

# ip46nat User's Guide

Author: Tomasz Mrugalski  
version 2008-07-02

# Table of contents

|     |  |    |
|-----|--|----|
| 1   | Project overview.....                                | 3  |
| 1.1 | Phase 1: IPv4 to IPv6 NAT.....                       | 3  |
| 1.2 | Phase 2: IPv4 over IPv6 tunnelling.....              | 3  |
| 2   | Project status.....                                  | 3  |
| 2.1 | Latest status.....                                   | 3  |
| 2.2 | Revision history.....                                | 3  |
| 3   | Installation.....                                    | 4  |
| 3.1 | Connecting LinkSys.....                              | 4  |
| 3.2 | Firmware upgrade (using original web interface)..... | 4  |
| 3.3 | First connection.....                                | 6  |
| 3.4 | Package installation.....                            | 6  |
| 3.5 | Firmware upgrade (using linux console).....          | 7  |
| 3.6 | Installing ip46nat kernel module.....                | 7  |
| 4   | Network configuration.....                           | 8  |
| 5   | NAT operation.....                                   | 9  |
| 5.1 | IPv4 to IPv6 NAT.....                                | 9  |
| 5.2 | IPv6 to IPv4 NAT.....                                | 9  |
| 5.3 | Dibbler installation.....                            | 9  |
| 5.4 | Using dibbler to control NAT.....                    | 11 |
| 6   | Compilation.....                                     | 11 |
| 6.1 | Linux for LinkSys (OpenWRT).....                     | 12 |
| 6.2 | ip46nat kernel module.....                           | 13 |
| 6.3 | ip46nat kernel module as an ipk package.....         | 13 |
| 6.4 | dibbler software.....                                | 14 |
| 6.5 | dibbler software as an ipk package.....              | 14 |
| 7   | Phase 1 testing.....                                 | 15 |
| 7.1 | IPv4 to IPv6 traffic.....                            | 15 |
| 7.2 | IPv6 to IPv4 traffic.....                            | 17 |
| 7.3 | Firewall configuration.....                          | 19 |
| 7.4 | Negative testing.....                                | 19 |
| 7.5 | Statistics.....                                      | 20 |
| 7.6 | Test conclusion.....                                 | 20 |
| 8   | Links.....   | 20 |
| 9   | Contact.....   | 21 |

# 1 Project overview

Goal of this project is to provide solution for seamlessly forward IPv4 traffic over IPv6 networks. There are two possible migration modes: IPv4 to IPv6 NAT (see section 1.1 for details) and IPv4 over PPP over L2TP over IPv6.

This document describes usage, installation and configuration of the software required to setup and run remotely configurable router capable of handling two different post-IPv4 traffic types: IPv4 to IPv6 translation and IPv4 over IPv6 tunneling. In particular, new features of the Dibbler software (implementation of the DHCPv6 server and client) used for remote automatic configuration is described.

In particular, following software will be developed: enhancements to the Dibbler software, new kernel modules for IPv4 to IPv6 encapsulation and IPv4 over IPv6 tunneling, porting Dibbler code to LinkSys™ embedded device. Although developed software is delivered mainly in an easier to use compiled form, source code is also provided. This document describes procedures required to build toolchain used for cross-compilation, and the compilation of the developed code as well.

## 1.1 Phase 1: IPv4 to IPv6 NAT

First approach to the IPv4 over IPv6 network assumes that incoming IPv4 packets will be converted and forwarded as IPv6. IPv4 addresses will be "expanded" into full IPv6 addresses, using 2 extra prefixes: M and P. Returning IPv6 packets will be converted back to IPv4.

## 1.2 Phase 2: IPv4 over IPv6 tunnelling

Second approach assumes that IPv4 packets will be tunelled over IPv6. Every IPv4 packet will be excapsulated in extra IPv6 header. Extra steps to configure IPv4-in-IPv6 tunnel must be provided, like routing configuration.

# 2 Project status

## 2.1 Latest status

For latest status, list of tasks already completed, work in progress and upcoming tasks, see project webpage (<http://klub.com.pl/ip46nat/>). Should priorities change during code development, please contact Tomasz Mrugalski.

## 2.2 Revision history

| Date       | Version | Primary Author   | Change description  |
|------------|---------|------------------|---|
| 2008-06-04 | -       | Tomasz Mrugalski | Initial documentation release   |
| 2008-06-22 | -       | Tomasz Mrugalski | Current status section removed (link to website provided instead), project overview updated, compilation process described (sections 6.1-6.5 added) |
| 2008-07-02 |         | Tomasz Mrugalski | Dibbler-NAT integration, phase 1 completion testing, extra description regarding device model.  |
|            |         |                  |   |

### 3 Installation

To complete installation, several steps are required. Following sections describe, how to achieve specific goals. In general, there are several approaches:

1. Install required software (i.e. modified Linux kernel + ip46nat module) on a PC machine and use it to perform IPv4-IPv6 NAT. This scenario may be used as an preliminary validation scenario.
2. Install Linux on LinkSys WRT54 system, install ip46nat module.

#### 3.1 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.1 below.



Fig. 1: 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.

#### 3.2 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. 3 for example how to check your particular model.



Fig. 2: LinkSys device model check

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.2 below.

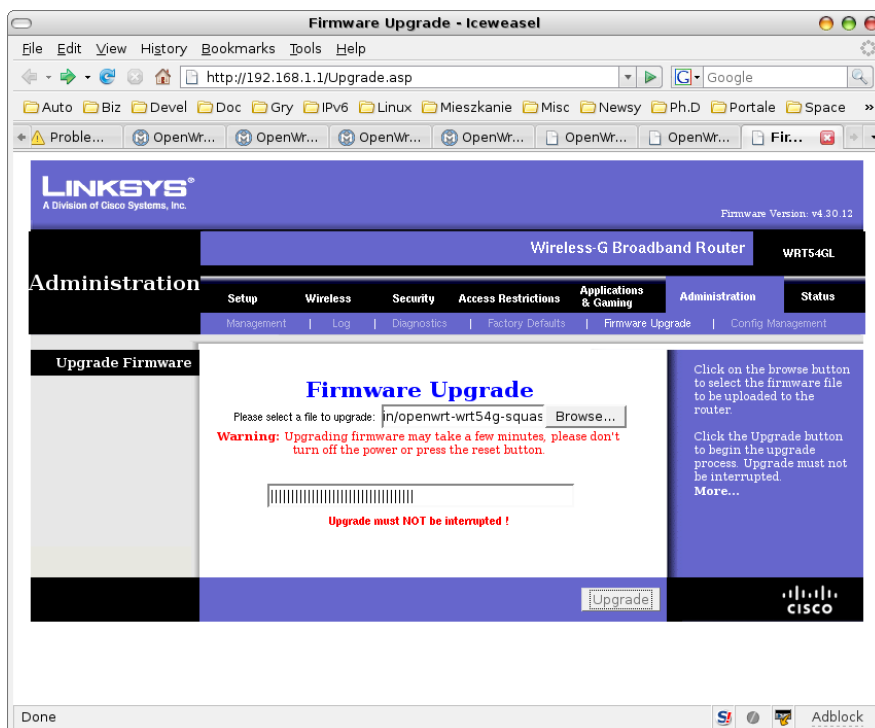


Fig. 3: Firmware upgrade using original web interface

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



Following packages are currently recommended:

- 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
- iptables - optional package, may be used to configure IPv4 firewall and/or IPv4 only NAT.
- ip6table - 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
- dibbler-server - DHCPv6 server, used to distribute addresses and configuration parameters to other nodes.

### **3.5 Firmware upgrade (using linux console)**

To perform firmware upgrade from OpenWRT (i.e. when your LinkSys device was flashed already), mtd tool must be used. 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 recognised as a valid image by the original web interface.

### **3.6 Installing ip46nat kernel module**

To perform IPv4-to-IPv6 NAT, a separate kernel module have been developed. Although it would be possible to achieve similar functionality in the user space, kernel module provides the best efficiency available. For the easiness of installation, it is being distributed as a ipk package. Please install it as any other ipk packages:

```
ipkg install kmod-ip46nat_2.6.23.17-brcm47xx-1_mipsel.ipk
```

After installation is complete, ip46nat kernel module may be loaded, using following command:

```
insmod /lib/modules/2.6.23.17/ip46nat.ko v6prefixm=2000:: v6prefixp=3000::  
v4addr=10.10.1.0
```

Module reports its operation using normal kernel messages. To see kernel output, `dmesg` command may be used. It also appears to be useful to filter `dmesg` output using `tail` command. To see last 10 messages, use following command:

```
dmesg | tail -30
```

After module is loaded, it reports operation readiness and begins to filter incoming traffic immediately:

```
IPv4-IPv6 NAT module loaded: v6prefix: 3000::, v6prefixm: 2000::, v4addr: 10.10.1.0
Handlers for IPv4 and IPv6 installed.
#IPv4 rcvd (rcvd so far: 1) [src=192.168.1.100, dst=192.168.1.1,looking for 10.10.1.0/24]
#IPv4 rcvd (rcvd so far: 2) [src=192.168.1.100, dst=192.168.1.1,looking for 10.10.1.0/24]
#IPv4 rcvd (rcvd so far: 3) [src=192.168.1.100, dst=192.168.1.1,looking for 10.10.1.0/24]
```

Kernel module may be unloaded at any time. To do so, use command:

```
rmmod ip46nat
```

After kernel is unloaded, statistics are being presented. To see them, use `dmesg` command.

```
Handlers for IPv4 and IPv6 removed.
---IPv4-IPv6 NAT statistics-----
IPv4-to-IPv6 packets: 0
IPv6-to-IPv4 packets: 0
IPv4 rcvd: 74, sent: 0
IPv6 rcvd: 0, sent: 0
IPv4 dropped (too large): 0
IPv4 dropped (no route): 0
IPv6 dropped (no route): 0
IPv4 dropped (transmission failed): 0
IPv6 dropped (transmission failed): 0
-----
IPv4-IPv6 NAT module unloaded.
```

Note: from the early demo perspective, it is also possible to compile and run Linux kernel with `ip46nat` module on a PC. For a details regarding module compilation, see section 5.

## 4 Network configuration

Before attempting to perform configuration, it is strongly recommended to be familiar with the following guide: <http://wiki.openwrt.org/OpenWrtDocs/NetworkInterfaces>

All LinkSys WRT routers use one common ethernet interface, duplicated using different vlans. In the latest OpenWRT version, that is being reported as `eth0`, `eth0.0`, `eth0.1` etc.

To enable IPv4 and IPv6 forwarding, use following command:

```
echo 1 > /proc/sys/net/ipv4/conf/all/forwarding
echo 1 > /proc/sys/net/ipv6/conf/all/forwarding
```

To configure routing, `ip` command may be used. Assuming that we want to NAT traffic from `192.168.1.0/24` to `2000::/64`, following command may be used:

```
ip route add 192.168.1.0/24 dev eth0.0
ip route add 2000::/64 dev eth0.1
```

In case of WRT54GL v1.1 (the model used by author), rightmost socket (next to power supply) is called eth0.0 (that is the interface that has default 192.168.1.1 address assigned). Another interface, labeled as WAN, is called eth0.1.

## 5 NAT operation

After module is loaded (see section 3.6 for details), module will start printing information about all received IPv4 and IPv6 traffic. Packets that match configured criteria are marked using \*. When match is found, packet will be recoded and transmitted.

### 5.1 IPv4 to IPv6 NAT

During module insertion, v4addr parameter is specified. It is being used as a source IPv4 address filter, used with /24 bitmask. For example, if v4addr is equal to 192.168.1.0, all incoming traffic from 192.168.1.0/24 network will be translated to IPv6. Source IPv6 address will be created as a concatenation of the P prefix (v6prefixp parameter specified during module insertion) and IPv4 address. Destination IPv6 address will be created as a concatenation of the M prefix (v6prefixm parameter specified during module insertion). TTL field will be copied and decreased by 1.

For converted IPv6 packet to be transmitted successfully, IPv6 forwarding must be enabled. IPv6 routing must also be configured. See section 4 for details.

Please note that L4 layer (TCP, UDP, ICMP, etc.) checksums are not modified in any kind. That means that After IPv4 to IPv6 conversion, packets will not be accepted by destination router. They must be converted back to IPv4. That should not pose any concerns, however, as routers are not supposed to investigate L4 content at all. Thus packets must be converted back to IPv4 before they reach their destination.

Matched packet information will be reported in the following manner:

```
#IPv4 rcvd (rcvd so far: 3) [src=192.168.1.100, dst=192.168.1.1,looking for 192.168.1.0/24] *
```

### 5.2 IPv6 to IPv4 NAT

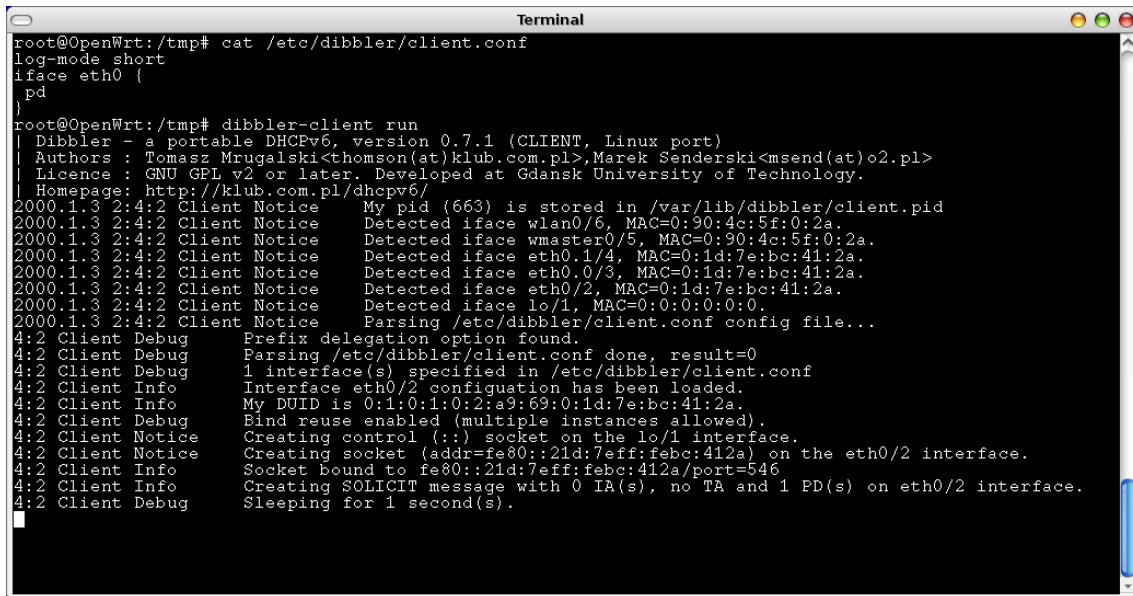
Once IPv4 packets are sent as IPv6, it is expected to receive responses. They are to be received as a IPv6 packets. Source IPv6 address will belong to the M prefix (v6prefixm parameter used during kernel module insertion) and will contain source IPv4 address embedded on 4 least significant octets. Destination IPv6 address will belong to the P prefix (v6prefixp parameter used during kernel module insertion) and will contain destination IPv4 address embedded on 4 least significant octets. If incoming IPv6 packet meets those criteria, it will be converted to IPv4 packet. Header checksum will be calculated, TTL will be decreased and packet will be sent.

For converted IPv4 packet to be transmitted successfully, IPv4 forwarding must be enabled. IPv4 routing must also be configured. See section 4 for details.

### 5.3 Dibbler installation

To install dibbler client, copy dibbler-client\_0.7.1-1\_mipsel.ipk, dibbler-server\_0.7.1-1\_mipsel.ipk and uclibcxx\_0.2.2-1\_mipsel.ipk files to /tmp directory on the OpenWRT box (please use newer





```
root@OpenWrt:/tmp# cat /etc/dibbler/client.conf
log-mode short
iface eth0 {
    pd
}
root@OpenWrt:/tmp# dibbler-client run
Dibbler - a portable DHCPv6, version 0.7.1 (CLIENT, Linux port)
Authors : Tomasz Mrugalski<thomson(at)klub.com.pl>, Marek Senderski<msend(at)o2.pl>
Licence : GNU GPL v2 or later. Developed at Gdansk University of Technology.
Homepage: http://klub.com.pl/dhcpv6/
2000.1.3 2:4:2 Client Notice My pid (663) is stored in /var/lib/dibbler/client.pid
2000.1.3 2:4:2 Client Notice Detected iface wlan0/6, MAC=0:90:4c:5f:0:2a.
2000.1.3 2:4:2 Client Notice Detected iface wmaster0/5, MAC=0:90:4c:5f:0:2a.
2000.1.3 2:4:2 Client Notice Detected iface eth0.1/4, MAC=0:1d:7e:bc:41:2a.
2000.1.3 2:4:2 Client Notice Detected iface eth0.0/3, MAC=0:1d:7e:bc:41:2a.
2000.1.3 2:4:2 Client Notice Detected iface eth0/2, MAC=0:1d:7e:bc:41:2a.
2000.1.3 2:4:2 Client Notice Detected iface lo/1, MAC=0:0:0:0:0:0.
2000.1.3 2:4:2 Client Notice Parsing /etc/dibbler/client.conf config file...
4:2 Client Debug Prefix delegation option found.
4:2 Client Debug Parsing /etc/dibbler/client.conf done, result=0
4:2 Client Debug 1 interface(s) specified in /etc/dibbler/client.conf
4:2 Client Info Interface eth0/2 configuration has been loaded.
4:2 Client Info My DUID is 0:1:0:1:0:2:a9:69:0:1d:7e:bc:41:2a.
4:2 Client Debug Bind reuse enabled (multiple instances allowed).
4:2 Client Notice Creating control (:) socket on the lo/1 interface.
4:2 Client Notice Creating socket (addr=fe80::21d:7eff:febc:412a) on the eth0/2 interface.
4:2 Client Info Socket bound to fe80::21d:7eff:febc:412a/port=546
4:2 Client Info Creating SOLICIT message with 0 IA(s), no TA and 1 PD(s) on eth0/2 interface.
4:2 Client Debug Sleeping for 1 second(s).
```

Fig. 6: Client start

Similar procedure may be used to configure dibbler server.

## 5.4 Using dibbler to control NAT

It is possible to use dibbler-client to control IPv4-to-IPv6 NAT. To do so, dibbler-client must be switched to a special mode. Instead of handling prefixes in a usual manner, it will call special external scripts instead. Those shell scripts may be easily modified to fulfill specific purposes. There are two scripts: mappingprefixadd and mappingprefixdel. Both must be stored in the /var/lib/dibbler directory. During execution, every script will get exactly 1 parameter that will be the (added or deleted) prefix. Here is the example of the configuration file.

```
log-mode short
experimental
mapping-prefix
iface eth0.0 {
    pd
}
```

Experimental keyword instructs dibbler to allow use experimental (i.e. non-standard) code. One of such non-standard extensions is mapping-prefix. Those scripts are the best place to add fixed parameters.

### Firewall – Limiting extra traffic

Internally IPv4-to-IPv6 NAT works as an extra packet handler. It means that each incoming packet may receive extra handling, i.e. conversion to IPv6 or IPv4. Regardless of the outcome (process matches specified criteria or not), it is still being processed by the kernel in a normal way. For example, when criteria matching IPv4 arrives, there will be actually 2 new packets transmitted: first - IPv4 packet according to normal routing, and second – IPv6 packet generated by the ip46nat module. To avoid this behavior, iptables filtering may be used. See section 7.1 for example.

## 6 Compilation

All parts of the ip46net project and its associated software is distributed as an open source.

Therefore full source code is available and can be compiled.

## 6.1 Linux for LinkSys (OpenWRT)

ip46nat project uses embedded linux ditribution 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 a SVN snapshot, revision 11276. To obtain this revision, issue following command:

```
svn co -r 11276 https://svn.openwrt.org/openwrt/trunk/
```

After checkout is complete, initial configuration make take 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.

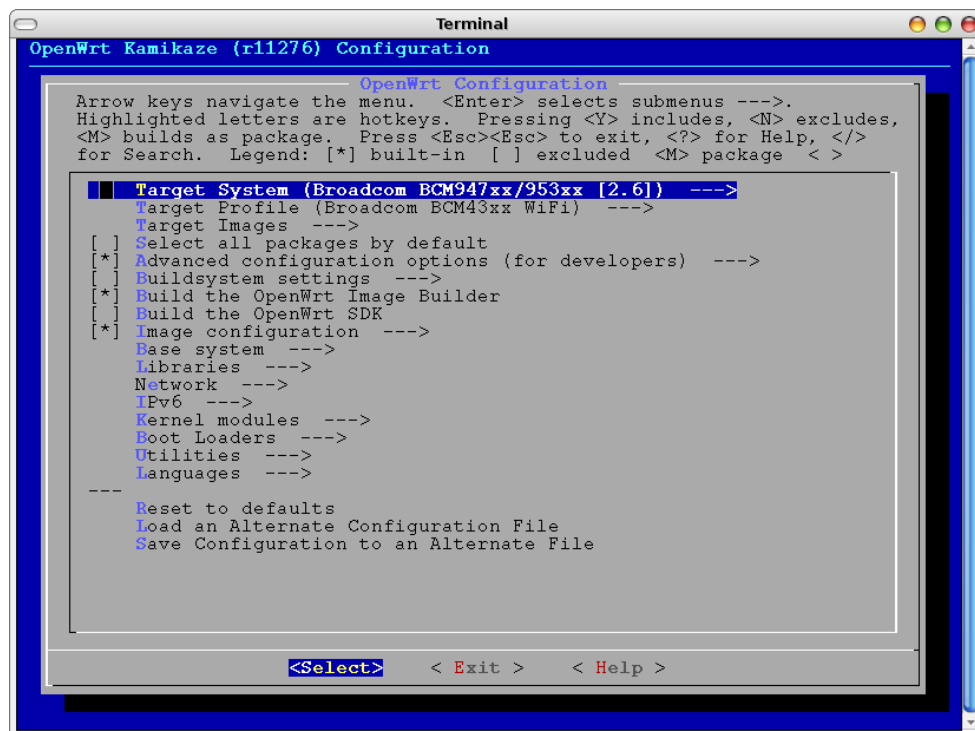


Fig. 7: OpenWRT configuration example

After configuration is complete, type **make** to download requires source code, build toolchain 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.

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.

## 6.2 ip46nat kernel module

ip46nat is a Linux kernel module. Although developed with embedded environment, it is not specific for embedded devices. It may run on any device that runs Linux kernel. That includes ordinary PC boxes. Due to extensive set of debugging and tracking tools available, it may be beneficial to run this module on a PC, at least during early configuration stages.

Ip46nat source code is available as a patch for various Linux kernels. It is strongly recommended to use only those kernels that patches are specifically provided for. In general, it is likely that patch prepared for a specific kernel version will work fine on other, similar version. But it may also fail.

To compile ip46nat as a module on a Linux box, follow this steps:

1. Download supported Linux kernel. This example assumes that Linux kernel 2.6.23.17 will be used.
2. Extract kernel using command: **tar xjvf linux-2.6.23.17.tar.bz2**
3. Apply ip46nat patch: **patch -p0 < ip46nat-2.6.23.17-4.patch**
4. Setup kernel configuration in a normal way: **make menuconfig**
5. Go to Networking => Networking Options => IPv4-IPv6 NAT and select it as a module. If this option is not available, make sure that patch was applied properly and that networking support, IPv4 and IPv6 are enabled.
6. Save kernel configuration (.config file)
7. Build kernel using following command: **make**
8. Build kernel modules using following command: **make modules**
9. Continue with normal kernel installation. See numerous installation help documents available here: <http://www.google.com/search?q=linux+kernel+installation>

## 6.3 ip46nat kernel module as an ipk package

To make ip46nat available from the OpenWRT configuration menu, download selected OpenWRT source (see section 6.1) and apply ip46nat-openWRT patch:

```
patch -p0 < ip46nat-openwrt-11276.patch
```

After successful patching, ip46nat module is available in the Kernel modules => Network support => kmod-ip46nat. Select it as a module. See Fig. 7.

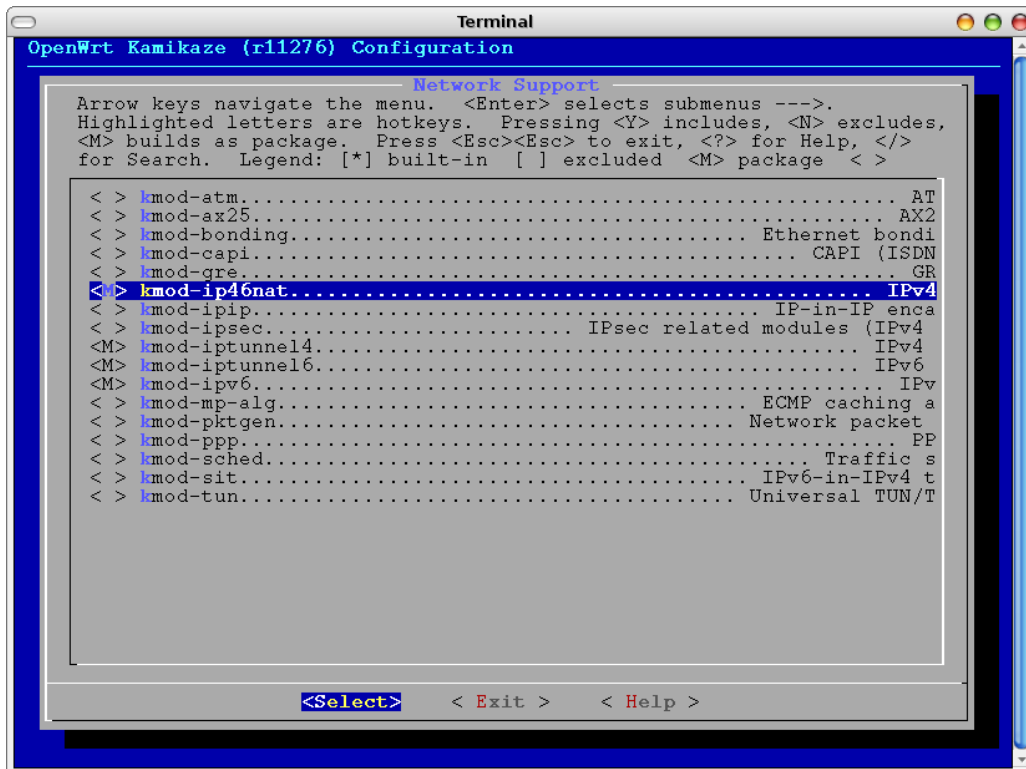


Fig. 8: *ip46nat* being selected in OpenWRT configuration menu

## 6.4 *dibbler* software

Dibbler software may be compiled for various systems: Windows, Linux or even embedded environment. In this section, instructions regarding Linux compilation are provided. For details regarding OpenWRT cross-compilation, please see next section.

Details regarding dibbler compilation are described in the Dibbler User's Guide, available on the project website: <http://klub.com.pl/dhcpv6/>. Advanced topics are also discussed in Dibbler Developer's Guide, also available on the same website.

To compile dibbler, download and extract latest sources. Dibbler consists of several entities: server, client, relay and requestor. To compile one or more components, issue make command followed by name of the component. For example, to compile server, client and relay, following commands may be used:

```
tar zxvf dibbler-0.7.1-src.tar.gz
cd dibbler-0.7.1
make server client relay
```

After compilation is complete, dibbler-server, dibbler-client and dibbler-relay binaries will be available in the current directory.

## 6.5 *dibbler* software as an ipk package

To compile dibbler for OpenWRT environment, several preparatory steps are necessary.

1. Checkout packages repository from the OpenWRT project:  
`svn co https://svn.openwrt.org/openwrt/packages/`

Note that there may be an old dibbler package. Do not use it as it is broken.

2. Copy or symlink packages/libs/uclibc++ directory (from packages repository) to

packages/uclibc++ directory (in the OpenWRT repository).

3. Download and extract openwrt-dibbler-0.7.1.tar.gz file.

After those steps, go to the OpenWRT directory. There should be following directories:

```
docs/  
include/  
package/  
scripts/  
target/  
toolchain/  
tools/
```

and some additional files. In the package directory, there should be (among others) dibbler and uclibc++ directories.

To compile uclibc++ and dibbler packages, please follow instructions from section 6.1. In the OpenWRT configuration menu, go to Libraries => uclibcxx for uClibc++ and IPv6 => dibbler-server, dibbler-client and dibbler-relay packages.

## 7 Phase 1 testing

This section provides report from the validation. It may also serve as a proof of concept. Most tests were performed in a configuration, where 2 PCs were connected via Linksys device. This scenario is presented in Fig. 9 below.

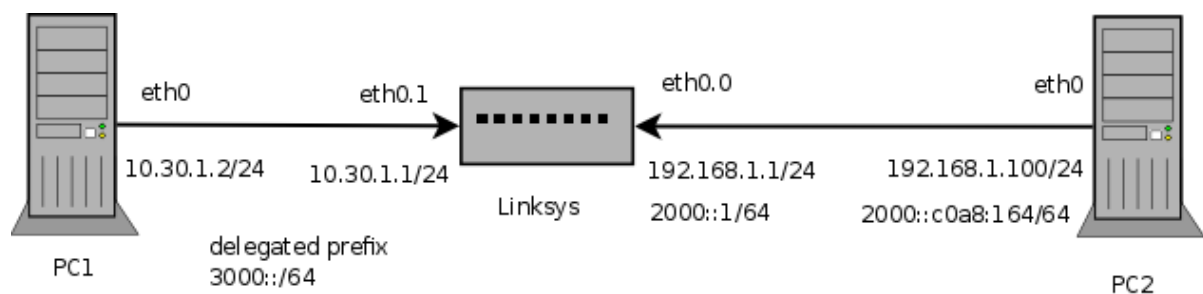


Fig. 9: Test network

### 7.1 IPv4 to IPv6 traffic

IPv4 packets were transmitted from the PC1. Single ICMPv4 packets were sent. Fig. 10 contains network captures of such packets. IPv4 packets after translation (they are IPv6 packets) are presented in Fig. 11. Please note that those IPv6 packets still carry ICMPv4 protocol data.

As packets size after translation increase, there is a size limit of the maximum IPv4 packets that may be translated. Translation adds extra 20 bytes, so maximum packet size is 1480. To transmit such packets, following command may be used:

```
ping -s 1452 192.168.1.100
```

Note that ping command uses -s (size) parameter to specify ICMPv4 protocol payload. That is increased by ICMP header (8 bytes) and IPv4 header (20 bytes). Therefore 1452+8+20 gives 1480. Any packets larger than this will be dropped by ip46nat module. To avoid such drops, the reasonable course of action is to limit MTU of the transmitting device. Also to help debugging such cases,

ip46nat has dedicated statistics for too large IPv4 packet drops.

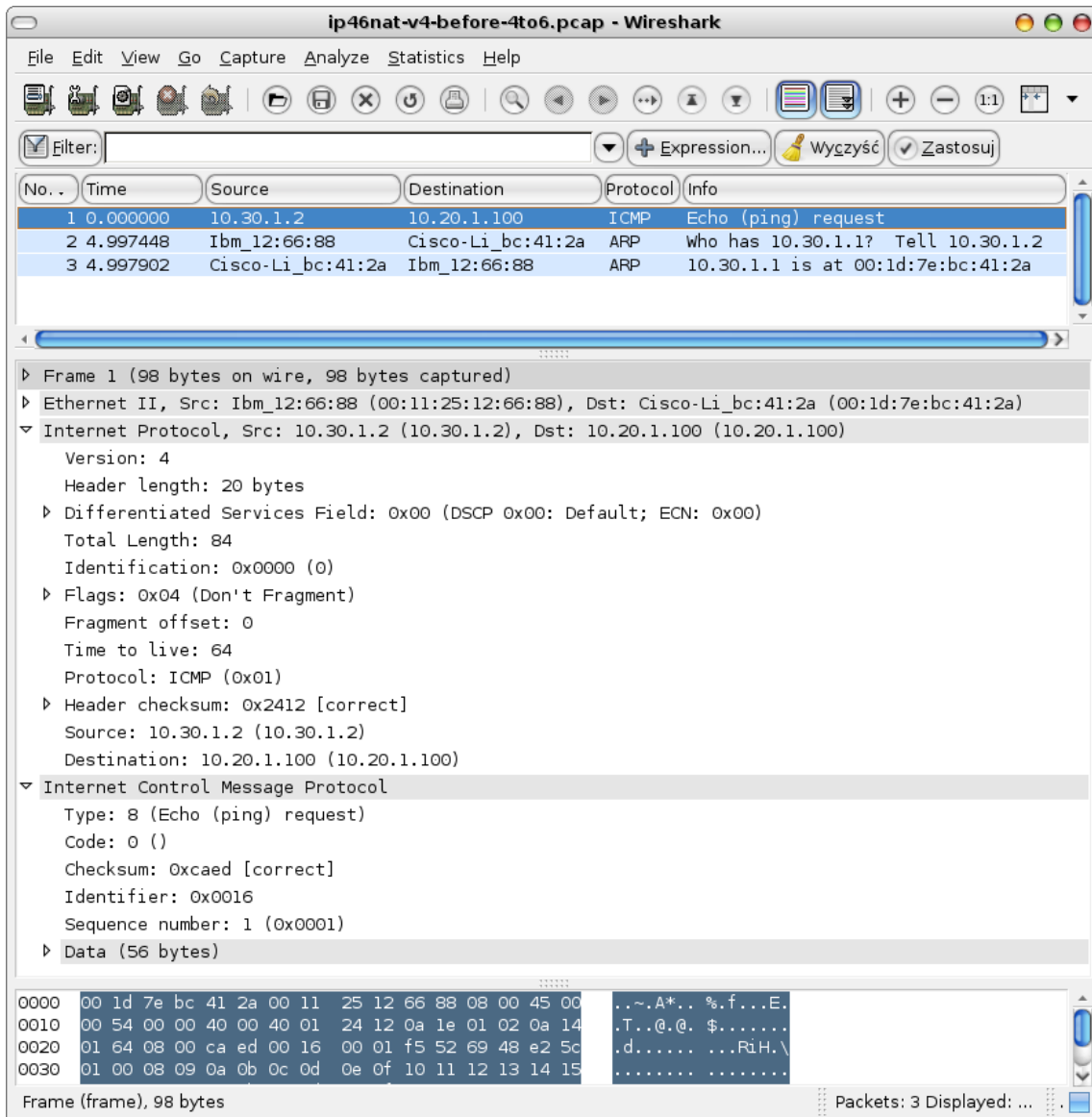


Fig. 10: IPv4 packet before translation

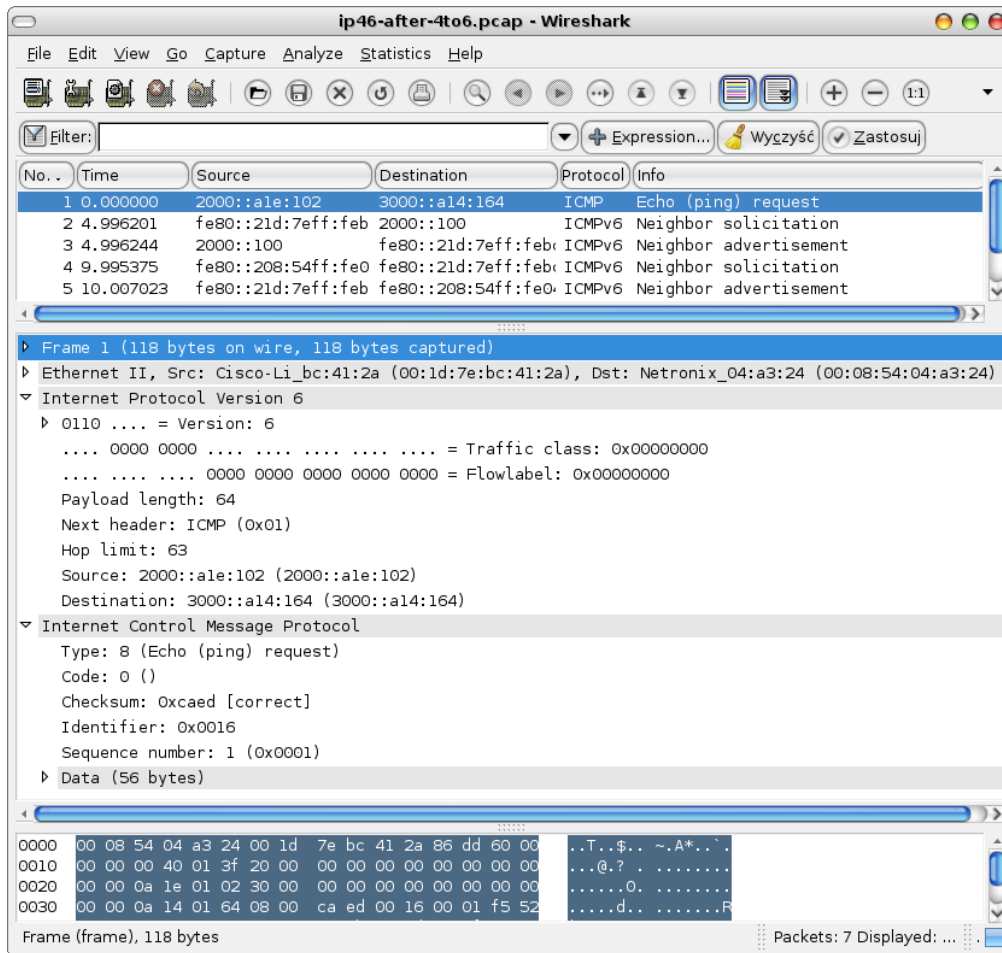


Fig. 11: IPv4 packets after IPv4-to-IPv6 translation

## 7.2 IPv6 to IPv4 traffic

Similar tests were performed for IPv6 traffic being translated to IPv4. See Fig. 12 for packets before translation and Fig. 13 for packets after translation.

Please note that to have packets transmitted, both its source and destination address should be legitimate. For example, IPv6 packet `src=2000::1,dst=2000::c0a8:164` will be translated to IPv4 packet `src=0.0.0.1,dst=192.168.1.100`. Although destination address is valid, source address is not and thus `ip46nat` will not be able to find appropriate route and will drop the packet. This error case will be logged properly. There is also separate statistic for this scenario.

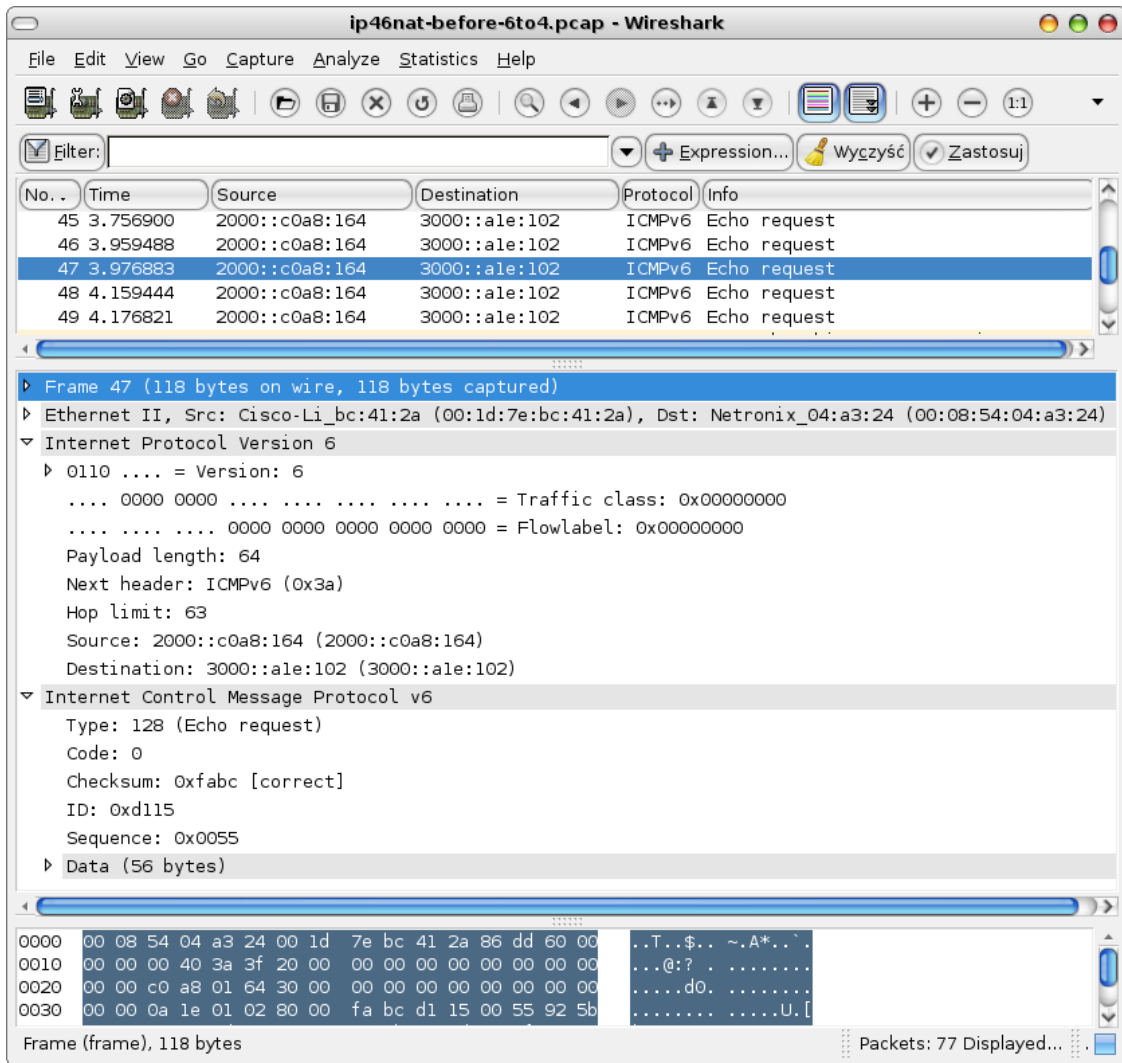


Fig. 12: IPv6 packet before reverse translation

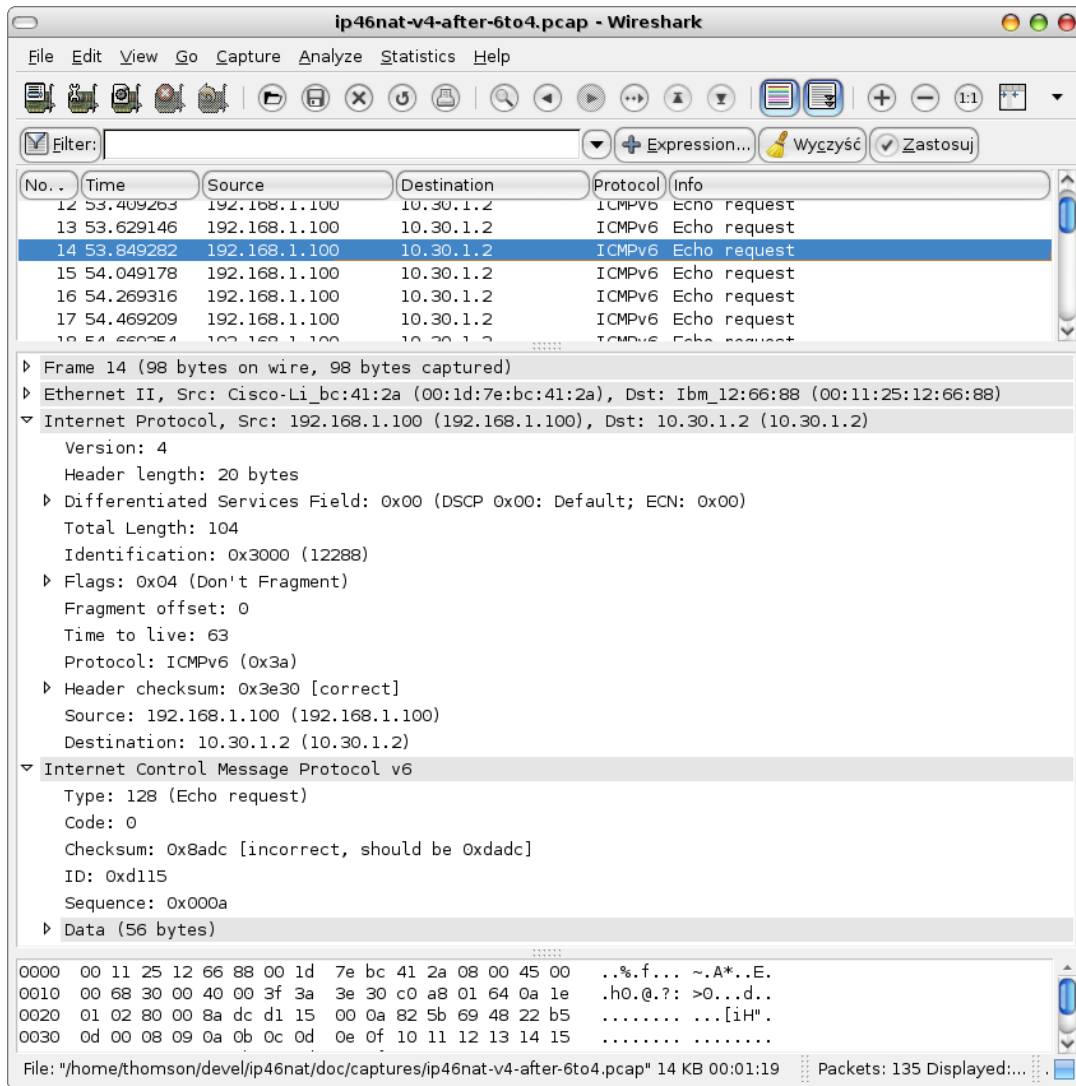


Fig. 13: IPv6 packet after reverse translation

### 7.3 Firewall configuration

To avoid packet duplication, IPv4 packets that are to be translated, should be dropped in a postprocessing phase. Assuming network configuration as in Fig. 9 (and end-user IPv4 segment is on the left side), following command may be used to drop packets AFTER being handled by ip46nat:

```
iptables -t nat -I POSTROUTING -s 10.30.1.0/24 -d ! 10.30.1.0/24 -j DROP
```

### 7.4 Negative testing

Too large IPv4 packets were sent to confirm that no buffer overflow occurs. Packets were dropped as expected and appropriate statistic was increased. Numerous not-matching criteria IPv4 and IPv6 packets were sent. None of them was ever translated. For easier debugging, ip46nat module prints every packet that is received, its source and destination address with values of configured filter. Packets matching criteria are labeled with asterisk (\*).

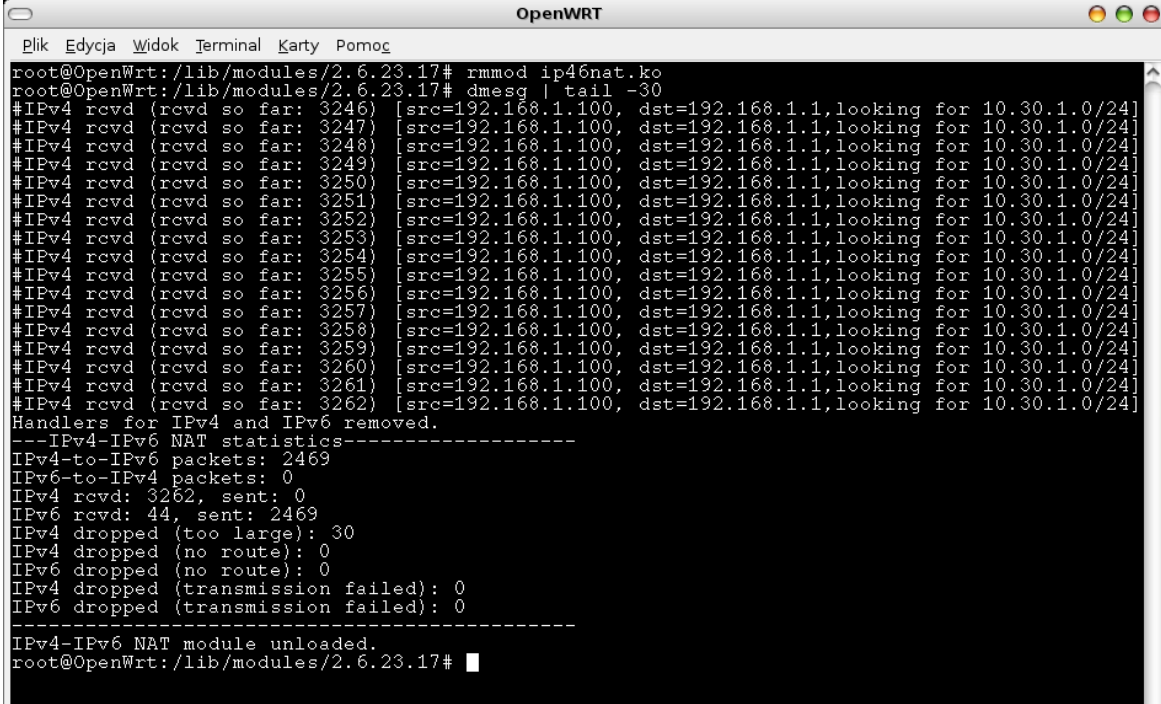
#### Performance testing

To make sure that the device is being able to handle heavy traffic, biggest possible (1480) packets

were transmitted in a continuous manner. After prolonged traffic handle the device behaved normally. See Fig. 14 for statistics after such testing.

## 7.5 Statistics

As specified in Fig.14, there are several statistics implemented. They may be used for overall operation measurements. They are printed after module is unloaded. To see them, use dmesg command. It seems useful to filter dmesg's output, e.g. Using tail -30 command.



```
root@OpenWrt:/lib/modules/2.6.23.17# rmmod ip46nat.ko
root@OpenWrt:/lib/modules/2.6.23.17# dmesg | tail -30
#IPv4 rcvd (rcvd so far: 3246) [src=192.168.1.100, dst=192.168.1.1, looking for 10.30.1.0/24]
#IPv4 rcvd (rcvd so far: 3247) [src=192.168.1.100, dst=192.168.1.1, looking for 10.30.1.0/24]
#IPv4 rcvd (rcvd so far: 3248) [src=192.168.1.100, dst=192.168.1.1, looking for 10.30.1.0/24]
#IPv4 rcvd (rcvd so far: 3249) [src=192.168.1.100, dst=192.168.1.1, looking for 10.30.1.0/24]
#IPv4 rcvd (rcvd so far: 3250) [src=192.168.1.100, dst=192.168.1.1, looking for 10.30.1.0/24]
#IPv4 rcvd (rcvd so far: 3251) [src=192.168.1.100, dst=192.168.1.1, looking for 10.30.1.0/24]
#IPv4 rcvd (rcvd so far: 3252) [src=192.168.1.100, dst=192.168.1.1, looking for 10.30.1.0/24]
#IPv4 rcvd (rcvd so far: 3253) [src=192.168.1.100, dst=192.168.1.1, looking for 10.30.1.0/24]
#IPv4 rcvd (rcvd so far: 3254) [src=192.168.1.100, dst=192.168.1.1, looking for 10.30.1.0/24]
#IPv4 rcvd (rcvd so far: 3255) [src=192.168.1.100, dst=192.168.1.1, looking for 10.30.1.0/24]
#IPv4 rcvd (rcvd so far: 3256) [src=192.168.1.100, dst=192.168.1.1, looking for 10.30.1.0/24]
#IPv4 rcvd (rcvd so far: 3257) [src=192.168.1.100, dst=192.168.1.1, looking for 10.30.1.0/24]
#IPv4 rcvd (rcvd so far: 3258) [src=192.168.1.100, dst=192.168.1.1, looking for 10.30.1.0/24]
#IPv4 rcvd (rcvd so far: 3259) [src=192.168.1.100, dst=192.168.1.1, looking for 10.30.1.0/24]
#IPv4 rcvd (rcvd so far: 3260) [src=192.168.1.100, dst=192.168.1.1, looking for 10.30.1.0/24]
#IPv4 rcvd (rcvd so far: 3261) [src=192.168.1.100, dst=192.168.1.1, looking for 10.30.1.0/24]
#IPv4 rcvd (rcvd so far: 3262) [src=192.168.1.100, dst=192.168.1.1, looking for 10.30.1.0/24]
Handlers for IPv4 and IPv6 removed.
---IPv4-IPv6 NAT statistics-----
IPv4-to-IPv6 packets: 2469
IPv6-to-IPv4 packets: 0
IPv4 rcvd: 3262, sent: 0
IPv6 rcvd: 44, sent: 2469
IPv4 dropped (too large): 30
IPv4 dropped (no route): 0
IPv6 dropped (no route): 0
IPv4 dropped (transmission failed): 0
IPv6 dropped (transmission failed): 0
-----
IPv4-IPv6 NAT module unloaded.
root@OpenWrt:/lib/modules/2.6.23.17#
```

Fig. 14: Statistics after some traffic

## 7.6 Test conclusion

Developed software, while being remotely configurable via DHCPv6, is able to translate IPv4 to IPv6 efficiently. Reverse traffic is also handled properly. Although no end-user solution was provided, all building blocks for experienced user are provided, accompanied with documentation that explains how to achieve the ultimate goal by executing all intermediate steps.

## 8 Links

Following links are recommended reading:

- <http://klub.com.pl/ip46nat/> - ip46nat project homepage.
- <http://openwrt.org/> - OpenWRT project website
- <http://downloads.openwrt.org/kamikaze/docs/openwrt.html> - OpenWRT manual
- <http://klub.com.pl/dhcpv6/> - Dibbler homepage. User's Guide, Developer's Guide and a dibbler source code is available here.

## 9 Contact

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