INNOWACYJNA GOSPODARKA
NARODOWA STRATEGIA SPÓJNOŚCI

inżynieria internetu przyszłości

UNIA EUROPEJSKA
EUROPEJSKI FUNDUSZ
ROZWOJU REGIONALNEGO

ip46nat
Universal IPv4-IPv6 translator
User's Guide

Author: Tomasz Mrugalski
Gdańsk University of Technology
date: 2011-03-12

Table of contents

1 Project overview

Goal of this project is to provide an experimental solution for universal IPv4-to-IPv6 translator. The software is intended to be run on any system with modern Linux. Two modes of Network Address Translation (NAT) are provided: translation of IPv4 packets to IPv6 packets (NAT46) and reverse translation of IPv6 packets to IPv4 (NAT64). The goal of this project is to provide reference implementation for experiments. As such, also IPv4 over IPv6 tunneling is also provided. There are two intended platform to run this software: regular x86 PC and an embedded router, running OpenWRT system.

This document describes usage, installation and configuration of the software required to setup and run router capable of handling two different post-IPv4 traffic types: IPv4 to IPv6 translation and IPv4 over IPv6 tunneling.

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 mapping prefixes. Returning IPv6 packets will be converted back to IPv4.

1.2 Phase 2: IPv4 over IPv6 tunneling

Second approach assumes that IPv4 packets will be tunneled over IPv6. Every IPv4 packet will be encapsulated 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 web page (http://klub.com.pl/ip46nat/). Should priorities change during code development, please contact Tomasz Mrugalski. As of 2010-09-02, both phases are in prototype phase.

2.2 Revision history

Initial release is planned for December 2010.

3 OpenWRT Installation

This chapter describes installation process on home router devices. In particular, it provides step by step installation for LinkSys WRT54GL devices. Reader may skip this chapter if there is no need to run the software on LinkSys device. To complete installation, several steps are required. Following sections describe, how to achieve specific steps. In general, following actions are required:

1. Install Linux on LinkSys WRT54 device (or similar)
2. Install ip46nat module (required for IPv4-to-IPv6 NAT)
3. Install ip6_tunnel (required for IPv4-over-IPv6 tunneling)
4. Install and setup dibbler-client for remote configuration (optional step)
5. Install dibbler-server on a PC. Server will provide configuration details for the client. (optional step)

It is also possible to install required software (i.e. modified Linux kernel + modules) on a PC machine and use it to perform IPv4-IPv6 NAT or IPv4-over-IPv6 tunneling. This approach may be used as validation or debugging environment.

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.

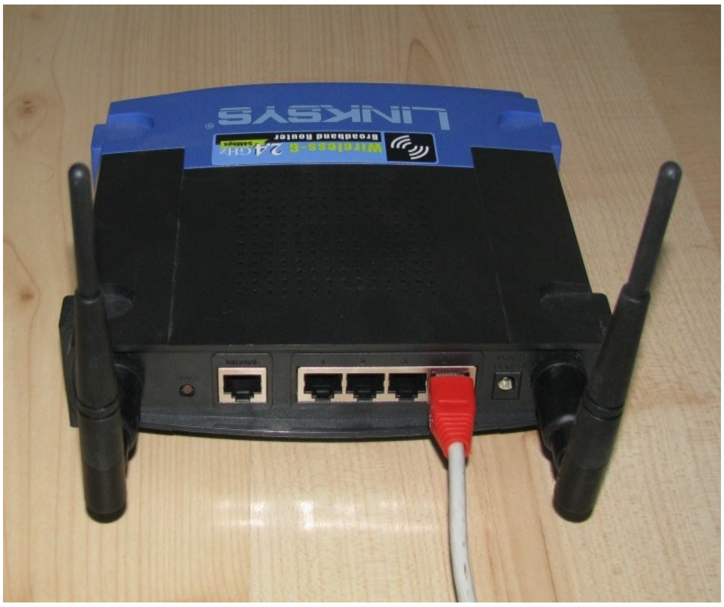


Fig. 1: Connecting LinkSys to LAN

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)

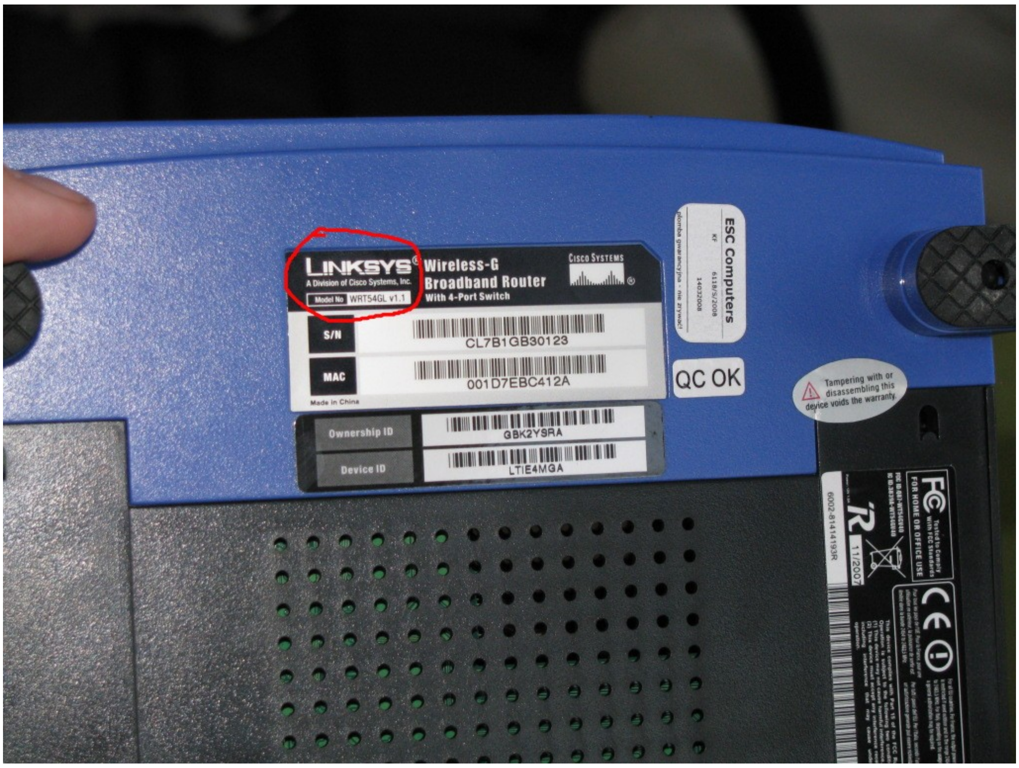


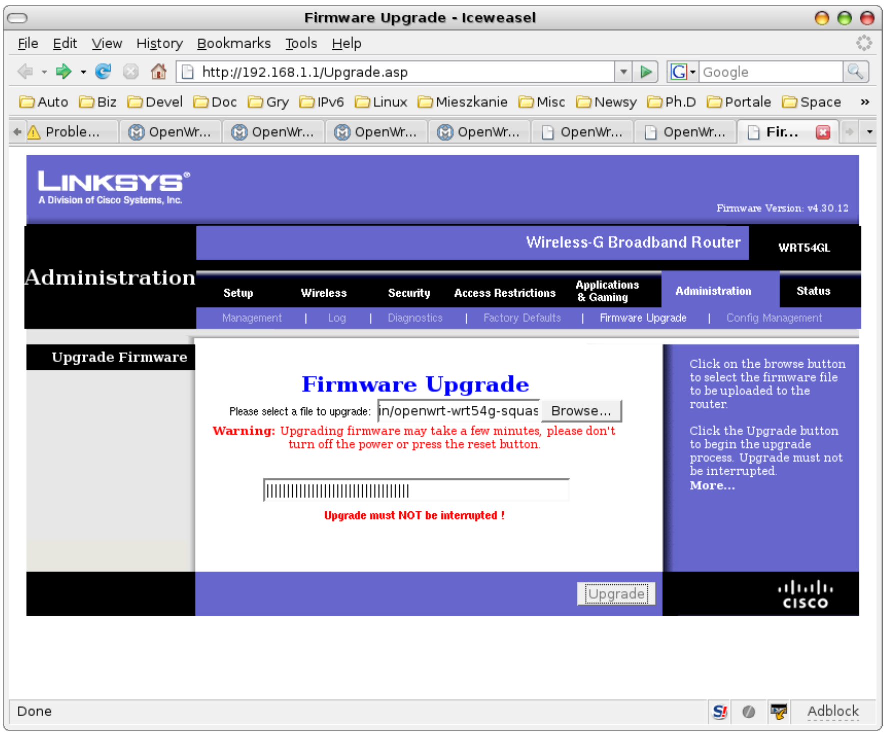
Fig. 2: LinkSys device model check

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.3 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


Firmware Upgrade - Iceweasel
File Edit View History Bookmarks Tools Help
http://192.168.1.1/Upgrade.asp
Google
Auto Biz Devel Doc Gry IPv6 Linux Mieszkanie Misc Newsy Ph.D Portale Space
Proble... OpenWr... OpenWr... OpenWr... OpenWr... OpenWr... OpenWr... Fir...
LINKSYS
A Division of Cisco Systems, Inc.
Firmware Version: v4.30.12
Wireless-G Broadband Router
WRT54GL
Administration
Setup Wireless Security Access Restrictions Applications & Gaming Administration Status
Management | Log | Diagnostics | Factory Defaults | Firmware Upgrade | Config Management
Upgrade Firmware
Firmware Upgrade
Browse...
Upgrade
More...
cisco
Done
Adblock

Please select a file to upgrade: in/openwrt-wrt54g-squas

Warning: Upgrading firmware may take a few minutes, please don't turn off the power or press the reset button.

Upgrade must NOT be interrupted !

Click on the browse button to select the firmware file to be uploaded to the router.

Click the Upgrade button to begin the upgrade process. Upgrade must not be interrupted.

Fig. 3: Firmware upgrade using original web interface

3.3 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.

3.4 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 openwopenwrt-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
```

3.5 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.

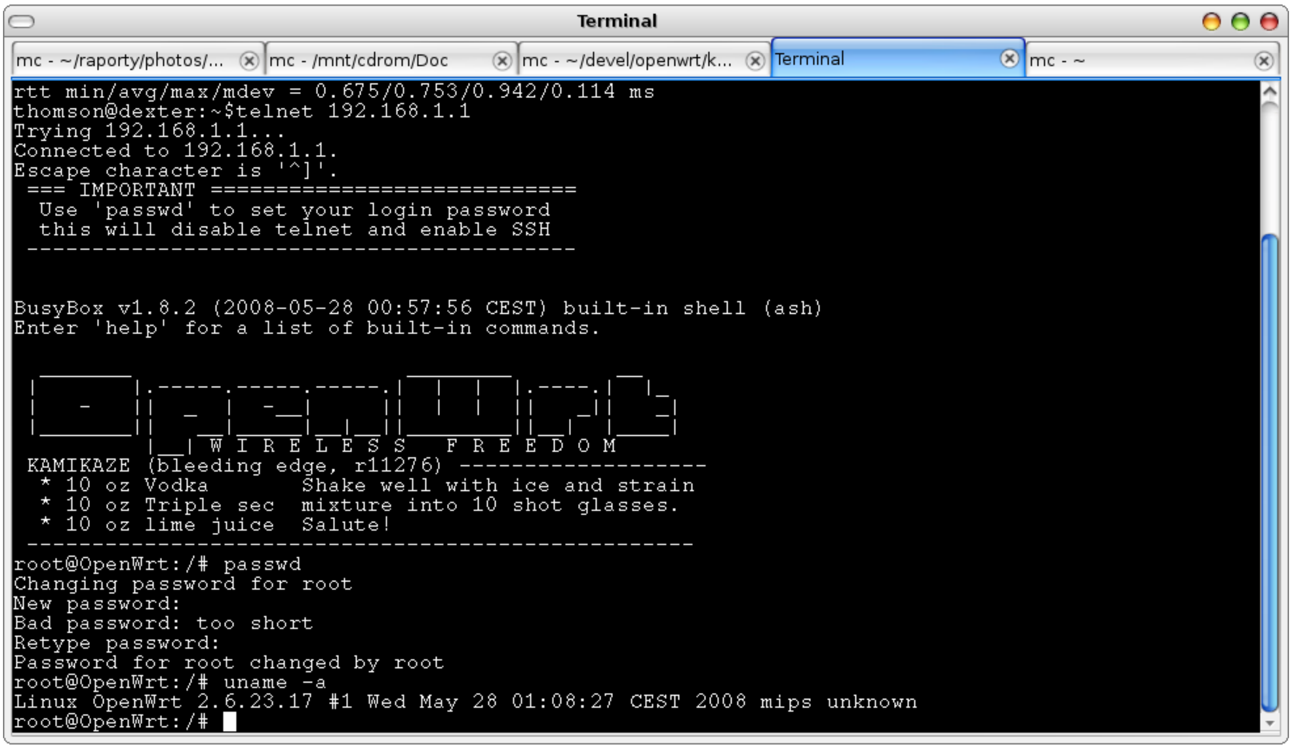
Bootloader may be configured to wait specified period for incoming TFTP packets. However, this waiting phase delays device boot, so vendors often disable this feature. Fortunately, it can be reenabled. To enable boot_wait phase, use following commands from the command-line:

```
nvram set boot_wait=on
nvram 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/

3.6 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.



```
rtt min/avg/max/mdev = 0.675/0.753/0.942/0.114 ms
thomson@dexter:~$telnet 192.168.1.1
Trying 192.168.1.1...
Connected to 192.168.1.1.
Escape character is '^]'.
 === IMPORTANT ============================
  Use 'passwd' to set your login password
  this will disable telnet and enable SSH
 ------------------------------------------


BusyBox v1.8.2 (2008-05-28 00:57:56 CEST) built-in shell (ash)
Enter 'help' for a list of built-in commands.

  _______                     ________        __
 |       |.-----.-----.-----.|  |  |  |.----.|  |_
 |   -   ||  _  |  -__|     ||  |  |  ||   _||   _|
 |_______||   __|_____|__|__||________||__|  |____|
          |__| W I R E L E S S   F R E E D O M
 KAMIKAZE (bleeding edge, r11276) -------------------
  * 10 oz Vodka       Shake well with ice and strain
  * 10 oz Triple sec  mixture into 10 shot glasses.
  * 10 oz lime juice  Salute!
 ---------------------------------------------------
root@OpenWrt:/# passwd
Changing password for root
New password:
Bad password: too short
Retype password:
Password for root changed by root
root@OpenWrt:/# uname -a
Linux OpenWrt 2.6.23.17 #1 Wed May 28 01:08:27 CEST 2008 mips unknown
root@OpenWrt:/#
```

Fig. 4: Linux installed on a LinkSys device

3.7 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 ipkg 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).

3.8 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_2.6.32.27 – 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.

4 PC Linux Installation

This section describes installation process on a regular PC running Linux system.

TODO:

5 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. Ports 1-4 are grouped together and named br-lan. Note that this name have been changed during OpenWRT development, so older tutorial available on the Internat may use different name. To see or modify interface names, see `/etc/config/network`.

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 br-lan
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 br-lan (that is the interface that has default 192.168.1.1 address assigned). Another interface, labeled as WAN, is called eth0.1.

6 Ip46nat module

This section describes ip46nat module, its parameters, fetatures and limitations.

6.1 ip46nat Ovewiew

ip46nat is a linux kernel module. Following parameters are currently supported:

1. **prefixlan** – specifies IPv6 prefix that represents IPv4 LAN. Source addresses originating from LAN will be mapped to this prefix. For traffic outgoing from LAN, this can also be regarded as source prefix. Example value is 2000::. Mandatory parameter.
2. **prefixwan** – specifies IPv6 prefix that represents IPv4 destination. Packets sent from LAN will have its destination addresses mapped to this prefix. Example value is 3000::. Mandatory parameter.
3. **v4addr** – specifies IPv4 address class that is going to be translated. It is used together with v4masklen parameter to form address/netmask form. Example value is 192.168.1.0. Mandatory parameter.
4. **v4masklen** – specifies IPv4 mask length and is expressed in bits. It is used together with v4addr parameter. Example value is 24. Optional parameter. If not specified, default value of 24 is used. Can be <1-32> range.
5. **v4offset** – specifies offset where IPv4 address is mapped in IPv6 address field. For example, if v4address is supposed to be located at the 4 last bytes of IPv6 address, the offset in 96.
6. **debug** – an optional parameter that controls, how much information is provided. Currently 3 values are supported: 0 – be quiet, log only errors, 1 – log matching traffic, 2 – log every packet (slows down operation considerably, use for initial debugging only)

6.2 ip46nat: IPv4-to-IPv6 NAT Usage

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. 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_2.6.25.16-brcm47xx-1_mipsel.ipk
```

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

```
insmod /lib/modules/2.6.25.16/ip46nat.ko prefixlan=2000:: prefixwan=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. For debugging purposes, ip46nat module prints information about every incoming packet and configured pattern.

After module is loaded, it 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.

To see last 10 messages, use following command:

```
dmesg | tail -10
```

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

```
IPv4-IPv6 NAT module loaded: v6prefixp: 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.

6.3 ip46nat: IPv4 to IPv6 NAT Operation

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. Destrination 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 concers, however, as routers are not supposed to intestigate 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] *
```

6.4 ip46nat: IPv6 to IPv4 NAT Operation

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.

6.5 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.

7 Additional services

This section described additional services that can installed and used, how to achieve several aspects of the LinkSys configuration in a manual way. Manual means a static, local configuration performed by locally issued commands or scripts. For automatic, remote, DHCPv6-based configuration, see section 6.

7.1 IPv4 over IPv6 Tunneling

After ip6_tunnel module with all required dependencies is loaded, it is possible to configure IPv4-over-IPv6 tunnel using ip command that is part of the iproute suite. As Ipv4-over-IPv6 tunneling is a very new addition, it required fairly recent version of the iproute code. See Links section for appropriate download links. Make sure to use version that is dedicated to the kernel that is being used.

After kernel is booted and appropriate ip command is available, it is possible to configure IPv4-over-IPv6 tunnel. Assuming that local node has address 2000::1 and remote tunnel endpoint is 2000::abcd, tunnel can be created using following commands:

```
ip -6 tunnel add foo mode ipip6 local 2000::2 remote 2000::100
ip link set up foo
```

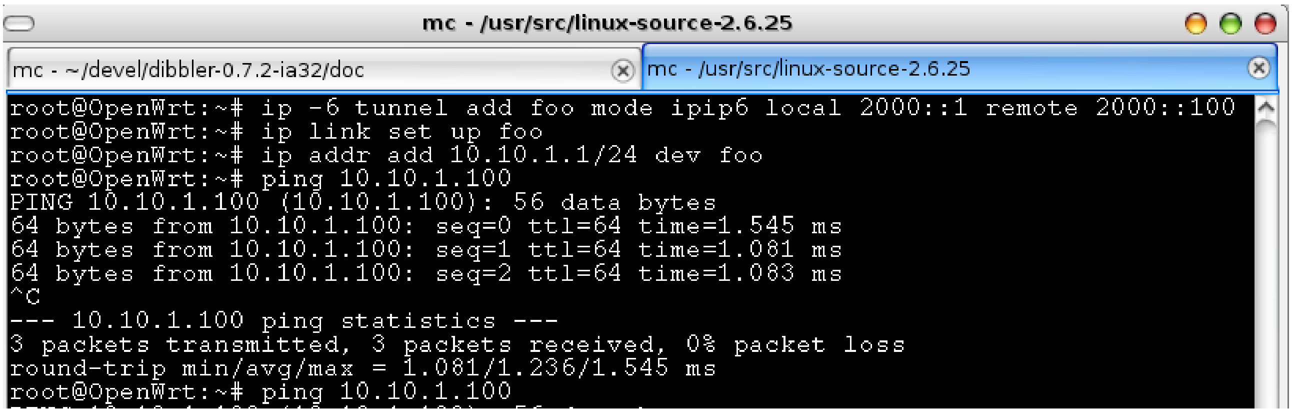
foo is a unique interface name that will be associated with this new tunnel. After tunnel is created, it is possible to add IPv4 address to local endpoint. If IPv4 address is also added to the remote end, IPv4 traffic over IPv6 is possible:

```
ip -6 tunnel add foo mode ipip6 local 2000::1 remote 2000::100
ip link set up foo
```

After that operation, IPv4 addresses may be assigned to the tunnel. If both tunnel endpoints – local (on this node) and remote (on the remote node) have assigned IPv4 addresses, normal IPv4 traffic may be exchanged between them. Routing may be configured to transmit IPv4 traffic via such interface:

```
ip route del default
ip route add default dev foo
```

For example tunnel configuration, see Fig. 5 below.


mc - /usr/src/linux-source-2.6.25
mc - ~/devel/dibbler-0.7.2-ia32/doc
mc - /usr/src/linux-source-2.6.25

```
root@OpenWrt:~# ip -6 tunnel add foo mode ipip6 local 2000::1 remote 2000::100
root@OpenWrt:~# ip link set up foo
root@OpenWrt:~# ip addr add 10.10.1.1/24 dev foo
root@OpenWrt:~# ping 10.10.1.100
PING 10.10.1.100 (10.10.1.100): 56 data bytes
64 bytes from 10.10.1.100: seq=0 ttl=64 time=1.545 ms
64 bytes from 10.10.1.100: seq=1 ttl=64 time=1.081 ms
64 bytes from 10.10.1.100: seq=2 ttl=64 time=1.083 ms
^C
--- 10.10.1.100 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 1.081/1.236/1.545 ms
root@OpenWrt:~# ping 10.10.1.100
```

Fig. 5: Tunnel creation

7.2 DNS proxy

IPv4 based DNS queries from the client can be handled in the same way as any other traffic, i.e. translated to IPv6 by using ip46nat module or tunneled over IPv6 using ip6_tunnel. However, it is also possible to use LinkSys device as a DNS proxy. It will receive incoming IPv4 queries from the client, issue IPv6 based queries to a specified ISP DNS server, handle returning IPv6 based responses and send its contents as a IPv4 based reply to the client. To set up DNS proxy, dnsmasq software may be used. To install it, issue following command:

```
ipkg install dnsmasq_2.45-1_mipsel.ipk
```

After the software is installed, it starts dnsmasq by default. It should be stopped:

```
/etc/init.d/dnsmasq
```

Assuming client's segment (br-lan interface) uses 192.168.1.0/24 pool and the ISP's DNS server is available at the abcd::1 address (eth0.1 interface), following command may be used:

```
dnsmasq -b -d -i br-lan -R --server=2000::1
```

The meaning of the switches is as follows:

- -b – Fake reverse lookups for RFC1918 private IPv4 addresses
- -d – Don't run as a daemon. In the initial runs, it is better to run this command in the console, so any potential issues generated by this app will be discovered.
- -i br-lan – Accept queries received on the br-lan interface
- -R – Ignore /etc/resolv.conf. Without this option, dnsmasq will also use all DNS entries from the /etc/resolv.conf file. To prevent this, -R option should be used.
- --server=2000::1 Forward all queries to the 2000::1 server

For the complete list of switches, please use dnsmasq –help command or see its man page:

http://www.thekelleys.org.uk/dnsmasq/docs/dnsmasq-man.html

Example dnsmasq execution is presented below:

```
root@OpenWrt:~# dnsmasq -b -d -i br-lan -R —server=2000::1
dnsmasq: started, version 2.45 cachesize 150
```

```
dnsmasq: compile time options: IPv6 GNU-getopt ISC-leasefile no-DBus no-
I18N TFTP
dnsmasq: using nameserver 2000::1#53
dnsmasq: read /etc/hosts - 1 addresses
```

7.3 Router Advertisements Daemon

Although not strictly related to this project goals (it is assumed that client's network is purely IPv4), it s possible to provide support for IPv6 in the client's network. Router Advertisement daemon (radvd) should be used for this. To install it, use following command:

```
ipkg install radvd_1.1-1_mipsel.ipk
```

Make sure that /etc/radvd.conf file contains proper cofiguration. Also radvd will refuse to start if IPv6 packet forwarding is disabled.

8 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.

8.1 OpenWRT compilation

ip46nat project uses stable branch (10.03 or “backfire”) 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 24521. It is possible to use later versions, but this possibility was not tested by author. To obtain this revision, issue following command:

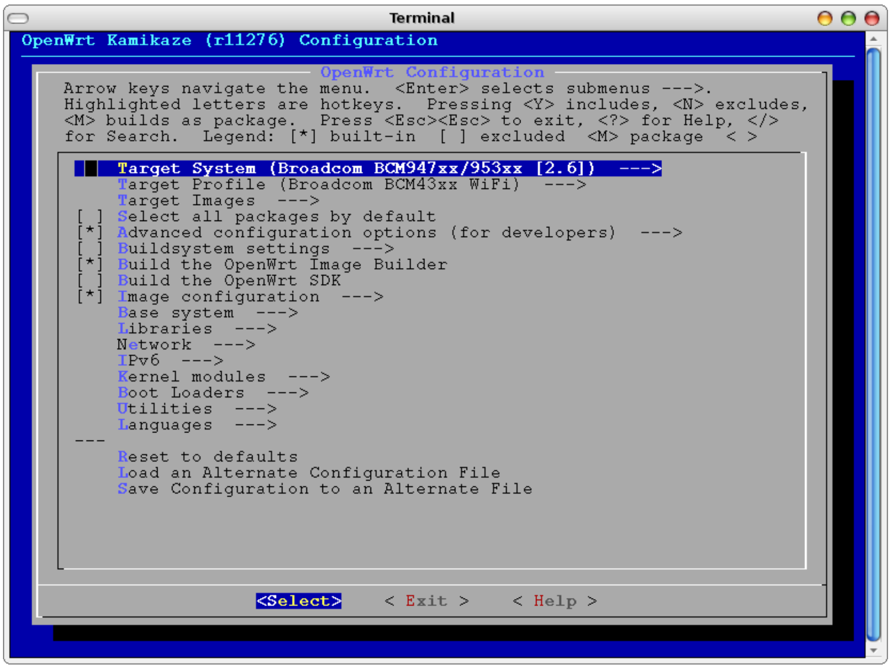
```
svn co -r 24521 svn://svn.openwrt.org/openwrt/branches/backfire
```

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.



```
OpenWrt Kamikaze (r11276) Configuration

                     OpenWrt Configuration
  Arrow keys navigate the menu.  <Enter> selects submenus --->.
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
  <M> builds as package.  Press <Esc><Esc> to exit, <?> for Help, </>
  for Search.  Legend: [*] built-in  [ ] excluded  <M> package  < >

      Target System (Broadcom BCM947xx/953xx [2.6])  --->
      Target Profile (Broadcom BCM43xx WiFi)  --->
      Target Images  --->
  [ ] Select all packages by default
  [*] Advanced configuration options (for developers)  --->
  [ ] Buildsystem settings  --->
  [*] Build the OpenWrt Image Builder
  [ ] Build the OpenWrt SDK
  [*] Image configuration  --->
      Base system  --->
      Libraries  --->
      Network  --->
      IPv6  --->
      Kernel modules  --->
      Boot Loaders  --->
      Utilities  --->
      Languages  --->
  ---
      Reset to defaults
      Load an Alternate Configuration File
      Save Configuration to an Alternate File

              <Select>    < Exit >    < Help >
```

Fig. 6: OpenWRT configuration example

After configuration is complete, type `make` to download requires 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.

8.2 Patching OpenWRT with ip46nat patches

After downloading Backfire source code from OpenWRT SVN repository (see section 8.1), additional modifications has to be introduced to make ip46nat work under OpenWRT environment:

1. Copy necessary kernel patches:
    1. target/linux/brcm47xx/patches-2.6.32/888-ip46nat-2.6.32.27-module.patch
    2. target/linux/brcm47xx/patches-2.6.32/888-ip46nat-2.6.32.27-misc.patch
2. Apply patch that enables building ip46nat modules as a OpenWRT package:

    patch -p0 < openwrt-backfire-r24521-ip46nat.patch

After those steps are complete, there should be extra option available in OpenWRT configuration (accessibly by “make menuconfig” command): Kernel modules → Network Support → kmod-ip46nat. Purpose of each patch is described in Section 8.4.

8.3 ip46nat patches

ip46nat is a Linux kernel module. Although developed with embedded environment in mind, it is not specific for embedded devices. It may be 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 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 2.6.32.27 will be used.
2. Extract kernel using command: **`tar xjvf linux-2.6.32.27.tar.bz2`**
3. Apply ip46nat patch: **`patch -p0 < 888-ip46nat-2.6.32.27-module.patch`**
4. Apply ip46nat miscellaneous fixes: **`patch -p0 < 888-ip46nat-2.6.32.27-misc.patch`**
5. Setup kernel configuration in a normal way: **`make menuconfig`**
6. 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.
7. Save kernel configuration (.config file)
8. Build kernel using following command: **`make`**
9. Build kernel modules using following command: **`make modules`**

10. Continue with normal kernel installation. See numerous installation help documents available here: http://www.google.com/search?q=linux+kernel+installation

Purpose of each patch is described in Section 8.4.

8.4 Patches description

There are 3 patches currently provided.

1. 888-ip46nat-2.6.32.27-module.patch – this patch contains the actual source code of the module. It creates net/ipv6/ip46nat.c file in Linux kernel source directory. Alternatively, instead of applying this patch, the ip46nat.c file can be copied into net/ipv6 directory directly.
2. **`888-ip46nat-2.6.32.27-misc.patch`** – ip46nat modules uses several functions that are not exported (not available to modules) by regular kernel. This patch exports those required functions.
3. openwrt-backfire-r24521-ip46nat.patch – this patch adds definitions for OpenWRT package.

When compiling ip46nat module by hand, only patches 1 and 2 are necessary. When compiling in OpenWRT environment, patch 3 is also required.

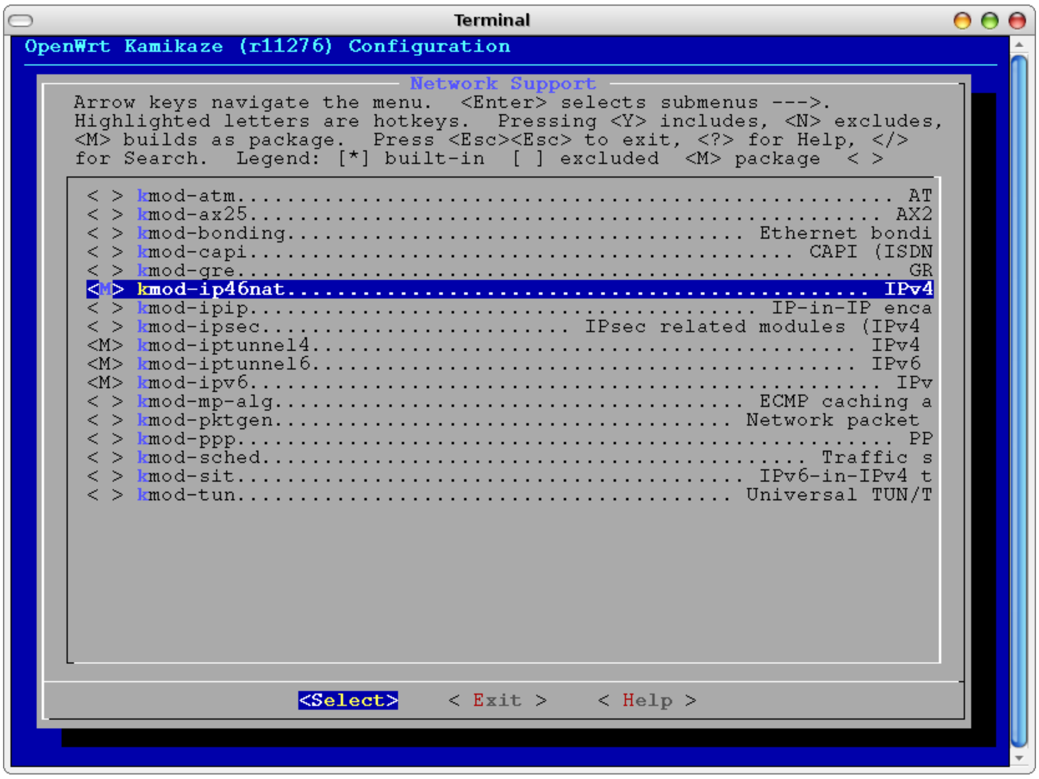
Note: Patches 1 and 2 follow OpenWRT naming convention. The leading number (0-999) designates the order of applying patches. As ip46nat is not considered crucial, it is better to apply it after most of the other patches (with number less than 888) are already applied.

8.5 ip46nat kernel module as an ipk package

To make ip46nat available from the OpenWRT configuration menu, download selected OpenWRT source (see section 7.1) and apply ip46nat-openWRT patch:

```
patch -p0 < ip46nat-openwrt-12386.patch
```

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



```
Terminal
OpenWrt Kamikaze (r11276) Configuration
                         Network Support
  Arrow keys navigate the menu.  <Enter> selects submenus --->.
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
  <M> builds as package.  Press <Esc><Esc> to exit, <?> for Help, </>
  for Search.  Legend: [*] built-in  [ ] excluded  <M> package  < >

    < > kmod-atm.................................................... AT
    < > kmod-ax25................................................... AX2
    < > kmod-bonding.................................. Ethernet bondi
    < > kmod-capi........................................... CAPI (ISDN
    < > kmod-gre.................................................... GR
    <M> kmod-ip46nat............................................... IPv4
    < > kmod-ipip.......................................... IP-in-IP enca
    < > kmod-ipsec........................ IPsec related modules (IPv4
    <M> kmod-iptunnel4........................................... IPv4
    <M> kmod-iptunnel6........................................... IPv6
    <M> kmod-ipv6.................................................. IPv
    < > kmod-mp-alg................................... ECMP caching a
    < > kmod-pktgen.................................. Network packet
    < > kmod-ppp..................................................... PP
    < > kmod-sched............................................ Traffic s
    < > kmod-sit.......................................... IPv6-in-IPv4 t
    < > kmod-tun......................................... Universal TUN/T

                 <Select>    < Exit >    < Help >
```

Fig. 7: ip46nat being selected in OpenWRT configuration menu

8.6 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 constists 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-0.7.2-src.tar.gz
cd dibbler-0.7.2
make server client relay
```

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

Normally, dibbler is released under GNU GPLv2 or later license. However, due to some legal concerns regarding interpretation of the “or later” part, dibbler 0.7.2 was released under GNU GPLv2 only.

8.7 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-0.7.2.tar.gz file.

Note: Those steps are not necessary when using source tree provided on the ip46nat project website.

After those steps are complete, 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 7.1. In the OpenWRT configuration menu, go to Libraries => uclibcxx for uClibc++ and IPv6 => dibbler-server, dibbler-client and dibbler-relay for dibbler packages.

8.8 IPv4-over-IPv6 tunnel modules

Kernels 2.6.25 and newer support IPv4 over IPv6 tunneling. Such tunnels are essential element of DS-Lite infrastructure. To compile this kernel with module supporting IPv4 over IPv6 tunneling, please download and extract Linux kernel sources. Following command may be used for compilation preparation:

```
make menuconfig
```

Go to Networking menu, then Networking Options and select IPv6: IP-in-IPv6 tunnel. Also select all other required modules. After saving changes, type make to begin kernel compilation. There are numerous tutorials and walkthroughs available on the Internet regarding Linux kernel configuration.

Two modules will be created: ip6_tunnel.ko and tunnel6.ko. After booting up the kernel, those modules may be loaded using following command:

```
modprobe ip6_tunnel
```

Compiled modules are supposed to be used in the kernel they were compiled with. They may refuse to work with other kernel versions or even with kernels compiled from the same source version, but with different parameters.

9 Source code

This section discusses internal architecture for developed or extended software.

9.1 Code overview for ip46nat module

ip46nat module was developed from scratch. Since there is one specific use of its operation, all module parameters are specified during module loading. Module loading operation is handled by the hello_init() function. It checks if all required parameters are provided and well formed. After the check is successful, it calls register_handlers() function. Most packet handling in the kernel code is done via handlers. Once packet is received from the network, all handlers for specific packet type are called. In this case, there are 2 handlers defined and used: ipv4_handler and ipv6_handler. Both functions check if the received traffic meet expected criteria, i.e. their address parameters are as expected. Once the packet is checked and is considered to perform NAT, one of 2 separate functions is called: ipv4_send_as_ipv6 or ipv6_send_as_ipv4. In both cases packet header is rewritten, TCP/UDP checksum recalculated and routing for a new packets is selected. If the routing is present, packet is finally sent. Missing routing may means that:

- there is no routing configured for this destination address.
- Packets using 0.0.x.x class as a source address, will not be sent.
- Traffic forwarding is not enabled.

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 IPv4-to-IPv6 NAT (Phase 1) testing

This section provides report from the phase 1 validation. It may also serve as a proof of concept. For the testing purposes, LinkSys will be referred to as DUT – Device Under Test. Most tests were performed in a configuration, where 2 PCs were connected via DUT. This scenario is presented in Fig. 8 below.

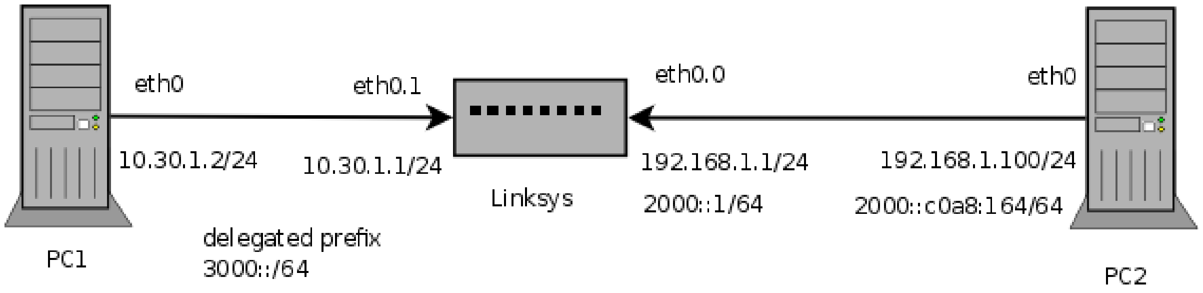


Fig. 8: Test network

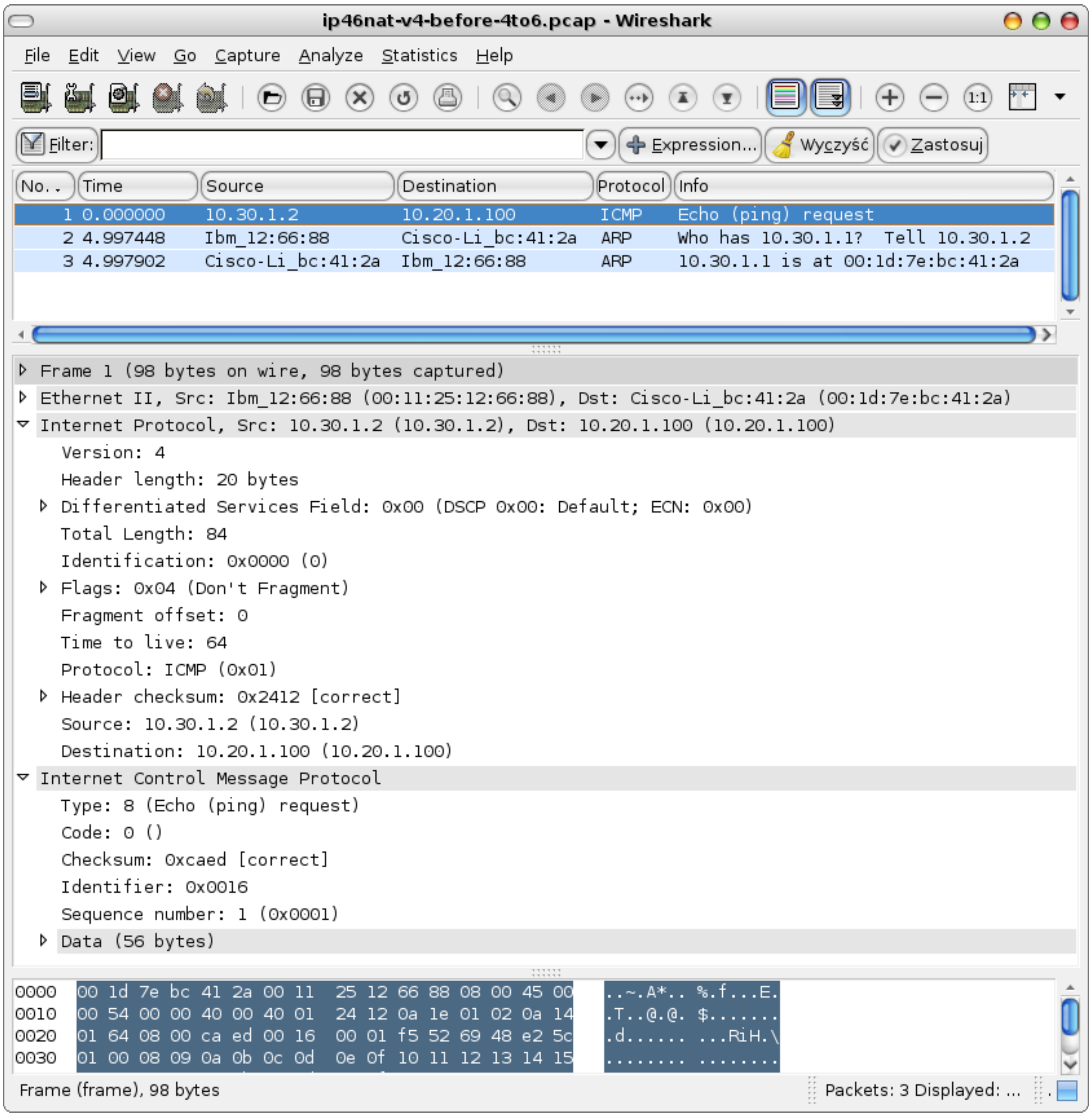
11.1 IPv4 to IPv6 traffic

IPv4 packets were transmitted from the PC1. Single ICMPv4 packets were sent. Fig. 12 contains network captures of such packets. IPv4 packets after translation (they are IPv6 packets) are presented in Fig. 13. 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 that 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.


ip46nat-v4-before-4to6.pcap - Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter:
Expression...
Wyczyść
Zastosuj

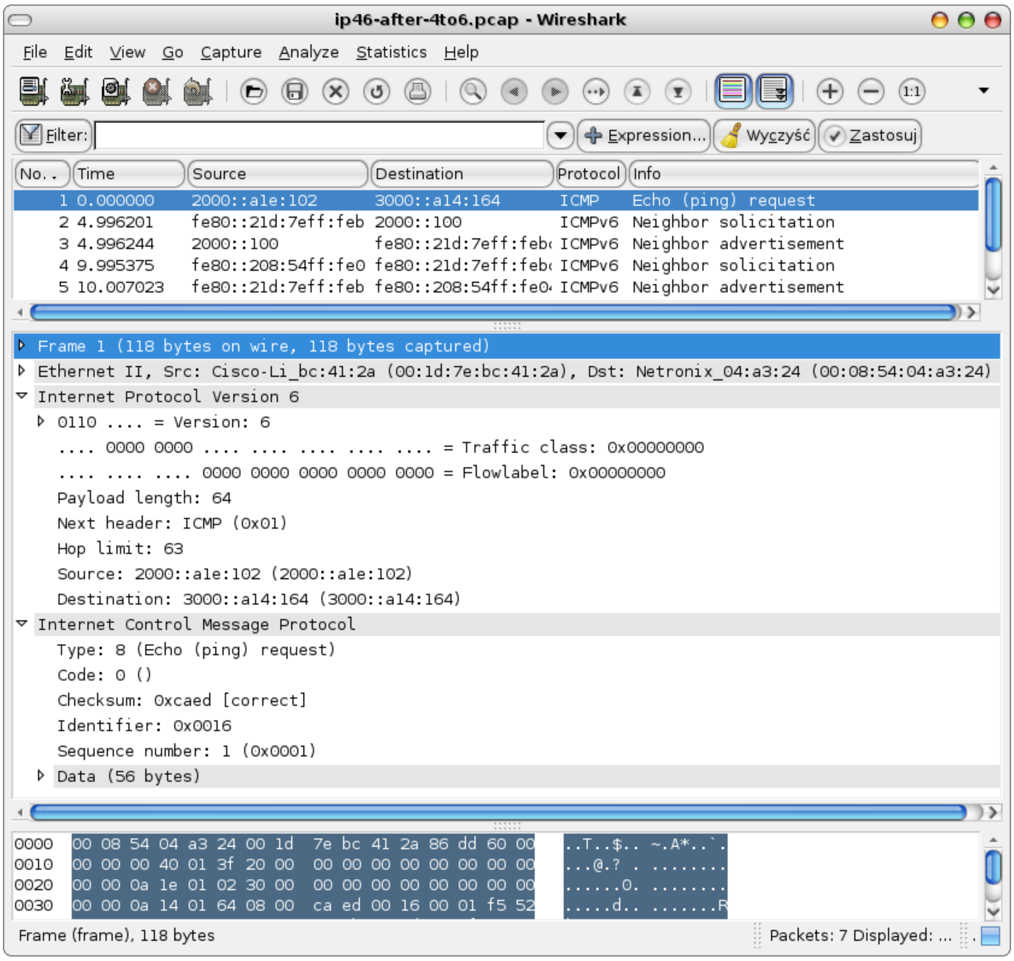
```
No. .  Time       Source            Destination       Protocol  Info
  1 0.000000   10.30.1.2         10.20.1.100        ICMP     Echo (ping) request
  2 4.997448   Ibm_12:66:88      Cisco-Li_bc:41:2a  ARP      Who has 10.30.1.1?  Tell 10.30.1.2
  3 4.997902   Cisco-Li_bc:41:2a Ibm_12:66:88       ARP      10.30.1.1 is at 00:1d:7e:bc:41:2a
```

```
▷ Frame 1 (98 bytes on wire, 98 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: 10.30.1.2 (10.30.1.2), Dst: 10.20.1.100 (10.20.1.100)
    Version: 4
    Header length: 20 bytes
  ▷ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
    Total Length: 84
    Identification: 0x0000 (0)
  ▷ Flags: 0x04 (Don't Fragment)
    Fragment offset: 0
    Time to live: 64
    Protocol: ICMP (0x01)
  ▷ Header checksum: 0x2412 [correct]
    Source: 10.30.1.2 (10.30.1.2)
    Destination: 10.20.1.100 (10.20.1.100)
▽ Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0 ()
    Checksum: 0xcaed [correct]
    Identifier: 0x0016
    Sequence number: 1 (0x0001)
  ▷ Data (56 bytes)
```

```
0000  00 1d 7e bc 41 2a 00 11  25 12 66 88 08 00 45 00   ..~.A*.. %.f...E.
0010  00 54 00 00 40 00 40 01  24 12 0a 1e 01 02 0a 14   .T..@.@. $.......
0020  01 64 08 00 ca ed 00 16  00 01 f5 52 69 48 e2 5c   .d...... ...RiH.\
0030  01 00 08 09 0a 0b 0c 0d  0e 0f 10 11 12 13 14 15   ........ ........
```

Frame (frame), 98 bytes   Packets: 3 Displayed: ...

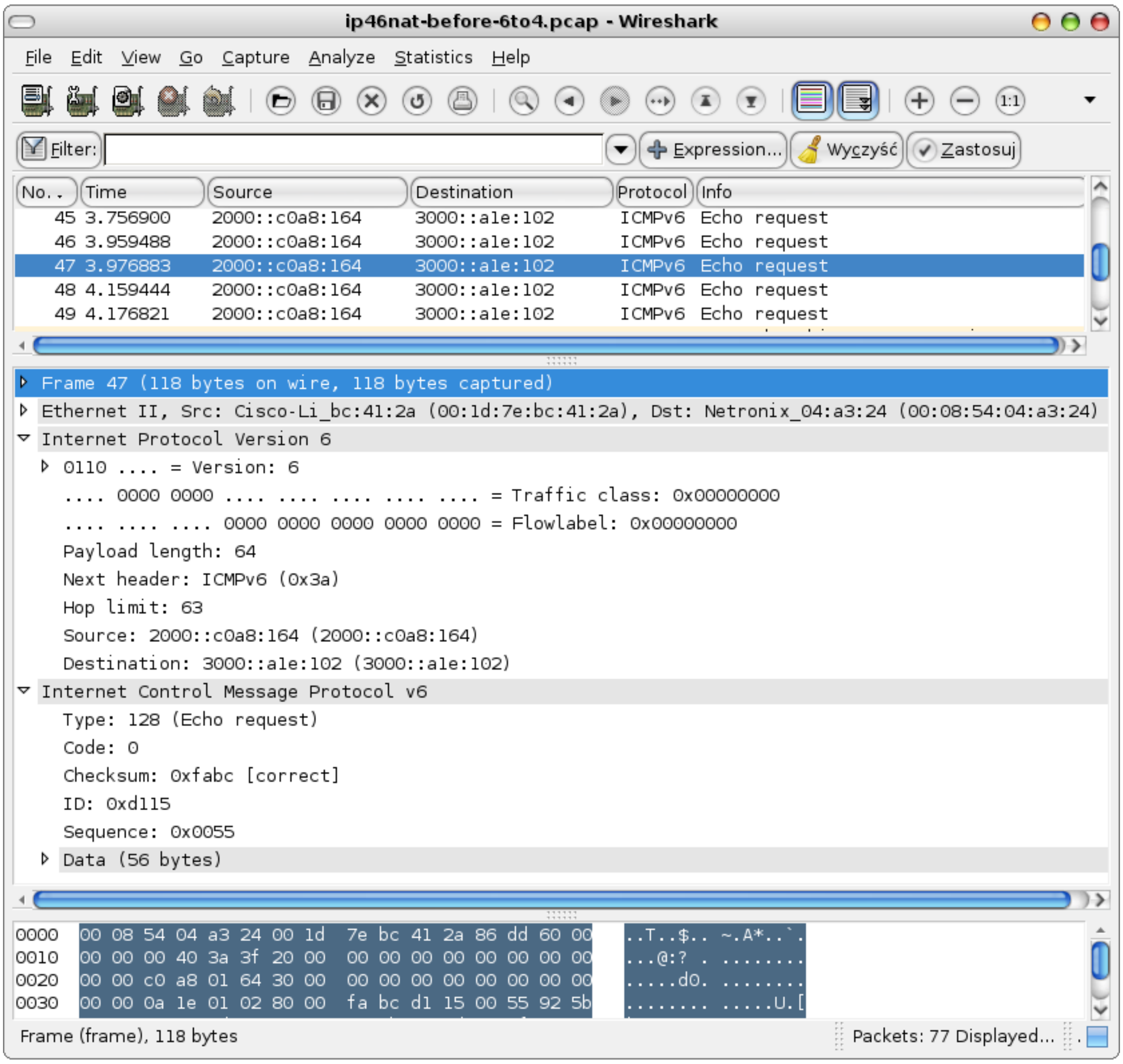
Fig. 9: IPv4 packet before translation


ip46-after-4to6.pcap - Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter:
Expression...
Wyczyść
Zastosuj

```
No. . Time       Source               Destination          Protocol Info
    1 0.000000   2000::a1e:102        3000::a14:164        ICMP     Echo (ping) request
    2 4.996201   fe80::21d:7eff:feb   2000::100            ICMPv6   Neighbor solicitation
    3 4.996244   2000::100            fe80::21d:7eff:feb   ICMPv6   Neighbor advertisement
    4 9.995375   fe80::208:54ff:fe0   fe80::21d:7eff:feb   ICMPv6   Neighbor solicitation
    5 10.007023  fe80::21d:7eff:feb   fe80::208:54ff:fe0   ICMPv6   Neighbor advertisement

Frame 1 (118 bytes on wire, 118 bytes captured)
Ethernet II, Src: Cisco-Li_bc:41:2a (00:1d:7e:bc:41:2a), Dst: Netronix_04:a3:24 (00:08:54:04:a3:24)
Internet Protocol Version 6
  0110 .... = Version: 6
  .... 0000 0000 .... .... .... .... .... = Traffic class: 0x00000000
  .... .... .... 0000 0000 0000 0000 0000 = Flowlabel: 0x00000000
  Payload length: 64
  Next header: ICMP (0x01)
  Hop limit: 63
  Source: 2000::a1e:102 (2000::a1e:102)
  Destination: 3000::a14:164 (3000::a14:164)
Internet Control Message Protocol
  Type: 8 (Echo (ping) request)
  Code: 0 ()
  Checksum: 0xcaed [correct]
  Identifier: 0x0016
  Sequence number: 1 (0x0001)
  Data (56 bytes)

0000  00 08 54 04 a3 24 00 1d  7e bc 41 2a 86 dd 60 00   ..T..$.. ~.A*..`.
0010  00 00 00 40 01 3f 20 00  00 00 00 00 00 00 00 00   ...@.? . ........
0020  00 00 0a 1e 01 02 30 00  00 00 00 00 00 00 00 00   ......0. ........
0030  00 00 0a 14 01 64 08 00  ca ed 00 16 00 01 f5 52   .....d.. .......R

Frame (frame), 118 bytes                    Packets: 7 Displayed: ...
```

Fig. 10: IPv4 packets after IPv4-to-IPv6 translation

11.2 IPv6 to IPv4 traffic



ip46nat-before-6to4.pcap - Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter:
Expression...
Wyczyść
Zastosuj

```
No. .  Time       Source            Destination       Protocol  Info
  45 3.756900    2000::c0a8:164    3000::a1e:102     ICMPv6    Echo request
  46 3.959488    2000::c0a8:164    3000::a1e:102     ICMPv6    Echo request
  47 3.976883    2000::c0a8:164    3000::a1e:102     ICMPv6    Echo request
  48 4.159444    2000::c0a8:164    3000::a1e:102     ICMPv6    Echo request
  49 4.176821    2000::c0a8:164    3000::a1e:102     ICMPv6    Echo request
```

```
▷ Frame 47 (118 bytes on wire, 118 bytes captured)
▷ Ethernet II, Src: Cisco-Li_bc:41:2a (00:1d:7e:bc:41:2a), Dst: Netronix_04:a3:24 (00:08:54:04:a3:24)
▽ Internet Protocol Version 6
  ▷ 0110 .... = Version: 6
    .... 0000 0000 .... .... .... .... .... = Traffic class: 0x00000000
    .... .... .... 0000 0000 0000 0000 0000 = Flowlabel: 0x00000000
    Payload length: 64
    Next header: ICMPv6 (0x3a)
    Hop limit: 63
    Source: 2000::c0a8:164 (2000::c0a8:164)
    Destination: 3000::a1e:102 (3000::a1e:102)
▽ Internet Control Message Protocol v6
    Type: 128 (Echo request)
    Code: 0
    Checksum: 0xfabc [correct]
    ID: 0xd115
    Sequence: 0x0055
  ▷ Data (56 bytes)
```

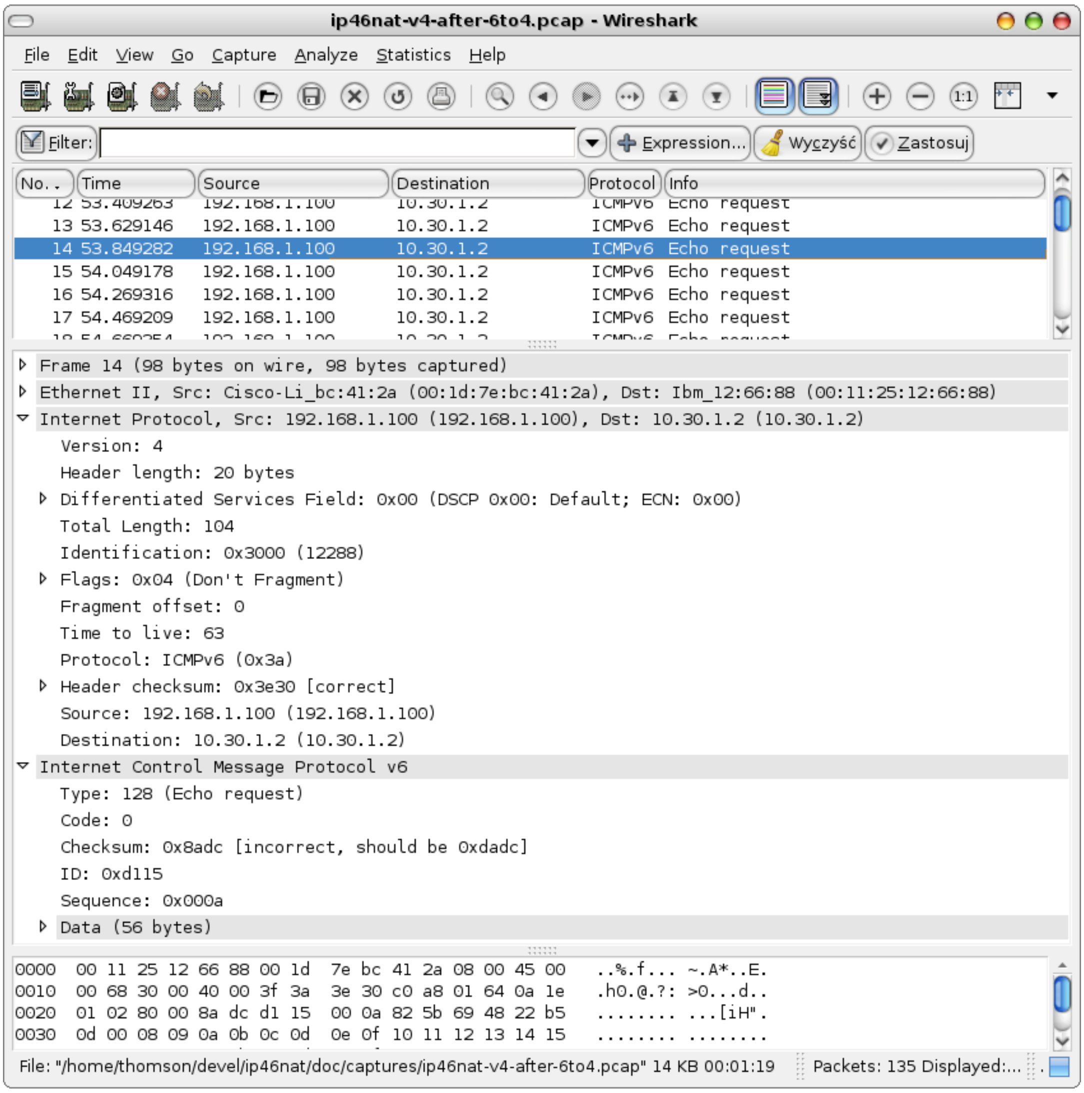
```
0000  00 08 54 04 a3 24 00 1d  7e bc 41 2a 86 dd 60 00   ..T..$.. ~.A*..`.
0010  00 00 00 40 3a 3f 20 00  00 00 00 00 00 00 00 00   ...@:? . ........
0020  00 00 c0 a8 01 64 30 00  00 00 00 00 00 00 00 00   .....d0. ........
0030  00 00 0a 1e 01 02 80 00  fa bc d1 15 00 55 92 5b   ........ .....U.[
```

```
Frame (frame), 118 bytes                    Packets: 77 Displayed...
```

Fig. 11: IPv6 packet before reverse translation

Similar tests were performed for IPv6 traffic being translated to IPv4. See Fig. 14 for packets before translation and Fig. 15 for packets after translation.

Please note that to have packets transmitted, both its source and destination address should be legitimate, i.e. routing should be configured for such packets. 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 accordingly. There is also separate statistic for this condition.


ip46nat-v4-after-6to4.pcap - Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter:
Expression...
Wyczyść
Zastosuj

```
No. . Time        Source          Destination    Protocol Info
   12 53.409263   192.168.1.100   10.30.1.2      ICMPv6   Echo request
   13 53.629146   192.168.1.100   10.30.1.2      ICMPv6   Echo request
   14 53.849282   192.168.1.100   10.30.1.2      ICMPv6   Echo request
   15 54.049178   192.168.1.100   10.30.1.2      ICMPv6   Echo request
   16 54.269316   192.168.1.100   10.30.1.2      ICMPv6   Echo request
   17 54.469209   192.168.1.100   10.30.1.2      ICMPv6   Echo request
   18 54.669254   192.168.1.100   10.30.1.2      ICMPv6   Echo request
```

```
▷ Frame 14 (98 bytes on wire, 98 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: 192.168.1.100 (192.168.1.100), Dst: 10.30.1.2 (10.30.1.2)
    Version: 4
    Header length: 20 bytes
  ▷ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
    Total Length: 104
    Identification: 0x3000 (12288)
  ▷ Flags: 0x04 (Don't Fragment)
    Fragment offset: 0
    Time to live: 63
    Protocol: ICMPv6 (0x3a)
  ▷ Header checksum: 0x3e30 [correct]
    Source: 192.168.1.100 (192.168.1.100)
    Destination: 10.30.1.2 (10.30.1.2)
▽ Internet Control Message Protocol v6
    Type: 128 (Echo request)
    Code: 0
    Checksum: 0x8adc [incorrect, should be 0xdadc]
    ID: 0xd115
    Sequence: 0x000a
  ▷ Data (56 bytes)
```

```
0000  00 11 25 12 66 88 00 1d  7e bc 41 2a 08 00 45 00   ..%.f... ~.A*..E.
0010  00 68 30 00 40 00 3f 3a  3e 30 c0 a8 01 64 0a 1e   .h0.@.?: >0...d..
0020  01 02 80 00 8a dc d1 15  00 0a 82 5b 69 48 22 b5   ........ ...[iH".
0030  0d 00 08 09 0a 0b 0c 0d  0e 0f 10 11 12 13 14 15   ........ ........
```

File: "/home/thomson/devel/ip46nat/doc/captures/ip46nat-v4-after-6to4.pcap" 14 KB 00:01:19 Packets: 135 Displayed:...

Fig. 12: IPv6 packet after reverse translation

11.3 Firewall configuration

Due to internal Linux kernel architecture, after ip46nat module processes the packet, it is not possible to forbid normal Linux IPv4 routing procedures to continue. To avoid packet duplication, IPv4 packets that are to be translated, should be dropped in a postprocessing phase (they will be sent as IPv6 only). Assuming network configuration as in Fig. 11 (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
```

11.4 Negative testing

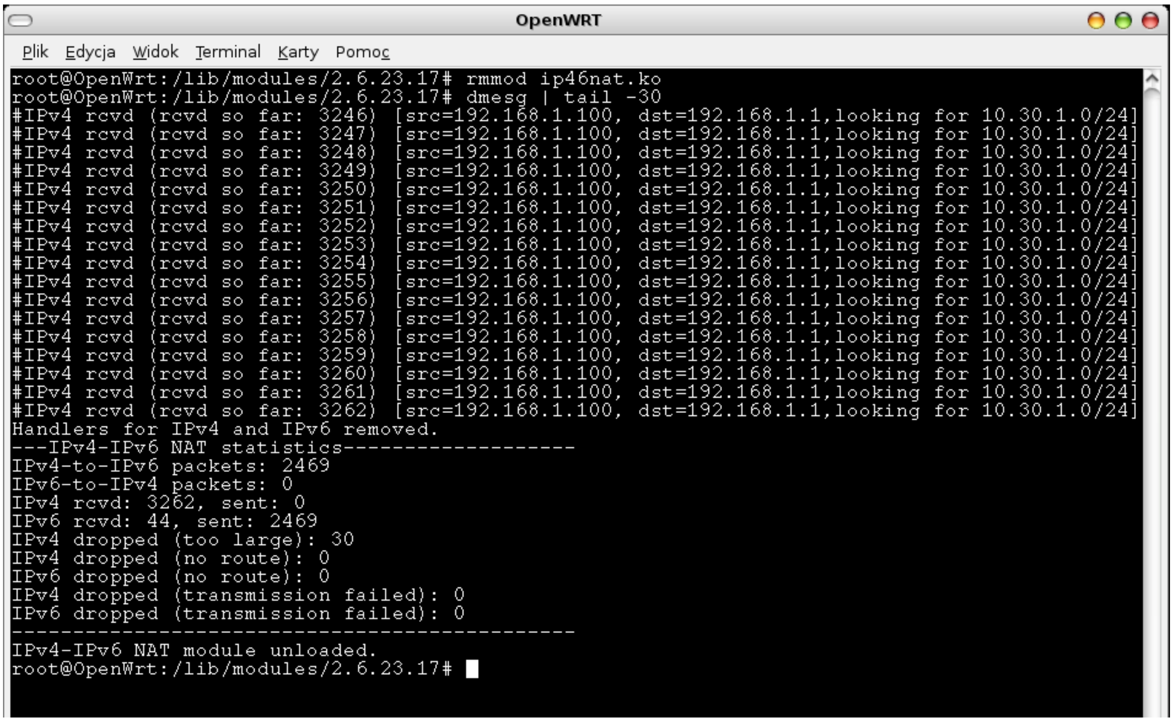
Too large IPv4 packets were sent to confirm that no buffer overflow occurs. Packets were dropped as expected and appropriate statistic was increased. Numerous not-matching criteria IPv4 and IPv6 packets were sent. None of them was ever translated. For easier debugging, ip46nat module prints every packet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

11.5 Performance testing

To make sure that the device is being able to handle heavy traffic, biggest possible (1480) packets were transmitted in an continuous manner. After prolonged traffic handle the device behaved normally. See Fig. 14 for statistics after such testing.

11.6 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. 13: Statistics after some traffic

11.7 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. Such maturity level is adequate for the intended status of the product – a research prototype, intended for laboratory experiments.

12 IPv4 over IPv6 tunneling (Phase 2) testing

IPv4 over IPv6 validation can be split into 2 parts: traffic validation and remote configuration validation. For a realistic test environment, 2 PCs were connected to the DUT: PC1 is a end-user's equipment (Vista PC). PC2 is a operator's server (Linux PC). This architecture is presented in Fig. 14. PC2 also acts as a web server. It has configured IPv4-over-IPv6 tunnel. There is extra IPv4 address (192.168.2.100) assigned to the tunnel.

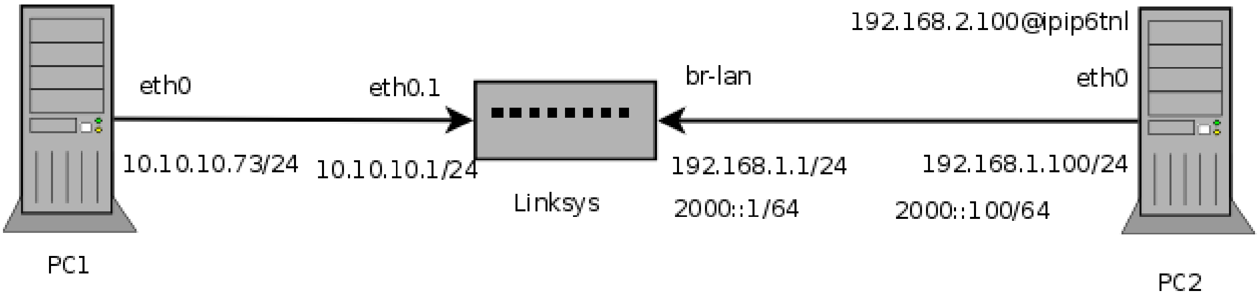


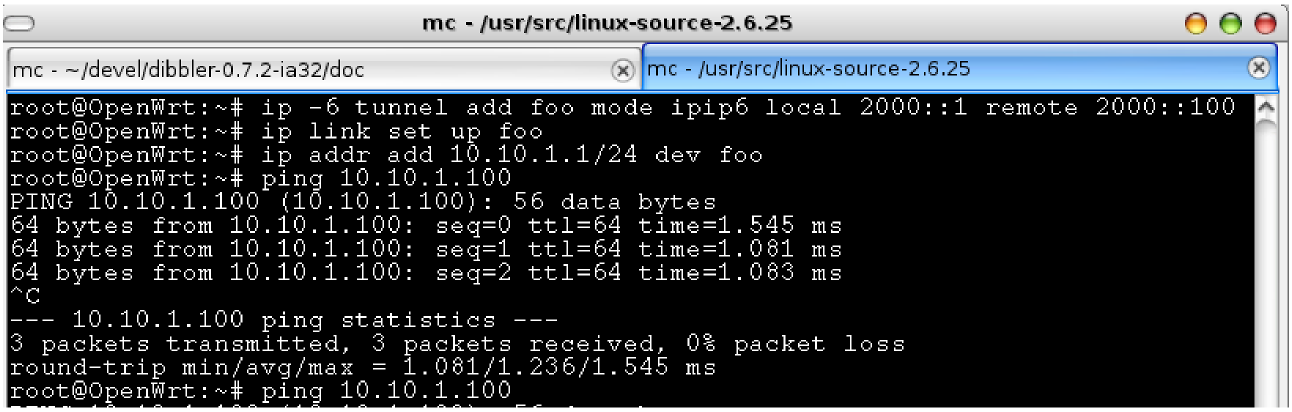
Fig. 14: IPv4 over IPv6 testing infrastructure

12.1 Traffic validation

Goal of this test was to validate if DUT is handling IPv4-over-IPv6 traffic properly. Tunnel on DUT was configured according to description in section 7.6.

It receives incoming IPv4 packets on the eth0.1 from the 10.10.10.0/24 class. That traffic is encapsulated in IPv6 packets and sent via br-lan interface. Returning IPv4 packets are received on the br-lan. To simulate real destination, PC2 has extra IPv4 address (192.168.2.100) assigned to the ipip6tnl interface (local name for the tunnel interface).

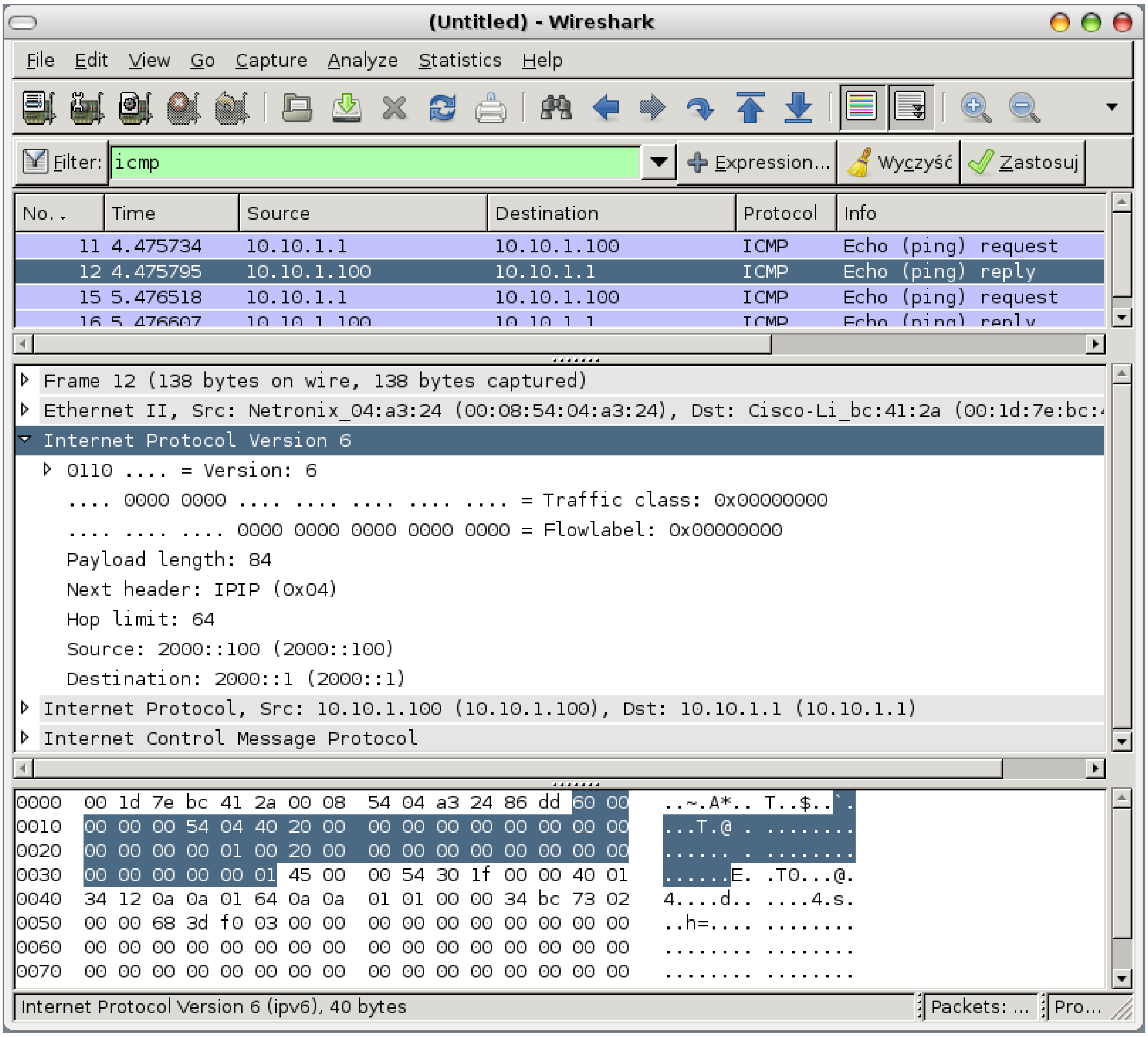
Tests started with simple ICMP messages. Pings were sent from the DUT to PC2. This exchange is presented in Fig. 15.


mc - /usr/src/linux-source-2.6.25
mc - ~/devel/dibbler-0.7.2-ia32/doc
mc - /usr/src/linux-source-2.6.25

```
root@OpenWrt:~# ip -6 tunnel add foo mode ipip6 local 2000::1 remote 2000::100
root@OpenWrt:~# ip link set up foo
root@OpenWrt:~# ip addr add 10.10.1.1/24 dev foo
root@OpenWrt:~# ping 10.10.1.100
PING 10.10.1.100 (10.10.1.100): 56 data bytes
64 bytes from 10.10.1.100: seq=0 ttl=64 time=1.545 ms
64 bytes from 10.10.1.100: seq=1 ttl=64 time=1.081 ms
64 bytes from 10.10.1.100: seq=2 ttl=64 time=1.083 ms
^C
--- 10.10.1.100 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 1.081/1.236/1.545 ms
root@OpenWrt:~# ping 10.10.1.100
```

Fig. 15: ping from DUT via IPv4 over IPv6

The same ICMP echo/reply interaction was captured using Wireshark tool. That is presented in Fig. 16 below.



No. .	Time	Source	Destination	Protocol	Info
11	4.475734	10.10.1.1	10.10.1.100	ICMP	Echo (ping) request
12	4.475795	10.10.1.100	10.10.1.1	ICMP	Echo (ping) reply
15	5.476518	10.10.1.1	10.10.1.100	ICMP	Echo (ping) request
16	5.476607	10.10.1.100	10.10.1.1	ICMP	Echo (ping) reply

```
▷ Frame 12 (138 bytes on wire, 138 bytes captured)
▷ Ethernet II, Src: Netronix_04:a3:24 (00:08:54:04:a3:24), Dst: Cisco-Li_bc:41:2a (00:1d:7e:bc:4
▽ Internet Protocol Version 6
  ▷ 0110 .... = Version: 6
    .... 0000 0000 .... .... .... .... .... = Traffic class: 0x00000000
    .... .... .... 0000 0000 0000 0000 0000 = Flowlabel: 0x00000000
    Payload length: 84
    Next header: IPIP (0x04)
    Hop limit: 64
    Source: 2000::100 (2000::100)
    Destination: 2000::1 (2000::1)
▷ Internet Protocol, Src: 10.10.1.100 (10.10.1.100), Dst: 10.10.1.1 (10.10.1.1)
▷ Internet Control Message Protocol
```

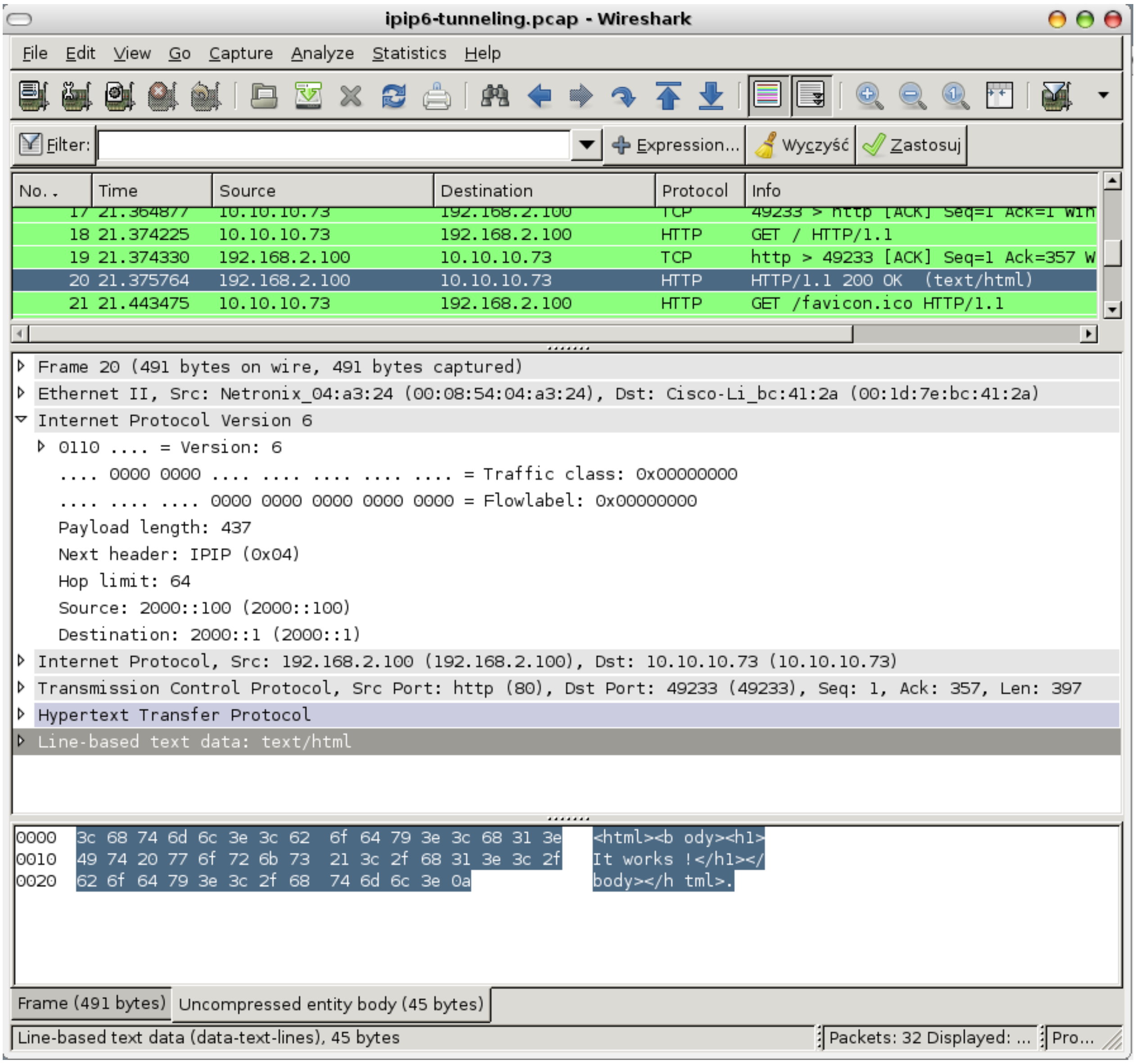
```
0000  00 1d 7e bc 41 2a 00 08  54 04 a3 24 86 dd 60 00   ..~.A*.. T..$..`.
0010  00 00 00 54 04 40 20 00  00 00 00 00 00 00 00 00   ...T.@ . ........
0020  00 00 00 00 01 00 20 00  00 00 00 00 00 00 00 00   ...... . ........
0030  00 00 00 00 00 01 45 00  00 54 30 1f 00 00 40 01   ......E. .T0...@.
0040  34 12 0a 0a 01 64 0a 0a  01 01 00 00 34 bc 73 02   4....d.. ....4.s.
0050  00 00 68 3d f0 03 00 00  00 00 00 00 00 00 00 00   ..h=.... ........
0060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
```

Internet Protocol Version 6 (ipv6), 40 bytes | Packets: ... | Pro...

Fig. 16: ICMP transmission observed on Wireshark

After basic functionality was confirmed, web server was started on PC2. PC1 has been configured to use DUT as a default gateway. Also DUT has its default route configured via tunnel. It is possible that client (PC1, 10.10.10.73) is able to reach its destination (192.168.2.100), as confirmed with the ping interface. After starting web browser on the PC1, it was possible to connect to the PC2 and download web page content. This transmission was captured using Wireshark. That traffic is depicted in Fig. 17 below.

That scenario concluded traffic handling validation.


ipip6-tunneling.pcap - Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter:
Expression...
Wyczyść
Zastosuj

```
No. .  Time         Source          Destination     Protocol  Info
    17 21.364877    10.10.10.73     192.168.2.100   TCP       49233 > http [ACK] Seq=1 Ack=1 Win
    18 21.374225    10.10.10.73     192.168.2.100   HTTP      GET / HTTP/1.1
    19 21.374330    192.168.2.100   10.10.10.73     TCP       http > 49233 [ACK] Seq=1 Ack=357 W
    20 21.375764    192.168.2.100   10.10.10.73     HTTP      HTTP/1.1 200 OK  (text/html)
    21 21.443475    10.10.10.73     192.168.2.100   HTTP      GET /favicon.ico HTTP/1.1
```

```
▷ Frame 20 (491 bytes on wire, 491 bytes captured)
▷ Ethernet II, Src: Netronix_04:a3:24 (00:08:54:04:a3:24), Dst: Cisco-Li_bc:41:2a (00:1d:7e:bc:41:2a)
▽ Internet Protocol Version 6
  ▷ 0110 .... = Version: 6
    .... 0000 0000 .... .... .... .... .... = Traffic class: 0x00000000
    .... .... .... 0000 0000 0000 0000 0000 = Flowlabel: 0x00000000
    Payload length: 437
    Next header: IPIP (0x04)
    Hop limit: 64
    Source: 2000::100 (2000::100)
    Destination: 2000::1 (2000::1)
▷ Internet Protocol, Src: 192.168.2.100 (192.168.2.100), Dst: 10.10.10.73 (10.10.10.73)
▷ Transmission Control Protocol, Src Port: http (80), Dst Port: 49233 (49233), Seq: 1, Ack: 357, Len: 397
▷ Hypertext Transfer Protocol
▷ Line-based text data: text/html
```

```
0000  3c 68 74 6d 6c 3e 3c 62  6f 64 79 3e 3c 68 31 3e   <html><b ody><h1>
0010  49 74 20 77 6f 72 6b 73  21 3c 2f 68 31 3e 3c 2f   It works !</h1></
0020  62 6f 64 79 3e 3c 2f 68  74 6d 6c 3e 0a            body></h tml>.
```

```
Line-based text data (data-text-lines), 45 bytes | Packets: 32 Displayed: ... | Pro...
```

Fig. 17: HTTP traffic over IPv4 over IPv6

13 Links

Following links are recommended reading:

- http://iip.net.pl/ - the homepage of Future Internet Engineering project.
- 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.
- http://www.thelinuxblog.com/linux-man-pages/1/ash – ash (bash-like shell replacement) manual page
- http://www.linuxfoundation.org/en/Net:Iproute2 – iproute homepage
- http://devresources.linux-foundation.org/dev/iproute2/download/ - iproute source code download

14 Contact

Author of this project can be reached using e-mail tomasz.mrugalski@eti.pg.gda.pl.

15 Acknowledgement

This project is been partially supported by the Polish Ministry of Science and Higher Education under the European Regional Development Fund, Grant No. POIG.01.01.02-00-045/09-00 Future Internet Engineering.