0.1:1, 1.0.0.2:2*). The NAT remembers this information and rewrites this packet into (*1.0.0.3:3, 1.0.0.2:2*) and passes it on. When *1.0.0.2* receives the packet, it sees *1.0.0.3* as the sender. For the return packet, illustrated in Figure 2 on the next page, *1.0.0.2* sends (*1.0.0.2:2, 1.0.0.3:3*). When the packet arrives at *1.0.0.3*, it recalls that previously a packet travelled from *10.0.0.1:1* to *1.0.0.2:2* and concludes that this must be a return packet, so it rewrites the packet to (*1.0.0.2:2, 10.0.0.1:1*) and passes it on to the LAN, where it reaches *10.0.0.1*. [1]

However, if *1.0.0.2* would have sent the first packet, even if it knew *10.0.0.1* was behind *1.0.0.3*, the NAT would not have had enough information to conclude that the packet should have been delivered to *10.0.0.1*. [1] It gets even

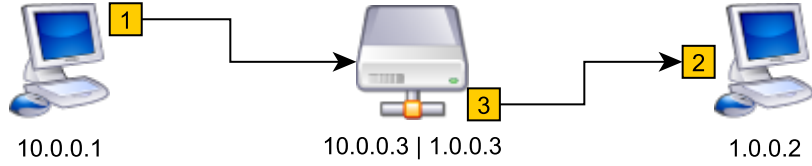


Figure 1: Path of packet traveling from 10.0.0.1:1 to 1.0.0.2:2

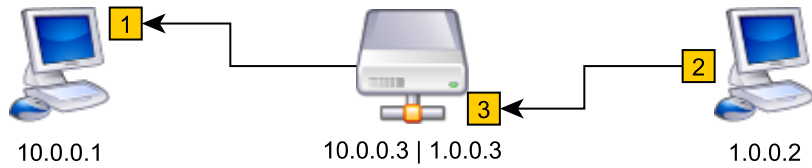


Figure 2: Path of return packet for Figure 1

worse when both parties are behind a NAT, as neither can successfully initiate a connection. With the rapid growth of the internet, this has become a fairly common situation. To remedy this, NATs often offer some sort of a configuration protocol to manipulate their mappings of *external address:external port* to *internal address: internal port*. Unfortunately, each vendor provides its own configuration protocol, most common being telnet and web-based user interfaces.

Relaying is a technique whereas two hosts A and B, each behind a different NAT, communicate with each other through an intermediate host C, known as the relay, which is accessible for both. This type of communication is usually managed by software in the application layer, as on the transport layer there are two separate connections, one between host A and C, the other between host B and C, as seen in Figure 3.

Friend-to-Friend Computing Framework (F2F Computing) is a platform for

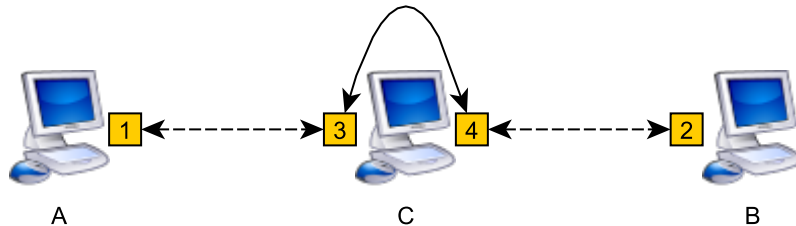


Figure 3: Relaying. Connection between A and B is relayed over C

writing distributed applications and services. It offers a middleware on top of the multi-protocol instant messengers (IM). Messaging networks are known to be slow, due to usually being relayed through a central server or other nodes. Even when this is not the case, messaging protocols generate overhead. Network performance could be improved by establishing direct connections between peers.[2]

Universal Plug and Play (UPnP) is a set of networking protocols designed to bring easy-to-use, standards-based connectivity between a wide variety of devices. Among other things it provides configuration of the aforementioned mappings, also known as port forwarding. UPnP combines widely supported technologies and is thus a good choice for applications that need to receive incoming traffic.[3]

The aim of this thesis is to analyze how UPnP can be used to establish direct connections between peers and implement a tool for this purpose. I do not provide a sufficient solution to the direct connectivity problem in general, but rather a simple method that could be attempted before other, more complex low-level NAT-traversal techniques, like UDP hole punching.

# Chapter 1

## Motivation

F2F Computing uses IM communication channels to create a peer-to-peer (P2P) network. Performance is usually not as critical as availability for IM networks, so intermediate nodes are often used to relay connections between peers behind NATs. These kind of channels are sufficient for control protocols, but lack the performance for high data transfers. Thus, external channels with better characteristics are needed in addition to what are already provided by the IM.

Direct connections are a natural choice for data transfer needs. However, due to the varieties in network topology and devices, direct connections might not be always available. As we learned before, NATs pose the most prominent threat, the inability to address the true destination. There are basically two ways to resolve this problem.

First, if possible, we could explicitly tell the NAT where to deliver a packet matching a certain criteria. Most often a port on the NAT is configured to map, or forward to a machine and port on the LAN. Vendors usually provide their own protocol for this purpose. In addition, there are standardised protocols: NAT Port Mapping Protocol (NAT-PMP) and the more generic UPnP.

Secondly, we could use the return packet handling to actually establish a connection. This is known as UDP or TCP hole punching. This kind of NAT traversal does not work with all types of NATs as their behaviour is not standardized. It relies on how well the behaviour of the NAT can be externally assessed and predicted.

The former methods have many advantages over the latter. Low-level packet manipulation operations may require elevated permissions, depending on the platform. Also, the libraries for those operations usually differ from one operating system to another, making porting much more difficult. On the other hand, we might just have an interface for doing exactly what we need. As each vendor tends to have their own version of an administration interface, we would soon run into trouble supporting all of them. Usually some kind of authentication is used to prevent unauthorized modifications, which complicates automatic configuration by software. NAT-PMP is a specialized protocol that is well-defined and efficient. It would be a good choice for port forwarding needs, but suffers from

a lack of support. UPnP uses HyperText Transport Protocol (HTTP), Simple Service Discovery Protocol (SSDP), Extensible Markup Language (XML) and SOAP. It is designed for more general networking and thus has more overhead than other protocols, but since the technologies are old and well established on many platforms, it has gained widespread support.

## Chapter 2

# Existing work

In this chapter I will give a short overview of existing work on the subject of establishing direct connections. This is not a list of all the existing work, just some honorable mentions.

### 2.1 libupnp

The portable SDK for UPnP Devices (libupnp) provides developers with an API and open source code for building control points, devices, and bridges that are compliant with version 1.0 of the UPnP Device Architecture Specification and support several operating systems like Linux, \*BSD, Solaris and others.[4]

It is a very well written library that comes with its own embedded XML library, Linux DOM2 XML Parser Version 1.2. Maybe it would have been a better idea to use another, popular XML library, as there are already plenty to choose from. The API is somewhat low-level.

### 2.2 CyberLink for Java

CyberLink for Java is a development package for UPnP™ developers. CyberLink controls these protocols automatically, and supports to create your devices and control points quickly. CyberLink is also available in C, C++, Objective C and Perl.[5]

It is a fully functional UPnP library with good high-level abstraction. The networking code is somewhat poor however, but it's easy to fix. Also, some classes are too tightly coupled.

### 2.3 UPNPLib

UPNPLib is a UPnP library written in Java to discover, interact with and monitor UPnP devices. The project has been unmaintained since November

2006.[6]

## 2.4 linux-igd

Linux UPnP Internet Gateway Device is a daemon that emulates Microsoft's Internet Connection Service (ICS). It implements the UPnP Internet Gateway Device specification (IGD) and allows UPnP aware clients, such as MSN Messenger to work properly from behind a NAT firewall.[7]

## 2.5 JPunch

JPunch is a UDP hole punching library in Java. UDP hole punching tries to solve the issue of direct connectivity by having an intermediary open machine assist in the manipulation of outgoing packets on either host.[8]

## 2.6 NAT-PMP

NAT Port Mapping Protocol is a protocol for automating the process of creating Network Address Translation (NAT) port mappings. Included in the protocol is a method for retrieving the external IP address of a NAT gateway, thus allowing a client to make this external IP address and port number known to peers that may wish to communicate with it.[9]

## Chapter 3

# UPnP

UPnP is a protocol for communication between controllers, or control points, and devices. It provides a distributed, open networking architecture to enable seamless proximity networking in addition to control and data transfer among networked devices [3].

The UPnP Device Architecture is designed to support zero-configuration and automatic discovery for a breadth of device categories from a wide range of vendors. Devices can dynamically join a network, obtain an IP address, convey its capabilities and learn about the presence and capabilities of other devices. Additionally, devices can leave the network without leaving any unwanted state behind.

Technologies used in UPnP networking include protocols such as TCP, UDP, HTTP, XML and SOAP. Contracts are based declaratively, expressed in XML, and transmitted via HTTP. The choice for these protocols is supported by their proven success in the industry and widespread implementation on different physical media. This is a key factor in enabling multiple-vendor interoperability.

Each UPnP-enabled may publish any number of root devices. The root devices themselves may decompose into multiple sub-devices, which in turn might consist of embedded devices, et cetera. Each device has a number of services that consist of actions and state variables.

UPnP networking consists of 5 steps: discovery, description, control, eventing, presentation. Eventing and presentation are not relevant in our case, so they are omitted.

### 3.1 Discovery

The UPnP discovery protocol allows new devices do advertise its services to control points on the network. Also, it allows new control points to search for devices on the network. The discovery protocol can use either multicast or unicast address, the former used to locate devices in a specified network scope, the latter used to find details about a known device. The protocol used for

discovery is based on Simple Service Discovery Protocol, an expired IETF draft by Microsoft and Hewlett-Packard. SSDP protocol largely resembles HTTP, but uses UDP instead of TCP at transport layer. Also, different from HTTP is the possibility of multiple responses or none at all. Due to the unreliable nature of UDP, it is recommended for the packets to be sent multiple times at random intervals and clients should wait at minimum a second for responses.

In Listing 3.1 we can see a search request for an Internet Gateway Device.

**M-SEARCH** Method for SSDP search requests

\* Request applies generally and not to a specific resource

**HOST** The IP address and port of the target. For multicast addresses the port must be 1900, for unicast a different port may be used. IPv4 multicast address is 239.255.255.250. IPv6 link-local, site-local and global addresses are respectively fe02::c, fe05::c and fe0e::c. IPv6 addresses are encoded surrounded in square brackets:

HOST: [::1]:1900

**MAN** Namespace of the HTTP extension, must be "ssdp:discover" (with quotes) for search requests.

**MX** Time to live in seconds for the search request. Devices should wait a random duration between 0 and this many seconds before sending a response

**ST** Search target. In our case it is in the format

urn:schemas-upnp-org:device:deviceType:version.

A possible response can be seen in Listing 3.2 on the next page. Receiving such a response enables the next step, description.

**CACHE-CONTROL** the max-age directive specifies the time in seconds this response can be considered valid

**EXT** required for backwards compatibility with UPnP 1.0

**LOCATION** URL to the root device description

**SERVER** Given by UPnP vendor. Contains device name and version, followed by UPnP version

**ST** The search target this response applies to

**USN** Unique Server Name, single URI

Listing 3.1: Discovery request using IPv4 multicast

```
M-SEARCH * HTTP/1.1
HOST:239.255.255.250:1900
MAN:"ssdp:discover"
MX:3
ST:urn:schemas-upnp-org:device:InternetGatewayDevice:1
```

Listing 3.2: Discovery response

```

HTTP/1.1 200 OK
CACHE-CONTROL: max-age=1800
EXT:
LOCATION: http://192.168.1.254:80/upnp/IGD.xml
SERVER: Thomson TG 784 8.2.3.A UPnP/1.0 (00-24-17-49-71-12)
ST: urn:schemas-upnp-org:device:InternetGatewayDevice:1
USN: uuid:UPnP_Thomson TG784-1_00-24-17-49-71-12::urn:
      schemas-upnp-org:device:InternetGatewayDevice:1

```

## 3.2 Description

Once we know the location of the device description file, we can retrieve it and determine its capabilities. This is performed by a simple HTTP GET request for the given URL and the response is an XML file.

In Listing 3.3 we can see a fragment of Thomson TG784 `InternetGatewayDevice:1` device descriptor [10]. From that we can see that this device supports a service named `WANIPConnection` whose description is located at `http://192.168.1.254:80/upnp/WANIPConnection.xml`.

Retrieving the service description, we can see which actions the service provides. In Listing 3.4 we can see a fragment of such description. In Table 3.1 on page 14 we can see the meaning of arguments to `AddPortMapping`, as per `WANIPConnection:1` specification [11].

Listing 3.3: Thomson TG784 `InternetGatewayDevice` root device description

```

<URLBase>http://192.168.1.254:80</URLBase>
<!-- ... -->
<service>
<serviceType>urn:schemas-upnp-org:service:WANIPConnection:1
  </serviceType>
<serviceId>urn:upnp-org:serviceId:WANIPConn1</serviceId>
<controlURL>/upnp/control/igd/wanipc_1_1_1</controlURL>
<eventSubURL>/upnp/event/igd/wanipc_1_1_1</eventSubURL>
<SCPDURL>/upnp/WANIPConnection.xml</SCPDURL>
</service>

```

Listing 3.4: Thomson TG784 `WANIPConnection` service description

```

<action>
<name>AddPortMapping</name>
<argumentList>
<argument>
  <name>NewRemoteHost</name>
  <direction>in</direction>

```

```

</argument>
<argument>
  <name>NewExternalPort</name>
  <direction>in</direction>
</argument>
<argument>
  <name>NewProtocol</name>
  <direction>in</direction>
</argument>
<argument>
  <name>NewInternalPort</name>
  <direction>in</direction>
</argument>
<argument>
  <name>NewInternalClient</name>
  <direction>in</direction>
</argument>
<argument>
  <name>NewEnabled</name>
  <direction>in</direction>
</argument>
<argument>
  <name>NewPortMappingDescription</name>
  <direction>in</direction>
</argument>
<argument>
  <name>NewLeaseDuration</name>
  <direction>in</direction>
</argument>
</argumentList>
</action>

```

### 3.3 Control

Having retrieved the service description, we can control the device by invoking actions on the device's service. The invocation is done using SOAP over HTTP. The message body contains a single element named by the action and has child-elements with the names of the input arguments and their values.

In Listing 3.5 we can see a request for **AddPortMapping** invocation. Port 10000 on the WAN interface of the gateway is forwarded to port 10000 on the LAN interface of the target machine. A success response can be seen in Listing 3.6 on the following page and a failure response in case of non-existing target machine in Listing 3.7 on page 15.

Table 3.1: AddPortMapping arguments

Name	Description
NewRemoteHost	IP address of the WAN interface on the gateway
NewExternalPort	Port on the WAN interface on the gateway
NewProtocol	Transport layer protocol. TCP or UDP
NewInternalPort	Target machine IP address
NewInternalClient	Target machine port
NewEnabled	Whether to enable this port mapping. 1 to enable, 0 to disable
NewPortMappingDescription	A description for the mapping, used only for presentation
NewLeaseDuration	Time to live, in seconds, for this port mapping. A value of 0 defines a mapping of infinite lifetime.

Listing 3.5: AddPortMapping request

```

POST /upnp/control/igd/wanipc_1_1_1 HTTP/1.1
SOAPAction: "urn:schemas-upnp-org:service:WANIPConnection
:1#AddPortMapping"
Host: 192.168.1.254

<SOAP-ENV:Envelope>
<SOAP-ENV:Body>
<m:AddPortMapping>
<NewRemoteHost>90.191.98.253</NewRemoteHost>
<NewExternalPort>10000</NewExternalPort>
<NewProtocol>TCP</NewProtocol>
<NewInternalPort>10000</NewInternalPort>
<NewInternalClient>192.168.1.70</NewInternalClient>
<NewEnabled>1</NewEnabled>
<NewPortMappingDescription>Test
  </NewPortMappingDescription>
<NewLeaseDuration>0</NewLeaseDuration>
</m:AddPortMapping>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Listing 3.6: AddPortMapping success response

```

HTTP/1.0 200 OK

<s:Envelope>
<s:Body>

```

```
<m:AddPortMappingResponse></m:AddPortMappingResponse>
</s:Body>
</s:Envelope>
```

Listing 3.7: AddPortMapping failure response

HTTP/1.0 500 Internal Server Error

```
<s:Envelope>
<s:Body>
<s:Fault>
<faultcode>s:Client</faultcode>
<faultstring>UPnPError</faultstring>
<detail>
<UPnPError>
<errorCode xmlns="">702</errorCode>
<errorDescription xmlns="">Invalid value specified
  </errorDescription>
</UPnPError>
</detail>
</s:Fault>
</s:Body>
</s:Envelope>
```

## Chapter 4

# Portforward

Our aim was to create a tool that would allow us to configure port forwarding for the gateway using UPnP. Initially, it was planned to be used by F2F Computing Framework, but due to the demand from other projects, it was decided that a standalone tool be made. Java was chosen as a platform for its portability. For ease of use, command-line scripts were made that call the java code.

### 4.1 Prototype

Since UPnP combines so many technologies, writing everything from scratch would have been infeasible. The alternatives were to combine libraries of different technologies or use a UPnP library that already did that. I found that cgupnpjava fits our needs the best.

Advantages of cgupnpjava:

- Discovery
- Description
- Control
- Eventing
- Uses industry-proven XML manipulation library
- High-level API

Disadvantages of cgupnpjava:

- Blocking I/O
- Tightly coupled code
- Some bad coding practices including level of abstraction

Once the library was chosen, a prototype was developed to test the usability of the library. In this phase, some cgupnpjava bugs, like network I/O threads never terminating, were discovered.

## 4.2 The program

The program was written in Java and uses a modified version of the cgupnpjava library. In particular, network code was rewritten to use non-blocking operations. This was needed because the old version did not cope with the case where no UPnP device responded to the discovery. Also, it did not properly keep track of the root device description URL and failed to load service descriptors on some devices.

It supports IPv4 multicast and unicast discovery. For each device found, it prints out the descriptor URL, which can be later used to bypass the discovery: LOCATION `http://10.0.0.3:8080/igd.xml`

The supported actions are:

- List all port mappings with their status, protocol, external and internal addresses
- Add a port mapping
- Remove a port mapping

## 4.3 Issues

The program was written with IPv6 support in mind, but currently, due to Java bug #6230761<sup>1</sup>, IPv6 does not work with non-blocking network I/O. Until a fix is released, IPv6 support remains unverified.

A few devices sent device descriptions that did not match the specification. This was however not a big issue, as the devices, services and actions critical to port forwarding were present.

All the devices used for testing responded with their own set of error codes that were not in the specification. For example, Thomson TG784 responded with 702 instead of 402 for invalid arguments. Also, error messages were too generic and did not hint at the point of error or cause. For example, when trying to forward port to a non-existing machine, `Invalid value specified` was sent, but there was no indication to the argument named `NewInternalHost`, nor that a machine with the given address did not exist. In our case, we only register success or failure and do not need to handle different types of errors.

---

<sup>1</sup>[http://bugs.sun.com/view\\_bug.do?bug\\_id=6230761](http://bugs.sun.com/view_bug.do?bug_id=6230761)

## 4.4 Usage

The program was implemented as a command-line tool, packaged as a Java Archive (JAR). The general usage syntax is `java -jar portforward.jar [arguments]`, although it is easier to use wrappers:

- On Windows: `portforward [arguments]`
- On Linux: `./portforward [arguments]`

Use `-h` as the single argument to get a list of supported arguments and their descriptions.

Listing 4.1: Forwarding port 10000

```
$ ./portforward -m -i 10000 -t
LOCATION http://192.168.1.254:80/upnp/IGD.xml
OK ADD 90.90.90.90:10000 192.168.1.1:10000
```

In Listing 4.1 we can see an example invocation of the program and its output. The `-m` argument tells to use IPv4 multicast discovery, `-i 10000` specifies the port and `-t` tells to forward only TCP. From the output we can see:

- A router was discovered with a UPnP device description at the URL `http://192.168.1.254:80/upnp/IGD.xml`
- The router's WAN IP address is `90.90.90.90`
- Port 10000 on the router WAN interface is now forwarded to port 10000 on our machine LAN interface having the address `192.168.1.1`

This tool can be used by other programs by invoking the program and screen-scraping the results.

# Conclusion

In this thesis we address the issue of connecting hosts behind NATs. UPnP is used to configure port mapping on the gateway in order to reach the host. We implement a command line tool specifically for this purpose. The tool uses a modified CyberLink for Java library to discover and control UPnP-enabled gateways.

We conclude that port forwarding with UPnP is relatively simple. Should a UPnP library be missing or in some way inadequate on some platform, it would be easy to write one, given that libraries for the technologies involved exist. Since XML processing and TCP/UDP networking are fairly common, the tools for those should be widely available.

Being able to configure port mappings on the router with little effort gives us the option to discard other, more expensive ways of establishing direct connections.

In the future the tool is planned to be used by F2F Computing Framework. In order to bridge the gap between native applications and Java, the tool is planned to be ported into C++ as a library. This would ease the integration with other projects, as a command line tool, with its invocation and screen-scraping, has a higher overhead.

# Pordisiire universaalse isehäällestumisega

Bakalaureusetöö (6EAP)

Gert Palok

Resümee

Seoses IPv4 aadressiruumi täitumisega tekkis vajadus võrguaadresside translatoorite järele (NAT). See tõi endaga kaasa mitmeid probleeme, üheks peamiseks masinate peitumine lüüside taha. Kui kaks lüüsi taga asuvat masinat tahavad omavahel suhelda, näeb kumbi vaid teineteise lüüsi. Kui oleks võimalik lüüse automaatselt seadistada nii, et sissetulev ühendus sisevõrku õigele masinale suunataks, siis oleks võimalikud otseühendused. UPnP aitab meil seda teha, pakkudes standardiseeritud protokoll mitmesuguste seadmetega, sealhulgas lüüsidega, suhtlemiseks.

Me realiseerisime UPnP-ga portide suunamise seadistamise tarkvara Javas cgupnpjava teeki kasutades. Samuti märgime ära mõned esinenud raskused ning erinevused spetsifikatsiooni ning implementatsioonide vahel.

UPnP kasutab UDP ja TCP võrguprotokolle ning toetub laialtlevinud tehnoloogiatele HTTP, XML ja SOAP. Kui mingil platvormil on puudu UPnP teek, siis seotud tehnoloogiate teekide olemasolul on võimalik UPnP teeki üsna lihtne implementeerida.

# Bibliography

- [1] “Nat types.” [Online]. Available: <http://list.sipfoundry.org/archive/ietf-behave/pdf00000.pdf>
- [2] U. Norbistrath, “Friend-to-friend (f2f) computing,” Jan. 2010. [Online]. Available: <http://ulno.net/f2f/>
- [3] UPnP-Forum, “Upnp device architecture version 1.1,” Oct. 2008. [Online]. Available: <http://upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.1.pdf>
- [4] “Portable sdk for upnp devices (libupnp 1.6.6),” May 2008. [Online]. Available: <http://pupnp.sourceforge.net/>
- [5] CyberGarage, “Cyberlink for java,” Mar. 2010. [Online]. Available: <http://www.cybergarage.org/cgi-bin/twiki/view/Main/CyberLinkForJava>
- [6] Super Bonbon Industries, “Upnplib,” Nov. 2006. [Online]. Available: <http://www.sbbi.net/site/upnp/index.html>
- [7] “The linux upnp internet gateway device,” Feb. 2007. [Online]. Available: <http://linux-igd.sourceforge.net/>
- [8] A. Lind, “Jpunch: Java udp hole punching library,” Apr. 2009. [Online]. Available: <http://ulno.net/projects/jpunch>
- [9] S. Cheshire, M. Krochmal, Apple Inc., K. Sekar, and Sharpcast Inc., “Nat port mapping protocol (nat-pmp),” Apr. 2008. [Online]. Available: <http://tools.ietf.org/html/draft-cheshire-nat-pmp-03>
- [10] UPnP-Forum, “Upnp internetgatewaydevice:1 version 1.01,” Nov. 2001. [Online]. Available: <http://upnp.org/specs/gw/UPnP-gw-InternetGatewayDevice-v1-Device.pdf>
- [11] —, “Upnp wanipconnection:1 version 1.01,” Nov. 2001. [Online]. Available: <http://upnp.org/specs/gw/UPnP-gw-InternetGatewayDevice-v1-Device.pdf>