

portproxy  
User's Guide

Author: Tomasz Mrugalski  
version 2009-10-22

# Table of contents

1 Project overview.....	4
1.1 Phase 1: Portproxy.....	4
1.2 Phase 2: Port forwarding GUI.....	4
2 Project status.....	4
2.1 Phase 1: portproxy.....	4
2.2 Phase 2: Port forwarding GUI.....	4
2.3 Revision history.....	4
3 Portproxy overview.....	5
4 Compilation.....	5
4.1 OpenWRT compilation.....	5
4.2 MiniUPNP daemon.....	6
4.2.1 UPnP client.....	6
4.2.2 NAT-PMP client.....	7
5 Configuration.....	7
5.1 CGN.....	7
5.1.1 softwire configuration.....	7
5.2 OpenWRT box.....	8
5.2.1 Softwire configuration.....	9
5.2.2 Running portproxy.....	9
6 Testing.....	10
6.1 NAT-PMP.....	10
6.1.1 Incoming NAT-PMP request.....	10
6.1.2 Returning NAT-PMP response.....	10
6.1.3 NAT-PMP end to end testing.....	11
6.2 UPnP.....	11
6.2.1 SSDP announcements .....	12
6.2.2 Incoming UPNP query.....	13
6.2.3 Returning UPNP response.....	14
6.3 UPnP end to end testing.....	15
6.4 All in one testing.....	19
7 Source code.....	22
7.1 Portproxy source code.....	22
7.1.1 Concepts.....	22
7.1.2 Architecture.....	22
8 Firmware installation.....	22
8.1 Connecting LinkSys.....	22
8.2 Firmware upgrade (using original web interface).....	23
8.3 Firmware upgrade (using linux console).....	24
8.4 Firmware upgrade (TFTP).....	24
8.5 Enabling boot_wait phase.....	25
8.6 First connection.....	26
8.7 ipk packages.....	26
8.8 Package installation.....	26
9 Best practices and debugging tips.....	27
10 Links.....	27
11 Contact.....	28

# 1 Project overview

This document describes usage, installation and configuration of the software required to setup and run portproxy – a software that resolves port forwarding configuration issues. Also, accompanying GUI for manual port forward configuration is the second goal of this project.

## 1.1 Phase 1: Portproxy

Portproxy is a software used to fill the communication gap between clients located in the LAN and CGN located in ISP network. The goal of this phase is to develop software that allows clients to configure port forwarding. Two protocols are used for that purpose: UPnP and NAT-PMP. Goal of this project is to forward UPnP and NAT-PMP traffic between LAN and CGN in a transparent (from the client's perspective) way. Portproxy is a command-line tool that runs on any device that uses OpenWRT as an operating system.

## 1.2 Phase 2: Port forwarding GUI

Second phase of this project focuses on manual port configuration. Clients located in LAN should be able to manually configure their home router to forward selected ports. Web-based GUI will be developed for that purpose. GUI should also include other aspects, like tunnel configuration. GUI must be easily extensible.

# 2 Project status

For latest status, list of tasks already completed, work in progress and upcoming tasks, see project web page (<http://klub.com.pl/portproxy/>). Should priorities change during code development, please contact Tomasz Mrugalski. As of 2008-09-02, both phases are fully functional.

## 2.1 Phase 1: portproxy

As of 2009-10-22, portproxy is fully functional. It is able to forward NAT-PMP traffic and handle required UPnP messages.

## 2.2 Phase 2: Port forwarding GUI

As of 2009-10-22, Port forwarding GUI is in development.

## 2.3 Revision history

Date	Change description
22.10.2009	Initial documentation release

### 3 Portproxy overview

Portproxy is a flexible command-line tool used for forwarding NAT-PMP and UPnP traffic.

Portproxy provides number of options for tuning its operation. Help is available with -h command.

Result of running portproxy -h command:

```
root@OpenWrt:/tmp# ./portproxy -h
| Portproxy - NAT-PMP/uPNP proxy, version 2009-10-22
| Author : Tomasz Mrugalski<tomasz.mrugalski@gmail.com>
| Licence : GNU GPL v2 only. Developed at Gdansk University of Technology.
| Homepage: http://klub.com.pl/portproxy/
| portproxy 2009-10-22
|-----
--lan-iface name - defines LAN interface, (br-lan) by default
--wan-iface name - defines WAN interface, (eth0.1) by default
-l a.b.c.d - defines LAN IPv4 address (e.g. -l 192.168.1.1)
              Portproxy will listen for client's messages on this address
-w a.b.c.d - defines WAN IPv4 address (e.g. -w 80.10.128.1)
              Portproxy will use this address to forward messages to the NAT router
-n a.b.c.d - defines CGN IPv4 address (e.g. -w 80.10.128.2)
              Portproxy will forward incoming messages to this IPv4 address.
--nat-pmp X - Enables(X=1) or Disables (X=0) nat-pmp protocol (default = enabled)
--upnp-udp X - enables(X=1) or disables (X=0) uPNP UDP support (default = enabled)
--upnp-tcp X - enables(X=1) or disables (X=0) uPNP TCP support (default = enabled)
-v          - increases verbosity level to 2 (more verbose than default 1)
              (use twice to further increase verbosity)

Following options are for testing/debugging purposes only:
--nat-pmp-port X - specifies UDP port for NAT-PMP protocol (default = 5351)
--upnp-udp-port X - specifies UDP port for uPNP protocol (default = 1900)
--upnp-tcp-port1 X - specifies TCP port1 for uPNP protocol (default = 2869)
--upnp-tcp-port2 X - specifies TCP port2 for uPNP protocol (default = 2857)
```

By default, portproxy supports both NAT-PMP and UPnP. Each protocol can be disabled, using `--nat-pmp 0` or `--upnp-udp 0`, respectively. To change default behavior permanently, modify `ctx_defaults()` function in `portproxy.c`.

Portproxy by default uses `br-lan` as LAN interface and `eth0.1` as WAN interface. It attempts to guess IPv4 addresses that should be used. Usually, those auto-detection routines are working properly, but it is possible to force specific addresses. Three addresses are used:

- LAN IPv4 address. LAN sockets are bound do that address. Also responses from CGN are sent from this address to clients. Specified with -l option.
- WAN IPv4 address. WAN socket is bound to this address. It is used as a source address when sending forwarded requests to CGN. Specified with -w option.
- CGN IPv4 address. Client requests are forwarded to this address. Specified with -n option.

To run portproxy, usually only one option is required: definition of WAN interface. If not specified, `eth0.1` will be used for this purpose. To specify that softwire interface should be used, use `--wan-iface softwire`.

It is also possible to increase verbosity level. By default only essential messages are printed. To increase details, -v option should be used. It can be specified twice to increase details even more.

Although not very useful in production environment, it is also possible to adjust port numbers that portproxy binds and forwards data to. Use `--nat-pmp-port 12345`, `--upnp-udp-port 12345`, `--upnp-tcp-port1 12345` or `--upnp-tcp-port2 12345` to override default port selection.

Portproxy also provides preliminary support for TCP forwarding. As this functionality is considered

ALG operation, it is disabled by default and currently not used. It is possible, however, to enable it (`--upnp-tcp 1`). Portproxy will open 2 local TCP sockets and will wait for incoming connections. Once client connects to portproxy, another connection to CGN will be established. All data sent by client will be forwarded to CGN and vice versa.

## 4 GUI Overview

**TODO:** Describe GUI

## 5 Compilation

Number of software packages are required to establish test environment. Following sections describe steps required for successful compilation. All parts of the portproxy project and its associated software are distributed as an open source. Therefore full source code is available and can be compiled.

### 5.1 OpenWRT compilation

**TODO:** Update this section

portproxy project uses development branch (“kamikaze”) of the embedded Linux distribution called OpenWRT. First step to build a firmware for your embedded device is to download OpenWRT. OpenWRT is a set of tools that automate building process of the firmware. There are several ways of obtaining sources. It is possible to download stable sources or use SVN repository instead. For the stability purposes, all development related to ip46nat is done on one specific SVN snapshot, revision 12386. (During phase 1 development, revision 11276 was used, but it didn't support 2.6.25 kernel that was required for features used in phase 2). To obtain this revision, issue following command:

```
svn co -r 12386 https://svn.openwrt.org/openwrt/trunk/
```

After checkout is complete, initial configuration takes place. To start configuration, type:

```
make menuconfig
```

Please note that, although using the same framework, that interface is significantly different from a similarly looking, Linux kernel configuration. Also, you may want to postpone this step and install additional patches as described in the following sections. Example of the OpenWRT configuration process is presented in Fig. 6.

Alternatively, sources with necessary patches are available on the ip46nat project website. You can download and extract them instead of getting the source code from the OpenWRT's SVN repository.

After configuration is complete, type `make` to download required source code, build tool chain required for cross-compilation, all target binaries and images. This may take several hours, depending on the speed of the network connectivity and CPU. After subsequent rebuilds, this process is much shorter. For extra verbosity level, `make V=99` command may be used.

Make sure that the PC has Internet connectivity as OpenWRT downloads multiple additional packages, like kernel source. After compilation is complete, all packages and firmware images are available in the `bin/` directory.

### 5.2 MiniUPNP daemon

It is open source (BSD license) software, available on <http://miniupnp.free.fr/>. There are several packages available, but for the CGN, `miniupnpd` (daemon) is required. 20090904 version was used

for that purpose:

<http://miniupnp.free.fr/files/download.php?file=miniupnpd-20090904.tar.gz>

After downloading, it may be compiled, using following command:

```
make -f Makefile.linux
```

Depending on distribution used, errors regarding missing headers `libiptc/libiptc.h`, `iptables.h` and `linux/netfilter/nf_nat.h` may be encountered. Although it is possible that those headers may be provided by distribution specific packages, the best way is to download required iptables package. It seems that miniupnpd is very sensitive to differences in iptables. This practically means that specific miniupnpd version is coupled with only one iptables version. Note that this dependency is true for compilation only. Once compiled, miniupnpd should run properly, regardless of Linux kernel or iptable version installed in system.

Used miniupnpd version used was 20090904. It requires iptables-1.4.1:

<http://netfilter.org/projects/iptables/files/iptables-1.4.1.tar.bz2>

After extracting, its compilation is simple:

```
./configure  
make
```

Once the compilation is complete, it is now possible to compile miniupnpd. Path to iptables-1.4.1 is specified in the following way:

```
IPTABLESPATH=/home/thomson/devel/iptables-1.4.1 make -f Makefile.linux
```

Optionally, following line may be added to Makefile.linux to ease consecutive compilations:

```
IPTABLESPATH=/home/thomson/devel/iptables-1.4.1
```

After compilation is complete, miniupnpd binary is ready to use.

### 5.2.1 UPnP client

MiniUPnP client that supports port forwarding using UPnP protocol. Version 20090921 was used during testing:

<http://miniupnp.free.fr/files/download.php?file=miniupnpc-20090921.tar.gz>

Its compilation is very simple:

```
make
```

There are 2 binaries: `upnpc-shared` and `upnpc-static`. Share version requires installation of `libminiupnpc.so`, so it is more convenient to use static version.

### 5.2.2 NAT-PMP client

NAT-PMP support was added recently to miniupnpd. There is a separate subproject dedicated to that purpose. Simple NAT-PMP client is included. `Libnatpmp-20090713` was used:

<http://miniupnp.free.fr/files/download.php?file=libnatpmp-20090713.tar.gz>

After extracting the archive, compilation is simple:

```
make
```

Once again, static and shared clients will be compiled. It is easier to use static version: `natpmpc-static`.

## 6 Configuration

This section describes configuration required to run portproxy.

## 6.1 CGN

Carrier Grade NAT is located in the ISP network. It acts as a softwire termination point, performed NAT on decapsulated packets and also provides port forwarding services: NAT-PMP and uPNP. To simulate such device, ordinary PC running Linux may be used. Following configuration steps are required.

### 6.1.1 softwire configuration

It is assumed that softwire is configured over 2000::/64 prefix (CGN uses address 2000::36/64 and LinkSys box uses 2000::1). Following steps will configure softwire on CGN:

```
ip a a 2000::36/64 dev eth0
modprobe ip6_tunnel
ip -6 t a softwire mode ipip6 local 2000::36 remote 2000::1 00
ip a a 10.0.0.36/24 dev softwire
```

10.0.0.36 address is configured on softwire for NAT-PMP/uPNP service only.

NAT-PMP/uPNP daemon

For the testing purposes, MiniUPnP daemon was used.

UPnP/NAT-PMP daemon

After compilation, miniupnpd must be configured. Example configuration is provided with the sources. Here is the minimal configuration used during tests. Extra parameters and more configuration options are available in example \*.conf files distributed with source code:

```
# cfg-softwire.conf file
# WAN network interface
ext_ifname=eth1
# if the WAN interface has several IP addresses, you
# can specify the one to use below
ext_ip=1.2.3.4

# LAN network interfaces IPs / networks
listening_ip=10.0.0.36/24
# port for HTTP (descriptions and SOAP) traffic. set 0 for autoselect.
port=0

# enable NAT-PMP support (default is no)
enable_natpmp=yes

# enable UPnP support (default is yes)
enable_upnp=yes

# bitrates reported by daemon in bits per second
bitrate_up=1000000
bitrate_down=10000000

# "secure" mode : when enabled, UPnP client are allowed to add mappings only
# to their IP.
#secure_mode=yes
secure_mode=no

# report system uptime instead of daemon uptime
system_uptime=yes

# notify interval in seconds. default is 30 seconds.
notify_interval=60

# unused rules cleaning.
```

```

clean_ruleset_interval=600

# uuid : generate your own with "make genuuid"
uuid=fc4ec57e-b051-11db-88f8-0060085db3f6

# serial and model number the daemon will report to clients
# in its XML description
serial=12345678
model_number=1

# UPnP permission rules
allow 1024-65535 10.0.0.0/24 1024-65535
allow 1024-65535 192.168.1.0/24 1024-65535

```

eth1 and 1.2.3.4 are external interface and its address. This address will be reported to clients.

To start miniupnpd, use following command:

```
./miniupnpd -d -f cfg-software.conf
```

Debugging mode (-d) instructs miniupnpd that it should not work as a daemon, but rather print messages on the console. Successful start will look like this:

```

root@billabong:/home/thomson/devel/miniupnpd-20090904# ./miniupnpd -d -f
cfg-software.conf
miniupnpd[6697]: system uptime is 85640 seconds
miniupnpd[6697]: HTTP listening on port 40381
miniupnpd[6697]: Listening for NAT-PMP traffic on port 5351

```

Note that required SSDP announcements (M-NOTIFY) will be transmitted. Miniupnpd is now ready to receive queries.

## 6.2 OpenWRT box

OpenWRT box (LinkSys or any other OpenWRT capable device) requires several features to operate properly.

### 6.2.1 Software configuration

After appropriate firmware image is flashed, it is possible to use ssh to connect to the box. Following commands may be used to establish software. It is assumed that LAN interface (Ethernet sockets labelled 1-4 on the WRT54GL box) is br-lan and WAN interface (labelled "Internet" on the WRT54GL box) is eth0.1.

```

ip a a 2000::100/64 dev eth0 .1
modprobe ip6_tunnel
ip -6 t a software mode ipip6 local 2000::36 remote 2000::1 00
ip a a 10.0.0.100/24 dev software

```

### 6.2.2 Running portproxy

Portproxy is part of the firmware provided. No extra package installations are necessary. For details regarding portproxy, see Section 4. Typical use requires only specification of the WAN interface. In this example, it is software. Also, it may be convenient to add -v to increase verbosity (-v may be used twice to enable debugging messages). Following command starts portproxy:

```
root@OpenWrt:/tmp# portproxy -v --wan-iface software
```

Following output will be produced. Portproxy is now able to handle NAT-PMP and UPnP traffic.

```

root@OpenWrt:/tmp# ./portproxy -v --wan-iface softwire
| Portproxy - NAT-PMP/uPNP proxy, version 2009-10-22
| Author : Tomasz Mrugalski<tomasz.mrugalski(at)gmail.com>
| Licence : GNU GPL v2 only. Developed at Gdansk University of Technology.
| Homepage: http://klub.com.pl/portproxy/
Detected interfaces:
Interface lo, 1 address(es), MAC=00:00:00:00:00:00
  IPv4=127.0.0.1
Interface eth0, 0 address(es), MAC=00:1d:7e:bc:41:2a
Interface eth0.0, 0 address(es), MAC=00:1d:7e:bc:41:2a
Interface eth0.1, 0 address(es), MAC=00:1d:7e:bc:41:2a
Interface br-lan, 1 address(es), MAC=00:1d:7e:bc:41:2a
  IPv4=192.168.1.1
Interface ip6tnl0, 0 address(es), MAC=00:00:00:00:00:00
Interface softwire, 1 address(es), MAC=20:00:00:00:00:00
  IPv4=10.0.0.100
Configuration:
lan=br-lan
wan=softwire
lanIP=192.168.1.1
wanIP=10.0.0.100
routerIP=10.0.0.36
natpmp_udp_port=5351
upnp_udp_port=1900
upnp_tcp_port1=2869
upnp_tcp_port2=2857
fwd_tcp_port1=2869
fwd_tcp_port2=2857
shared_udp_sock=YES
enable_natpmp=YES
enable_upnp_tcp=NO
enable_upnp_udp=YES
Enabling uPNP (UDP)
Creating UDP socket: addr=0.0.0.0, port=1900 on interface br-lan, result=4
UDP(SSDP) socket created: sockid=4, port=1900
Successfully joint multicast group 239.255.255.250 (local IP=192.168.1.1)
Creating UDP socket: addr=0.0.0.0, port=1900 on interface softwire, result=5
UDP(SSDP) socket created: sockid=5, port=1900
Successfully joint multicast group 239.255.255.250 (local IP=10.0.0.100)
Sockets 4(LAN:br-lan) and 5(WAN:softwire) paired.
Enabling NAT-PMP
Creating UDP socket: addr=192.168.1.1, port=5351 on interface br-lan, result=7
UDP: sockid=7, port=5351
Opening single UDP transmission socket (random port)
Creating UDP socket: addr=10.0.0.100, port=0 on interface softwire, result=8
UDP: sockid=8, port=33467

Waiting for incoming packets (timeout: 3600s, 0us), 0 mappings (0 pkts handled)

```

## 7 Testing

After test environment is complete (see Section 6), tests can be started. Following sections demonstrate operational status of the portproxy.

### 7.1 NAT-PMP

NAT-PMP is a UDP based protocol, defined in <http://tools.ietf.org/html/draft-cheshire-nat-pmp-03>. It is strictly local, i.e. communication never spans more than one network and clients send queries to their default gateway. UDP messages is sent to UDP port 5351. Each query is responded by the NAT-PMP capable with a UDP response.

#### 7.1.1 Incoming NAT-PMP request

To initiate client NAT-PMP exchange, run following command:

```
./natpmpc-static
```

Client sends NAT-PMP request. It is received by portproxy, running on OpenWRT box. Incoming packet is received on LAN interface. It is being forwarded over softwire and mapping for expected response is added.

```

Waiting for incoming packets (timeout: 3600s, 0us), 0 mappings (8 pkts
handled)
Received UDP packet: 2 bytes (sock-id=7): 192.168.1.2/port 48184 ->
192.168.1.1/port 5351
Mapping UDP: src=192.168.1.2/port 48184,dst=192.168.1.1/port 5351 =>
src=10.0.0.100/port 47877,dst=10.0.0.36/port 5351 (sock=7/8)
Forwarding UDP query (2 bytes) to 10.0.0.36/port 5351

```

No.	Time	Source	Destination	Protocol	Info
9	10.010915	192.168.1.1	239.255.255.250	SSDP	NOTIFY * HTTP/1.1
10	10.012415	192.168.1.1	239.255.255.250	SSDP	NOTIFY * HTTP/1.1
11	30.101619	192.168.1.2	192.168.1.1	UDP	Source port: 48184 Destination port: nat-pmp
12	30.109979	192.168.1.1	192.168.1.2	UDP	Source port: nat-pmp Destination port: 48184
13	35.097955	Ibm_12:66:88	Cisco-Li_bc:41:2a	ARP	who has 192.168.1.1? Tell 192.168.1.2
14	35.098329	Cisco-Li_bc:41:2a	Ibm_12:66:88	ARP	192.168.1.1 is at 00:1d:7e:bc:41:2a

```

> Frame 11 (44 bytes on wire, 44 bytes captured)
> Ethernet II, Src: Ibm_12:66:88 (00:11:25:12:66:88), Dst: Cisco-Li_bc:41:2a (00:1d:7e:bc:41:2a)
> Internet Protocol, Src: 192.168.1.2 (192.168.1.2), Dst: 192.168.1.1 (192.168.1.1)
> User Datagram Protocol, Src Port: 48184 (48184), Dst Port: nat-pmp (5351)
> Data (2 bytes)

```

Fig. 1: NAT-PMP request (as seen over LAN)

No.	Time	Source	Destination	Protocol	Info
39	30.110367	10.0.0.100	10.0.0.36	UDP	Source port: 47877 Destination port: nat-pmp
40	30.110616	10.0.0.36	10.0.0.100	UDP	Source port: nat-pmp Destination port: 47877

```

> Frame 39 (84 bytes on wire, 84 bytes captured)
> Ethernet II, Src: Cisco-Li_bc:41:2a (00:1d:7e:bc:41:2a), Dst: AsustekC_9b:73:49 (00:1e:8c:9b:73:49)
> Internet Protocol Version 6
> Internet Protocol, Src: 10.0.0.100 (10.0.0.100), Dst: 10.0.0.36 (10.0.0.36)
> User Datagram Protocol, Src Port: 47877 (47877), Dst Port: nat-pmp (5351)
> Data (2 bytes)

```

Fig. 2: NAT-PMP request (as seen over software)

## 7.1.2 Returning NAT-PMP response

Forwarded packet causes NAT-PMP daemon, running on CGN, to generate response. As the original request was sent from OpenWRT box, response is also transmitted to OpenWRT box. Returning NAT-PMP response is received over software. Appropriate mapping is found and final destination address can be decided. Packet is then forwarded to LAN interface. As mapping is no longer required, it is deleted. Statistics are increased.

```

Waiting for incoming packets (timeout: 10s, 0us), 1 mappings (9 pkts
handled)
Received UDP packet: 12 bytes (sock-id=8): 10.0.0.36/port 5351 ->
10.0.0.100/port 47877
Handling response using following mapping:
Mapping UDP: src=192.168.1.2/port 48184,dst=192.168.1.1/port 5351 =>
src=10.0.0.100/port 47877,dst=10.0.0.36/port 5351 (sock=7/8)
Forwarding UDP response (12 bytes) to 10.0.0.36/port 48184

```

No. .	Time	Source	Destination	Protocol	Info
39	30.110367	10.0.0.100	10.0.0.36	UDP	Source port: 47877 Destination port: nat-pmp
40	30.110616	10.0.0.36	10.0.0.100	UDP	Source port: nat-pmp Destination port: 47877

```

> Frame 40 (94 bytes on wire, 94 bytes captured)
> Ethernet II, Src: AsustekC_9b:73:49 (00:1e:8c:9b:73:49), Dst: Cisco-Li_bc:41:2a (00:1d:7e:bc:41:2a)
> Internet Protocol Version 6
> Internet Protocol, Src: 10.0.0.36 (10.0.0.36), Dst: 10.0.0.100 (10.0.0.100)
> User Datagram Protocol, Src Port: nat-pmp (5351), Dst Port: 47877 (47877)
> Data (12 bytes)

```

Fig. 1: NAT-PMP response (seen over softwire)

No. .	Time	Source	Destination	Protocol	Info
9	10.010915	192.168.1.1	239.255.255.250	SSDP	NOTIFY * HTTP/1.1
10	10.012415	192.168.1.1	239.255.255.250	SSDP	NOTIFY * HTTP/1.1
11	30.101619	192.168.1.2	192.168.1.1	UDP	Source port: 48184 Destination port: nat-pmp
12	30.109979	192.168.1.1	192.168.1.2	UDP	Source port: nat-pmp Destination port: 48184
13	35.097955	Ibm_12:66:88	Cisco-Li_bc:41:2a	ARP	who has 192.168.1.1? Tell 192.168.1.2
14	35.098329	Cisco-Li_bc:41:2a	Ibm_12:66:88	ARP	192.168.1.1 is at 00:1d:7e:bc:41:2a

```

> Frame 12 (60 bytes on wire, 60 bytes captured)
> Ethernet II, Src: Cisco-Li_bc:41:2a (00:1d:7e:bc:41:2a), Dst: Ibm_12:66:88 (00:11:25:12:66:88)
> Internet Protocol, Src: 192.168.1.1 (192.168.1.1), Dst: 192.168.1.2 (192.168.1.2)
> User Datagram Protocol, Src Port: nat-pmp (5351), Dst Port: 48184 (48184)
> Data (12 bytes)

```

Fig. 2: NAT-PMP response (seen over LAN)

### 7.1.3 NAT-PMP end to end testing

As NAT-PMP messages are forwarded in both directions, client eventually receives response from CGN:

```

thomson@ecwm0tmrugals:~/devel/libnatpmp-20090713$ ./natpmpc-static
initnatpmp() returned 0 (SUCCESS)
sendpublicaddressrequest returned 2 (SUCCESS)
readnatpmpresponseorretry returned 0 (OK)
Public IP address : 1.2.3.4
epoch = 76257
closenatpmp() returned 0 (SUCCESS)

```

## 7.2 UPnP

Universal Plug-and-Play is protocol defined by UPnP Forum. Specification is available at <http://upnp.org/standardizeddcps/default.asp>. There are number of defined devices that this protocol supports, like media servers, printing devices or Internet Gateway Devices (IGD). There are several communication modes that are defined by the standard. First is a SSDP protocol. It uses multicast address 239.255.255.250 to discover IGD. Also IGD periodically announces its presence sending advertisements to this address. To control available UPnP capable devices (e.g. IGDs), UPnP defines UPnP control points. They may have been called clients. IGD announces location of its resource description, defined as a URL (e.g. Location: <http://10.0.0.36:37989/rootDesc.xml>). Such location URL is sent in announcements and in responses to clients. Once client receives such URL, it establishes HTTP connection to specified port. This connection is used for various operations. Getting external address, setting port forwarding and getting already configured port forwarding rules are most notable examples.

### 7.2.1 SSDP announcements

Each IGD is expected to periodically (and during initialization phase) announce its presence. Such SSDP packets are sent to multicast address 239.255.255.250. As IGD is located on CGN box, those

announcement are received over softwire. They must be forwarded to LAN interface, so clients will be able to discover available IGD. Responses are not expected, so mapping is not required.

```
Waiting for incoming packets (timeout: 3600s, 0us), 0 mappings (6 pkts handled)
Received UDP packet: 333 bytes (sock-id=5): 10.0.0.36/port 51608 -> 239.255.255.250/port 1900
Mcast pkt received on WAN interface (pkt->sock=5)
Forwarding SSDP multicast to LAN
```

No. .	Time	Source	Destination	Protocol	Info
1	0.000000	10.0.0.100	224.0.0.22	IGMP	V3 Membership Report / Join g
2	5.543794	10.0.0.100	224.0.0.22	IGMP	V3 Membership Report / Join g
3	9.985687	10.0.0.36	239.255.255.250	SSDP	NOTIFY * HTTP/1.1
4	9.985718	10.0.0.36	239.255.255.250	SSDP	NOTIFY * HTTP/1.1
5	9.985742	10.0.0.36	239.255.255.250	SSDP	NOTIFY * HTTP/1.1
6	9.985750	10.0.0.36	239.255.255.250	SSDP	NOTIFY * HTTP/1.1

▶ Frame 3 (351 bytes on wire, 351 bytes captured)

- ▶ Ethernet II, Src: AsustekC\_9b:73:49 (00:1e:8c:9b:73:49), Dst: Cisco-Li\_bc:41:2a (00:1d:7e:bc:41:2a)
- ▶ Internet Protocol Version 6
- ▶ Internet Protocol, Src: 10.0.0.36 (10.0.0.36), Dst: 239.255.255.250 (239.255.255.250)
- ▶ User Datagram Protocol, Src Port: 51608 (51608), Dst Port: ssdp (1900)
- ▼ Hypertext Transfer Protocol
  - ▶ NOTIFY \* HTTP/1.1\r\n
  - HOST:239.255.255.250:1900\r\n
  - Cache-Control:max-age=120\r\n
  - Location:http://10.0.0.36:37989/rootDesc.xml\r\n
  - Server: Ubuntu/intrepid UPnP/1.0 MiniUPnPd/1.3\r\n
  - NT:upnp:rootdevice\r\n
  - USN:uuid:fc4ec57e-b051-11db-88f8-0060085db3f6::upnp:rootdevice\r\n
  - NTS:ssdp:alive\r\n
  - \r\n

Fig. 3: SSDP NOTIFY message (as seen over softwire)

No. .	Time	Source	Destination	Protocol	Info
1	0.000000	192.168.1.1	224.0.0.22	IGMP	V3 Membership Report / Join group
2	2.035565	192.168.1.1	224.0.0.22	IGMP	V3 Membership Report / Join group
3	9.993438	192.168.1.1	239.255.255.250	SSDP	NOTIFY * HTTP/1.1
4	10.002422	192.168.1.1	239.255.255.250	SSDP	NOTIFY * HTTP/1.1
5	10.005418	192.168.1.1	239.255.255.250	SSDP	NOTIFY * HTTP/1.1
6	10.006917	192.168.1.1	239.255.255.250	SSDP	NOTIFY * HTTP/1.1

▶ Frame 3 (311 bytes on wire, 311 bytes captured)

- ▶ Ethernet II, Src: Cisco-Li\_bc:41:2a (00:1d:7e:bc:41:2a), Dst: IPv4mcast\_7f:ff:fa (01:00:5e:7f:ff:fa)
- ▶ Internet Protocol, Src: 192.168.1.1 (192.168.1.1), Dst: 239.255.255.250 (239.255.255.250)
- ▶ User Datagram Protocol, Src Port: ssdp (1900), Dst Port: ssdp (1900)
- ▼ Hypertext Transfer Protocol
  - ▶ NOTIFY \* HTTP/1.1\r\n
  - HOST:239.255.255.250:1900\r\n
  - Cache-Control:max-age=120\r\n
  - Location:http://10.0.0.36:37989/rootDesc.xml\r\n
  - Server: Ubuntu/intrepid UPnP/1.0 MiniUPnPd/1.3\r\n
  - NT:upnp:rootdevice\r\n
  - USN:uuid:fc4ec57e-b051-11db-88f8-0060085db3f6::upnp:rootdevice\r\n
  - NTS:ssdp:alive\r\n
  - \r\n

Fig. 4: SSDP NOTIFY message (as seen over LAN)

## 7.2.2 Incoming UPNP query

Clients may also actively try to discover available IGDs. To do so, they transmit M-SEARCH UDP packet to multicast (239.255.255.250) address. Portproxy receives such packets on LAN interface and forwards it over softwire. As response from IGD is expected, appropriate mapping is added.

```
Waiting for incoming packets (timeout: 3600s, 0us), 0 mappings (10 pkts handled)
Received UDP packet: 137 bytes (sock-id=4): 192.168.1.2/port 35602 -> 239.255.255.250/port 1900
Mapping UDP: src=192.168.1.2/port 35602,dst=239.255.255.250/port 1900 => src=10.0.0.100/port 47877,dst=239.255.255.250/port 1900 (sock=4/8)
Forwarding UDP query (137 bytes) to 239.255.255.250/port 1900
```

No. .	Time	Source	Destination	Protocol	Info
12	30.109979	192.168.1.1	192.168.1.2	UDP	Source port: nat-pmp Dest
13	35.097955	Ibm_12:66:88	Cisco-Li_bc:41:2a	ARP	Who has 192.168.1.1? Tell
14	35.098329	Cisco-Li_bc:41:2a	Ibm_12:66:88	ARP	192.168.1.1 is at 00:1d:7e
15	45.519617	192.168.1.2	239.255.255.250	SSDP	M-SEARCH * HTTP/1.1
16	45.526765	192.168.1.1	192.168.1.2	SSDP	HTTP/1.1 200 OK
17	45.528013	192.168.1.1	239.255.255.250	SSDP	M-SEARCH * HTTP/1.1
18	47.522706	192.168.1.2	10.0.0.36	TCP	40652 -> 27000 [SYN] Seq=0

▶ Frame 15 (179 bytes on wire, 179 bytes captured)

▶ Ethernet II, Src: Ibm\_12:66:88 (00:11:25:12:66:88), Dst: IPv4mcast\_7f:ff:fa (01:00:5e:7f:ff:fa)

▶ Internet Protocol, Src: 192.168.1.2 (192.168.1.2), Dst: 239.255.255.250 (239.255.255.250)

▶ User Datagram Protocol, Src Port: 35602 (35602), Dst Port: ssdp (1900)

▼ Hypertext Transfer Protocol

▶ M-SEARCH \* HTTP/1.1\r\n

HOST: 239.255.255.250:1900\r\n

ST: urn:schemas-upnp-org:device:InternetGatewayDevice:1\r\n

MAN: "ssdp:discover"\r\n

MX: 2\r\n

\r\n

Fig. 5: UPnP/SSDP M-SEARCH request (over LAN)

No. .	Time	Source	Destination	Protocol	Info
45	45.528075	10.0.0.100	239.255.255.250	SSDP	M-SEARCH * HTTP/1.1
46	45.528315	10.0.0.36	10.0.0.100	SSDP	HTTP/1.1 200 OK

▶ Frame 45 (219 bytes on wire, 219 bytes captured)

▶ Ethernet II, Src: Cisco-Li\_bc:41:2a (00:1d:7e:bc:41:2a), Dst: AsustekC\_9b:73:49 (00:1e:8c:9b:73:49)

▶ Internet Protocol Version 6

▶ Internet Protocol, Src: 10.0.0.100 (10.0.0.100), Dst: 239.255.255.250 (239.255.255.250)

▶ User Datagram Protocol, Src Port: 47877 (47877), Dst Port: ssdp (1900)

▼ Hypertext Transfer Protocol

▶ M-SEARCH \* HTTP/1.1\r\n

HOST: 239.255.255.250:1900\r\n

ST: urn:schemas-upnp-org:device:InternetGatewayDevice:1\r\n

MAN: "ssdp:discover"\r\n

MX: 2\r\n

\r\n

Fig. 6: SSDP M-SEARCH message (as seen over softwire)

### 7.2.3 Returning UPnP response

After IGD (located at CGN) receives M-SEARCH packet, it sends standard HTTP 200 OK response. This packet is received by portproxy, corresponding mapping is found and packet is finally sent do original client.

```
Waiting for incoming packets (timeout: 10s, 0us), 1 mappings (11 pkts handled)
Received UDP packet: 306 bytes (sock-id=8): 10.0.0.36/port 1900 -> 10.0.0.100/port 47877
Handling response using following mapping:
Mapping UDP: src=192.168.1.2/port 35602,dst=239.255.255.250/port 1900 => src=10.0.0.100/port 47877,dst=239.255.255.250/port 1900 (sock=4/8)
Forwarding UDP response (306 bytes) to 239.255.255.250/port 35602
```

This message contains URL with exact IGD resources location, so from now on, client is able to contact remote IGD without any further assistance from portproxy.

No. .	Time	Source	Destination	Protocol	Info
39	30.110367	10.0.0.100	10.0.0.36	UDP	Source port: 47877 Destination port: nat-pmp
40	30.110616	10.0.0.36	10.0.0.100	UDP	Source port: nat-pmp Destination port: 47877

```

> Frame 40 (94 bytes on wire, 94 bytes captured)
> Ethernet II, Src: AsustekC_9b:73:49 (00:1e:8c:9b:73:49), Dst: Cisco-Li_bc:41:2a (00:1d:7e:bc:41:2a)
> Internet Protocol Version 6
> Internet Protocol, Src: 10.0.0.36 (10.0.0.36), Dst: 10.0.0.100 (10.0.0.100)
> User Datagram Protocol, Src Port: nat-pmp (5351), Dst Port: 47877 (47877)
> Data (12 bytes)
```

Fig. 7: UPnP response (over softwire)

No. .	Time	Source	Destination	Protocol	Info
9	10.010915	192.168.1.1	239.255.255.250	SSDP	NOTIFY * HTTP/1.1
10	10.012415	192.168.1.1	239.255.255.250	SSDP	NOTIFY * HTTP/1.1
11	30.101619	192.168.1.2	192.168.1.1	UDP	Source port: 48184 Destination port: nat-pmp
12	30.109979	192.168.1.1	192.168.1.2	UDP	Source port: nat-pmp Destination port: 48184
13	35.097955	Ibm_12:66:88	Cisco-Li_bc:41:2a	ARP	Who has 192.168.1.1? Tell 192.168.1.2
14	35.098329	Cisco-Li_bc:41:2a	Ibm_12:66:88	ARP	192.168.1.1 is at 00:1d:7e:bc:41:2a

```

> Frame 12 (60 bytes on wire, 60 bytes captured)
> Ethernet II, Src: Cisco-Li_bc:41:2a (00:1d:7e:bc:41:2a), Dst: Ibm_12:66:88 (00:11:25:12:66:88)
> Internet Protocol, Src: 192.168.1.1 (192.168.1.1), Dst: 192.168.1.2 (192.168.1.2)
> User Datagram Protocol, Src Port: nat-pmp (5351), Dst Port: 48184 (48184)
> Data (12 bytes)
```

Fig. 8: UPnP response (over LAN)

## 7.3 UPnP end to end testing

Client requests forwarding external port 9929 to local IP 10.0.0.7/port 8288. Following command sends this request:

```
./upnpc-static -a 10.0.0.7 8288 9929 tcp
```

After successful communication thru portproxy, client completes UPnP port forwarding.

**TODO:** describe operations after discovery process is complete.

No.	Time	Source	Destination	Protocol	Info
51	47.549586	10.0.0.36	192.168.1.2	TCP	37989 > 49656 [SYN, ACK] S
52	47.549618	192.168.1.2	10.0.0.36	TCP	49656 > 37989 [ACK] Seq=1
53	47.549673	192.168.1.2	10.0.0.36	HTTP/XML	POST /ctl/IPConn HTTP/1.1

Frame 53 (945 bytes on wire, 945 bytes captured)  
 Ethernet II, Src: Ibm\_12:66:88 (00:11:25:12:66:88), Dst: Cisco-Li\_bc:41:2a (00:1d:7e:bc:41:2a)  
 Internet Protocol, Src: 192.168.1.2 (192.168.1.2), Dst: 10.0.0.36 (10.0.0.36)  
 Transmission Control Protocol, Src Port: 49656 (49656), Dst Port: 37989 (37989), Seq: 1, Ack: 1,  
 Hypertext Transfer Protocol  
 eXtensible Markup Language  
 <?xml  
 <s:Envelope  
   xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"  
   s:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"  
 <s:Body>  
   <u:AddPortMapping  
     xmlns:u="urn:schemas-upnp-org:service:WANIPConnection:1">  
     <NewRemoteHost>  
       </NewRemoteHost>  
     <NewExternalPort>  
       9329  
     </NewExternalPort>  
     <NewProtocol>  
       TCP  
     </NewProtocol>  
     <NewInternalPort>  
       8248  
     </NewInternalPort>  
   </u:AddPortMapping>  
 </s:Body>  
 </s:Envelope>

|      |   |                     |
|------|---|---------------------|
| 0000 | 00 1d 7e bc 41 2a 00 11 25 12 66 88 08 00 45 00 | ..~.A*.. %.f...E.   |
| 0010 | 03 a3 d2 e2 40 00 40 06 98 a4 c0 a8 01 02 0a 00 | ....@.@. ....       |
| 0020 | 00 24 c1 f8 94 65 13 d7 2b 45 53 26 72 1c 80 18 | .\$...e... +ES&r... |
| 0030 | 00 5c cf 63 00 00 01 01 08 0a 00 df f9 a4 01 21 | .\.c.... !          |
| 0040 | d2 0e 50 4f 53 54 20 2f 63 74 6c 2f 49 50 43 6f | ..POST / ctl/IPCo   |
| 0050 | 6e 6e 20 48 54 54 50 2f 31 2e 31 0d 0a 48 6f 73 | nn HTTP/ 1.1..Hos   |
| 0060 | 74 3a 20 31 30 2e 30 2e 30 2e 33 36 3a 33 37 39 | t: 10.0. 0.36:379   |
| 0070 | 38 39 0d 0a 55 73 65 72 2d 41 67 65 6e 74 3a 20 | 89..User -Agent:    |
| 0080 | 55 62 75 6e 74 75 2f 39 2e 30 34 2c 20 55 50 6e | Ubuntu/9 .04, UPn   |
| 0090 | 50 2f 31 2e 30 2c 20 4d 69 6e 69 55 50 6e 50 63 | P/1.0, M iniUPnPc   |
| 00a0 | 2f 31 2e 33 0d 0a 43 6f 6e 74 65 6e 74 2d 4c 65 | /1.3..Co ntent-Le   |
| 00b0 | 6e 67 74 68 3a 20 35 39 33 0d 0a 43 6f 6e 74 65 | ngth: 59 3..Conte   |

Fig. 9: Port Forward request (seen on LAN)



| No. . | Time      | Source    | Destination | Protocol | Info                         |
|-------|-----------|-----------|-------------|----------|------------------------------|
| 109   | 47.853390 | 10.0.0.36 | 192.168.1.2 | TCP      | 37989 > 49656 [ACK] Seq=1 Ac |
| 110   | 47.853842 | 10.0.0.36 | 192.168.1.2 | HTTP/XML | HTTP/1.1 200 OK              |
| 111   | 47.853851 | 10.0.0.36 | 192.168.1.2 | TCP      | 37989 > 49656 [FIN, ACK] Seq |

```

> Frame 110 (500 bytes on wire, 500 bytes captured)
> Ethernet II, Src: AsustekC_9b:73:49 (00:1e:8c:9b:73:49), Dst: Cisco-Li_bc:41:2a (00:1d:7e:bc:41:2a)
> Internet Protocol Version 6
> Internet Protocol, Src: 10.0.0.36 (10.0.0.36), Dst: 192.168.1.2 (192.168.1.2)
> Transmission Control Protocol, Src Port: 37989 (37989), Dst Port: 49656 (49656), Seq: 1, Ack: 880,
> Hypertext Transfer Protocol
< eXtensible Markup Language
  < ?xml
  < s:Envelope
    xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
    s:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  < s:Body>
    < u:AddPortMappingResponse
      xmlns:u="urn:schemas-upnp-org:service:WANIPConnection:1"/>
    < /s:Body>
  < /s:Envelope>

```

Fig. 11: Port Forward response (over software)

| No. . | Time      | Source      | Destination | Protocol | Info                          |
|-------|-----------|-------------|-------------|----------|-------------------------------|
| 53    | 47.549673 | 192.168.1.2 | 10.0.0.36   | HTTP/XML | POST /ctl/IPConn HTTP/1.1     |
| 54    | 47.551085 | 10.0.0.36   | 192.168.1.2 | TCP      | 37989 > 49656 [ACK] Seq=1 Ack |
| 55    | 47.551581 | 10.0.0.36   | 192.168.1.2 | HTTP/XML | HTTP/1.1 200 OK               |

```

> Frame 55 (460 bytes on wire, 460 bytes captured)
> Ethernet II, Src: Cisco-Li_bc:41:2a (00:1d:7e:bc:41:2a), Dst: Ibm_12:66:88 (00:11:25:12:66:88)
> Internet Protocol, Src: 10.0.0.36 (10.0.0.36), Dst: 192.168.1.2 (192.168.1.2)
> Transmission Control Protocol, Src Port: 37989 (37989), Dst Port: 49656 (49656), Seq: 1, Ack: 880,
> Hypertext Transfer Protocol
< eXtensible Markup Language
  < ?xml
  < s:Envelope
    xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
    s:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  < s:Body>
    < u:AddPortMappingResponse
      xmlns:u="urn:schemas-upnp-org:service:WANIPConnection:1"/>
    < /s:Body>
  < /s:Envelope>

```

Fig. 12: Port Forward response (over LAN)

**TODO:** Describe following log file.

```

thomson@client:~/miniupnpc-20090921$ ./upnpc-static -a 10.0.0.7 8288 9929 tcp
upnpc : miniupnpc library test client. (c) 2006-2009 Thomas Bernard
Go to http://miniupnp.free.fr/ or http://miniupnp.tuxfamily.org/
for more information.
List of UPNP devices found on the network :
desc: http://10.0.0.36:52164/rootDesc.xml
st: urn:schemas-upnp-org:device:InternetGatewayDevice:1

```

```

Found valid IGD : http://10.0.0.36:52164/ctl/IPConn
Local LAN ip address : 192.168.1.2
ExternalIPAddress = 1.2.3.4
InternalIP:Port = 10.0.0.7:8288
external 1.2.3.4:9929 TCP is redirected to internal 10.0.0.7:8288
thomson@ecwm0tmrugals:~/devel/miniupnpc-20090921$ ./upnpc-static -a
192.168.1.1 8248 9329 tcp
upnpc : miniupnpc library test client. (c) 2006-2009 Thomas Bernard
Go to http://miniupnp.free.fr/ or http://miniupnp.tuxfamily.org/
for more information.
List of UPNP devices found on the network :
 desc: http://10.0.0.36:37989/rootDesc.xml
  st: urn:schemas-upnp-org:device:InternetGatewayDevice:1

Found valid IGD : http://10.0.0.36:37989/ctl/IPConn
Local LAN ip address : 192.168.1.2
ExternalIPAddress = 1.2.3.4
InternalIP:Port = 192.168.1.1:8248
external 1.2.3.4:9329 TCP is redirected to internal 192.168.1.1:8248
thomson@ecwm0tmrugals:~/devel/miniupnpc-20090921$ ./upnpc-static -a
192.168.1.1 8248 9329 tcp

```

## 7.4 All in one testing

Following section describes all in one testing, including NAT-PMP and UPnP running together.

**TODO:** describe following log.

```

root@OpenWrt:/tmp# ./portproxy -v --wan-iface softwire
| Portproxy - NAT-PMP/uPNP proxy, version 2009-10-22
| Author : Tomasz Mrugalski<thomson(at)klub.com.pl>
| Licence : GNU GPL v2 only. Developed at Gdansk University of Technology.
| Homepage: http://klub.com.pl/portproxy/
Detected interfaces:
  Interface lo, 1 address(es), MAC=00:00:00:00:00:00
    IPv4=127.0.0.1
  Interface eth0, 0 address(es), MAC=00:1d:7e:bc:41:2a
  Interface eth0.0, 0 address(es), MAC=00:1d:7e:bc:41:2a
  Interface eth0.1, 0 address(es), MAC=00:1d:7e:bc:41:2a
  Interface br-lan, 1 address(es), MAC=00:1d:7e:bc:41:2a
    IPv4=192.168.1.1
  Interface ip6tnl0, 0 address(es), MAC=00:00:00:00:00:00
  Interface softwire, 1 address(es), MAC=20:00:00:00:00:00
    IPv4=10.0.0.100
Configuration:
  lan=br-lan
  wan=softwire
  lanIP=192.168.1.1
  wanIP=10.0.0.100
  routerIP=10.0.0.36
  natpmp_udp_port=5351
  upnp_udp_port=1900
  upnp_tcp_port1=2869
  upnp_tcp_port2=2857
  fwd_tcp_port1=2869
  fwd_tcp_port2=2857
  shared_udp_sock=YES
  enable_natpmp=YES
  enable_upnp_tcp=NO
  enable_upnp_udp=YES

```

```
Enabling uPNP (UDP)
Creating UDP socket: addr=0.0.0.0, port=1900 on interface br-lan, result=4
UDP(SSDP) socket created: sockid=4, port=1900
Successfully joint multicast group 239.255.255.250 (local IP=192.168.1.1)
Creating UDP socket: addr=0.0.0.0, port=1900 on interface softwire, result=5
UDP(SSDP) socket created: sockid=5, port=1900
Successfully joint multicast group 239.255.255.250 (local IP=10.0.0.100)
Sockets 4(LAN:br-lan) and 5(WAN:softwire) paired.
Enabling NAT-PMP
Creating UDP socket: addr=192.168.1.1, port=5351 on interface br-lan,
result=7
UDP: sockid=7, port=5351
Opening single UDP transmission socket (random port)
Creating UDP socket: addr=10.0.0.100, port=0 on interface softwire, result=8
UDP: sockid=8, port=47877

Waiting for incoming packets (timeout: 3600s, 0us), 0 mappings (0 pkts
handled)
Received UDP packet: 269 bytes (sock-id=5): 10.0.0.36/port 51608 ->
239.255.255.250/port 1900
Mcast pkt received on WAN interface (pkt->sock=5)
Forwarding SSDP multicast to LAN

Waiting for incoming packets (timeout: 3600s, 0us), 0 mappings (1 pkts
handled)
Received UDP packet: 341 bytes (sock-id=5): 10.0.0.36/port 51608 ->
239.255.255.250/port 1900
Mcast pkt received on WAN interface (pkt->sock=5)
Forwarding SSDP multicast to LAN

Waiting for incoming packets (timeout: 3600s, 0us), 0 mappings (2 pkts
handled)
Received UDP packet: 337 bytes (sock-id=5): 10.0.0.36/port 51608 ->
239.255.255.250/port 1900
Mcast pkt received on WAN interface (pkt->sock=5)
Forwarding SSDP multicast to LAN

Waiting for incoming packets (timeout: 3600s, 0us), 0 mappings (3 pkts
handled)
Received UDP packet: 317 bytes (sock-id=5): 10.0.0.36/port 51608 ->
239.255.255.250/port 1900
Mcast pkt received on WAN interface (pkt->sock=5)
Forwarding SSDP multicast to LAN

Waiting for incoming packets (timeout: 3600s, 0us), 0 mappings (4 pkts
handled)
Received UDP packet: 349 bytes (sock-id=5): 10.0.0.36/port 51608 ->
239.255.255.250/port 1900
Mcast pkt received on WAN interface (pkt->sock=5)
Forwarding SSDP multicast to LAN

Waiting for incoming packets (timeout: 3600s, 0us), 0 mappings (5 pkts
handled)
Received UDP packet: 331 bytes (sock-id=5): 10.0.0.36/port 51608 ->
239.255.255.250/port 1900
Mcast pkt received on WAN interface (pkt->sock=5)
Forwarding SSDP multicast to LAN

Waiting for incoming packets (timeout: 3600s, 0us), 0 mappings (6 pkts
handled)
Received UDP packet: 333 bytes (sock-id=5): 10.0.0.36/port 51608 ->
239.255.255.250/port 1900
```

```
Mcast pkt received on WAN interface (pkt->sock=5)
Forwarding SSDP multicast to LAN

Waiting for incoming packets (timeout: 3600s, 0us), 0 mappings (7 pkts
handled)
Received UDP packet: 333 bytes (sock-id=5): 10.0.0.36/port 51608 ->
239.255.255.250/port 1900
Mcast pkt received on WAN interface (pkt->sock=5)
Forwarding SSDP multicast to LAN

Waiting for incoming packets (timeout: 3600s, 0us), 0 mappings (8 pkts
handled)
Received UDP packet: 2 bytes (sock-id=7): 192.168.1.2/port 48184 ->
192.168.1.1/port 5351
Mapping UDP: src=192.168.1.2/port 48184,dst=192.168.1.1/port 5351 =>
src=10.0.0.100/port 47877,dst=10.0.0.36/port 5351 (sock=7/8)
Mapping UDP: src=192.168.1.2/port 48184,dst=192.168.1.1/port 5351 =>
src=10.0.0.100/port 47877,dst=10.0.0.36/port 5351 (sock=7/8)
Forwarding UDP query (2 bytes) to 10.0.0.36/port 5351

Waiting for incoming packets (timeout: 10s, 0us), 1 mappings (9 pkts
handled)
Received UDP packet: 12 bytes (sock-id=8): 10.0.0.36/port 5351 ->
10.0.0.100/port 47877
Handling response using following mapping:
Mapping UDP: src=192.168.1.2/port 48184,dst=192.168.1.1/port 5351 =>
src=10.0.0.100/port 47877,dst=10.0.0.36/port 5351 (sock=7/8)
Forwarding UDP response (12 bytes) to 10.0.0.36/port 48184

Waiting for incoming packets (timeout: 3600s, 0us), 0 mappings (10 pkts
handled)
Received UDP packet: 137 bytes (sock-id=4): 192.168.1.2/port 35602 ->
239.255.255.250/port 1900
Mapping UDP: src=192.168.1.2/port 35602,dst=239.255.255.250/port 1900 =>
src=10.0.0.100/port 47877,dst=239.255.255.250/port 1900 (sock=4/8)
Mapping UDP: src=192.168.1.2/port 35602,dst=239.255.255.250/port 1900 =>
src=10.0.0.100/port 47877,dst=239.255.255.250/port 1900 (sock=4/8)
Forwarding UDP query (137 bytes) to 239.255.255.250/port 1900

Waiting for incoming packets (timeout: 10s, 0us), 1 mappings (11 pkts
handled)
Received UDP packet: 306 bytes (sock-id=8): 10.0.0.36/port 1900 ->
10.0.0.100/port 47877
Handling response using following mapping:
Mapping UDP: src=192.168.1.2/port 35602,dst=239.255.255.250/port 1900 =>
src=10.0.0.100/port 47877,dst=239.255.255.250/port 1900 (sock=4/8)
Forwarding UDP response (306 bytes) to 239.255.255.250/port 35602

Waiting for incoming packets (timeout: 3600s, 0us), 0 mappings (12 pkts
handled)
Received UDP packet: 137 bytes (sock-id=5): 10.0.0.100/port 47877 ->
239.255.255.250/port 1900
Mcast pkt received on WAN interface (pkt->sock=5)
Forwarding SSDP multicast to LAN

Waiting for incoming packets (timeout: 3600s, 0us), 0 mappings (13 pkts
handled)
Received UDP packet: 149 bytes (sock-id=5): 10.0.0.36/port 51608 ->
239.255.255.250/port 1900
Mcast pkt received on WAN interface (pkt->sock=5)
Forwarding SSDP multicast to LAN
```

```
Waiting for incoming packets (timeout: 3600s, 0us), 0 mappings (14 pkts handled)
```

## 8 Source code

### 8.1 Portproxy source code

**TODO:**

#### 8.1.1 Concepts

**TODO:**

- describe multicast support
- UDP mappings (+timeouts)
- TCP forwarding

#### 8.1.2 Architecture

**TODO**

## 9 Firmware installation

There are no special techniques required to install provided firmware. Reader familiar with the firmware flashing on WRT devices may skip this section. It is provided as a reference for less experienced users.

### 9.1 Connecting LinkSys

Before any configuration or firmware modification, make sure that you have full connectivity to your LinkSys device. LinkSys devices by default use 192.168.1.1 address, so to communicate with it, another address from 192.168.1.0/24 pool is required. For example, PC may be configured to use 192.168.1.100/24 address. To check if you have connectivity with LinkSys device, use following command:

```
ping 192.168.1.1
```

Make sure that you have LinkSys connected using the rightmost socket. See Fig.13 below.



*Fig. 13: Connecting LinkSys to LAN*

Note: IPv4 address set to 192.168.1.1 is a LinkSys' default setup. It reverts to this configuration after every firmware upgrade. Also it is its default factory configuration.

## **9.2 Firmware upgrade (using original web interface)**

Before attempting to install Linux on LinkSys device, make sure that this particular model is supported. Please consult <http://wiki.openwrt.org/TableOfHardware> . Note that even small deviations are important. Sometimes version 1.0 and 1.1 are quite different. See Fig. 2 for example how to check your particular model. In general, at least 4MB flash and 16MB ram is required.

Linux installation on Linksys is being performed as a firmware upgrade. During the first installation, original web interface (provided by LinkSys) should be used. From PC using the same address space (see section 3.1), use web browser to connect to your LinkSys web interface. See Fig.14 below.

Select appropriate firmware image (i.e. file that ends with -squashfs.bin and is corresponding to the name of used device). There are some sanity checks in the firmware upgrade procedure, but using wrong image may result in rendering the router unusable. You may want to check supported hardware list: <http://wiki.openwrt.org/TableOfHardware?action=show&redirect=toh> Please verify that you are using proper firmware. After firmware was uploaded, flashing takes place. It can take up to 2 minutes. After completion, router will reboot. Make sure to not reboot or power off the router before it finishes flashing.

It also may be beneficial to read following installation guide: <http://wiki.openwrt.org/InstallingWrt54gl>

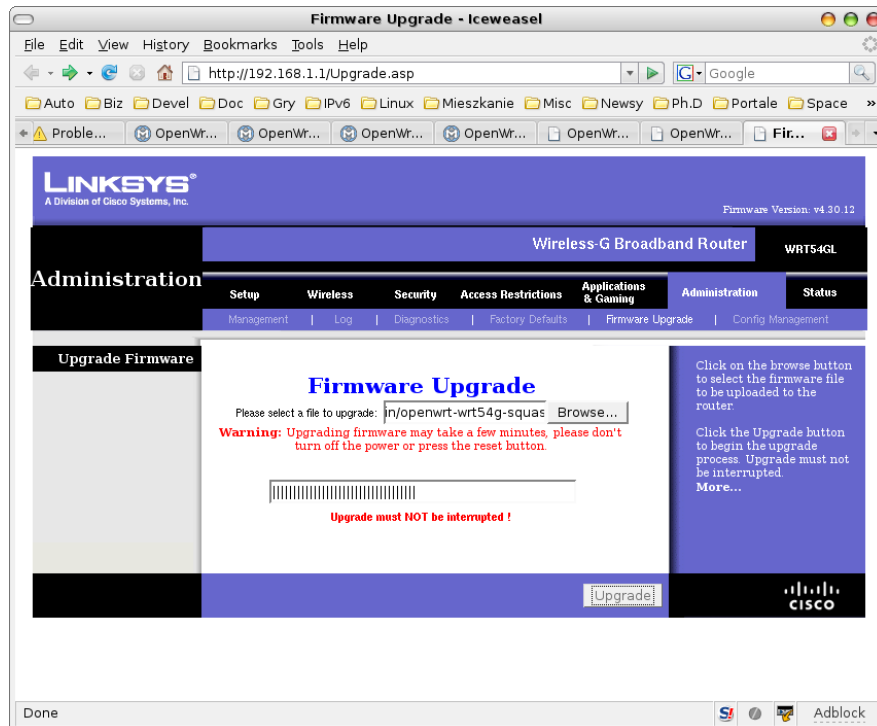


Fig. 14: Firmware upgrade using original web interface

### 9.3 Firmware upgrade (using linux console)

Firmware upgrade from OpenWRT (i.e. when your LinkSys device was flashed already) is done by using mtd tool. Copy openwrt-brcm47xx-squashfs.trx file to the /tmp directory. Note that this is a different image file than used in the web interface. Copy it to your LinkSys device:

```
scp openwrt-brcm47xx-squashfs.trx root@192.168.1.1:/tmp
```

This command will copy required firmware image to /tmp directory. Change to that directory and begin flashing, using following command:

```
cd /tmp
mtd -r write openwrt-brcm47xx-squashfs.trx linux
```

After flashing is complete, device will reboot. It takes up to 2 minutes to finish flashing and rebooting. Please note that after such firmware upgrade, all possible changes made to the router configuration will be lost. That includes all software packages installed and all configuration changes.

Note: .bin and .trx firmware image files contain the same image, but .trx is a "raw" image, while .bin has extra headers for the purpose of being recognized as a valid image by the original web interface.

Note: mtd is a command-line tool. As most of the OpenWRT software it may come pre-installed or as a separate package. If the mtd is missing, simply install mtd\_7\_mipsel.ipk package.

### 9.4 Firmware upgrade (TFTP)

Another way to install new firmware is to use TFTP protocol. When boot\_wait phase is enabled (see section 3.5), it is possible to upload new firmware using TFTP protocol. On a Linux box that uses IPv4 address from the same class (see section 3.1), use TFTP client to send firmware. For

example:

```
tftp 192.168.1.1
tftp>binary
tftp>rexmt 1
tftp>timeout 60
tftp>trace
Packet tracing on.
tftp> put openwrt-wrt54g-2.4-squashfs.bin
```

In the example above, client will try to send firmware at 1 second intervals. It will give up after 60 seconds. After those commands are typed, shut down and then boot your device. Following messages should be displayed:

```
$tftp 192.168.1.1
tftp> binary
tftp> rexmt 1
tftp> timeout 120
tftp> trace
Packet tracing on.
tftp> put openwrt-wrt54g-squashfs.bin
sent WRQ <file=openwrt-wrt54g-squashfs.bin, mode=octet>
sent WRQ <file=openwrt-wrt54g-squashfs.bin, mode=octet>
sent WRQ <file=openwrt-wrt54g-squashfs.bin, mode=octet>
sent WRQ <file=openwrt-wrt54g-squashfs.bin, mode=octet>
sent WRQ <file=openwrt-wrt54g-squashfs.bin, mode=octet>
sent WRQ <file=openwrt-wrt54g-squashfs.bin, mode=octet>
received ACK <block=0>
sent DATA <block=1, 512 bytes>
received ACK <block=1>
sent DATA <block=2, 512 bytes>
...
received ACK <block=3592>
sent DATA <block=3593, 32 bytes>
received ACK <block=3593>
Sent 1839136 bytes in 15.1 seconds
```

Note: bin images are expected to be uploaded via TFTP, not the raw trx images. Attempt to upload trx image (or in fact any other file that does not have proper headers) will cause bootloader to reject the file:

```
received ACK <block=0>
sent DATA <block=1, 512 bytes>
received ACK <block=0>
sent DATA <block=1, 512 bytes>
received ERROR <code=4, msg=code pattern incorrect>
Error code 4: code pattern incorrect
```

## 9.5 Enabling boot\_wait phase

Most embedded devices have multi-stage boot process. After the device is powered up, it runs bootloader. Its main task is to check if flash memory contains proper firmware, load and run it. Broken, corrupted or non-functional firmware may cause the endless loop of firmware loading and reboots or device crash. This state of device being unusable is often referred to as “bricked”. There are basically 2 methods of recovering such device:

1. Use JTAG connector to upload new firmware
2. Use TFTP to upload new firmware via network

As the first option requires hardware modification (most if not all LinkSys devices come without

JTAG connector, so soldering and extra JTAG cable is necessary), it is much easier to use TFTP transfer.

Bootloader may be configured to wait specified period for incoming TFTP packets. However, this waiting phase delays device boot, so vendors often disable this feature. Fortunately, it can be reenabled. To enable boot\_wait phase, use following commands from the command-line:

```
nvrn set boot_wait=on
nvrn commit
```

Note: this feature works under Linux kernels 2.4 only. For that purpose, you may want to use any stable OpenWRT firmware that is kernel-2.4 based. Once boot\_wait is enabled, it is safe to experiment with new firmware images (kernel 2.6 based for example). One way to obtain 2.4 based firmware is from OpenWRT homepage: <http://downloads.openwrt.org/kamikaze/7.09/>

## 9.6 First connection

After performing firmware upgrade, it is possible to connect to the router using telnet command. Please run following command: telnet 192.168.1.1 from a PC console. See fig. 3 for example session. There is no root password. Please change root password by issuing passwd command. After this operation, telnet service will be disabled. SSH will be enabled instead. Note that ssh requires some time (~30 seconds) to generate keys, so it will reject any connection attempts at that time.

## 9.7 ipk packages

OpenWRT provides wide set of network related tools. Due to flash memory constraints, often only a limited set of tools may be installed. They are managed as ipk packages.

During firmware compilation, every piece of software may be selected to be built as part of the firmware <\*>, separate package <M> or not built at all <>. It is advisable to include in the base firmware only those really necessary packages, as packages built-in into firmware cannot be deinstalled. (It is possible to remove them using ipkg remove command, but they will only become invisible and still take up precious space on flash memory).

## 9.8 Package installation

Software installation is being done using ipk packages. To install a package please copy it to /tmp directory, e.g. using scp command (run on a PC):

```
scp mtd_7_mipsel.ipk root@192.168.1.1:/tmp
```

After package is transferred to the device, use following command:

```
ipkg install /tmp/package_name.ipk
```

to install package. For example, to install mtd, use following command :

```
ipkg install /tmp/mtd_7_mipsel.ipk
```

Following packages are currently recommended for the ip46nat project:

- kernel\_2.6.25.16 – dummy package that is required by other packages.
- mtd - mtd is a tool used to flash router. That is the preferred way to flash router, once Linux have been installed.
- ip - powerful tool used for network configuration. Part of the iproute2 suite. For example, interfaces, addresses and tunnels are configured using this tool.

- iptables - optional package, may be used to configure IPv4 firewall and/or IPv4 only NAT.
- ip6tables - optional package, may be used to configure IPv6 firewall and/or IPv6 only NAT.
- uclibcxx - C++ library required to run all software written in C++, e.g. dibbler software
- dibbler-client - DHCPv6 client, used to retrieve configuration and configure the device.
- dibbler-server - DHCPv6 server, used to distribute addresses and configuration parameters to other nodes.

Kernel modules are also handled as ipk packages. To distinguish between user-space software and kernel modules, the latter use kmod- prefix. Here is a list of useful kernel modules:

- kmod-ipv6 – module that provides IPv6 capability.
- kmod-ip46nat – provides IPv4-to-IPv6 NAT functionality.
- kmod-ip6-tunnel – provides IPv4 over IPv6 tunneling. Requires iptunnel6 module to be present.
- kmod-iptunnel6 – this module is required by ip6-tunnel. It also requires IPv6 module.
- iptables – firewall and advanced routing tool
- ip6tables – IPv6 version of the iptables

For information how to compile additional software, see section 7.

Also keep in mind available space on the device. Use df -h command, if necessary. According to OpenWRT homepage, when base partition is full, various errors start to appear and firmware may get corrupted.

## 10 Best practices and debugging tips

This section provides recommendations, useful observations and best practices.

- Before you start any risky firmware upgrade, make sure that boot\_wait is enabled (see section 3.5). Firmware with Linux kernel 2.4 is required.
- Network configuration is specified in the /etc/config/network file. It contains information regarding port assignments, used IPv4 addresses and vlan assignments. For details, see section 4.
- When modifying scripts for OpenWRT, keep in mind that it uses ash instead of bash. The differences are rather minimal (e.g. different function definitions). For details regarding ash environment, see: <http://www.thelinuxblog.com/linux-man-pages/1/ash>

## 11 Links

Following links are recommended reading:

- <http://klub.com.pl/portproxy/> - ip46nat project homepage
- <http://upnp.org/standardizeddcps/default.asp> – UPnP specification
- <http://downloads.openwrt.org/kamikaze/docs/openwrt.html> - OpenWRT manual
- <http://tools.ietf.org/html/draft-cheshire-nat-pmp-03> – NAT-PMP draft (expired)
- <http://miniupnp.free.fr/> - MiniUPnP project homepage
- <http://miniupnp.free.fr/libnatpmp.html> – libnatpmp homepage

## 12 Contact

Author of this project can be reached using e-mail <mailto:tomasz.mrugalski@gmail.com>.