

Documentation

Implementing IPv4 gateways for native IPv6 machines

Feczák Szabolcs (feczo@koli.kando.hu)

Contents

1	Preface	1
2	Theory	2
3	Solutions	4
3.1	Summary of searching for the available implementations . . .	4
3.2	Linux	4
3.3	*BSD	5
3.3.1	Totd	6
3.3.2	Faithd	7
3.3.3	NAT-PT	8
4	Testing	10
A	URL list	12
B	Daemon configuration files	13
C	FreeBSD kernel configuration	15
D	Installation checkpoints	18

Chapter 1

Preface

The aim of this documentation to give an overview of the implementations currently ¹ available for establishing connection from native IPv6 machines ² to IPv4 machines.

¹December 18. 2001.

²without having dual-stack installed

Chapter 2

Theory

Theories can be gathered from IETF in the RFC state documents numbered

- RFC3142 An IPv6-to-IPv4 Transport Relay Translator
- RFC2766 Network Address Translation - Protocol Translation
- RFC3089 A SOCKS-based IPv6/IPv4 Gateway Mechanism

TRT and NAT-PT seems to be good later for everyday use. SOCKS based solutions have a big disadvantage: On the client side all applications must be modified or replaced, which need to be used with the gateway. There is a possibility to do a trick and use a wrapper script instead, but it does not always work. On the other hand SOCKS based IPv6->IPv4 gateway is not so much more, but a normal SOCKS proxy equipped with the capability of communicating with IPv6 clients. In my opinion this solution is only suitable if the organisation already has an IPv4 SOCKS system, in this case only small changes are required to adopt the IPv6 only protocol stack on the internal network. Otherwise it is better to set up normal proxy daemons capable to listen on IPv6 for each protocol, which needs to be used by the IPv6 clients. Typically is not much more, as web and E-mail access. In this case the client side can be unchanged (for example with transparent proxy) or only slight modifications are required.

With NAT-PT the communication is realized by use of a dedicated device that does the translation between IPv4 and IPv6 addresses and keeps state during the time of the session. The NAT-PT device also includes an application layer gateway to make translation possible between IPv4 and IPv6 DNS requests and answers.

Applicability scope:	domain
IPv4 requirements:	none
IPv4 address requirements:	>=1 per site

IPv6 requirements:	none
IPv6 address requirements:	none
Host requirements:	IPv6 stack
Router requirements:	none (but router might be the NAT-PT device)
NAT impact:	2 (or more) levels of translation.
Other requirements:	DNS in IPv6 network, ALG for application wich use literal addresses

TRT enables direct communication between IPv6 hosts and IPv4 hosts. This mechanism is somewhat similar to NAT-PT, but does the translation at the transport layer, not at the IP layer. There should be a dedicated router at a site to translate UDP,TCP/IPv6 to UDP,TCP/IPv4 and vice versa. also, there should be a DNS server which can map IPv4 addresses to IPv6 addresses. No modification is necessary for IPv6 hosts and IPv4 hosts. For scalability, multiple dedicated boxes can be installed for a site with multiple dummy IPv6 prefixes. UDP traffic can be relayed by the same technique as that of TCP.

Applicability scope:	domain
IPv4 requirements:	none
IPv4 address requirements:	1 per site
IPv6 requirements:	none
IPv6 address requirements:	One dummy prefix out of the site address
Host requirements:	none
Router requirements:	none, but an intermediate device must be a transport relay translator
NAT impact:	May depend on the application.
Other requirements:	DNS servers which can map IPv4 addresses to IPv6 addresses

Implementing both TRT and NAT-PT needs to have a mechanism which equips the clients with the possibility to tell the gateway, which IPv4 address do they want to talk to. Of course first they connect to the domain name server, if the address should be translated to IP address. At this point we need to forward them to our "fake" nameserver. This nameserver acts normally if the reply from the authority contains "AAAA" record(s), but if only "A" record can be found, then it uses a predefined prefix and adds the resolved Ipv4 address translated into its hexadecimal value. For example normally www.fht-esslingen.de has the IPv4 address of 134.108.34.3. If we have a prefix 3ffe:400:3d0:1000:: with 64 bit length, then the calculated IPv6 address will be 3ffe:400:3d0:1000::866c:2203.

Chapter 3

Solutions

3.1 Summary of searching for the available implementations

Searching through the world wide web everybody can realise, that although the basic IPv6 protocol stack is available for most of the well known operating systems, but the implementation of advanced features are still lacking or incomplete. It seems to be, that the mainstream of the IPv6 development fortunately happens in the open source sector by the USAGI, WIDE, KAME and TAHI projects.

3.2 Linux

First we will take a closer look on the nowadays more and more popular Linux operating system. To archive the most up to date state of the Linux IPv6 stack we need to fetch the snapshot ¹ from USAGI:

<ftp://ftp.linux-ipv6.org/pub/usagi/snap/>

which they release every other weeks. This package contains the latest patched kernelsource and the source of basic IPv6 capable applications. After successful compiling and installing we only have a bare system, without any tools, which helps us to set up the desired gateway functions. You can find only one implementation of NAT-PT on linux made by British Telecom Labs in 12 April 1999, so the code is pretty old.:

<http://www.ipv6.or.kr/english/natpt-overview.htm>

¹stable releases out of sight from this document, because the aim is testing the most recent codes instead of setting up a production state system

The code not just old, but pretty ugly and it needs an

```
#include <time.h>
```

line in the nat-pt.c file to compile. Afterwards the further problem is, that it only supports two protocols dns and ftp. Since no further development or support available it does not seem to be a viable solution. The new solution from BT is called Ultima which is a commercial software product, available only on FreeBSD.

3.3 *BSD

The most mature system today for IPv6 is one of the BSD systems, no better proof than all of the documents in connection with IPv6 at least mentioning BSD. The IPv6 protocol stack on BSD comes from the KAME project. They are a little bit ahead, than USAGI, but since they are working tightly together the enhancements of the KAME project will be probably appearing in the USAGI project later as well.

First let us start with FreeBSD for testing, because it is known to be the easiest BSD system. Though anybody can archive the same results with any other kind of BSD as well. The choice is free, the KAME project supports every of the BSD systems ². The KAME project issues snapshots every week, so a bit more rapidly than USAGI does. The stable releases of KAME are already integrated into all BSD systems, so they "come out of the box" with the most stable and up to date IPv6 stack.

Installing FreeBSD is not much more difficult, than a Slackware Linux. Though the system is different, so you have to get used to it. The GNU software packages are more or less the same, so if somebody is familiar with linux it is not too hard. The most notable thing to the install, that BSD systems are more sensitive for correct hardware components and proper bios settings, so if the system hangs on install, it is best practice to set bios defaults or failsafe mode. After getting the KAME snapshot from <ftp://ftp.kame.net/pub/kame/snap/>, or any of the mirrors mentioned on the KAME homepage, we need to configure, compile and install the kernel and basic IPv6 capable applications. The INSTALL file describes the things to do clearly and by following the instructions anyone can succeeded.

²FreeBSD, OpenBSD, NETBSD, BSD/OS

3.3.1 Totd

The first step is, as mentioned in the theoretical chapter, that we need to set up our special DNS daemon. One possibility is totd, which stands for Trick Or Treat Daemon. totd is available from the ports tree under `/usr/ports/net/totd/` or simply from source:

<http://www.vermicelli.pasta.cs.uit.no/ipv6/software.html>

Setting up totd is simple. We have to specify our prefix, which will be added before the translated IPv4 address, and a forwarder DNS server, which can be our existing DNS system on another machine, or we can move the normal BIND to other than the default port on the same machine as we run totd and ask it, to forward the queries to the non-standard port. totd acts both on IPv4 and IPv6, but it can be forced to use just one of them. We can specify more than one prefix, which is useful if we want implement simple load balancing between some gateway machines. Every time totd is reached by a client it jumps to the next prefix, so the client will turn to another gateway than the one before if the prefixes are assigned to different gateways. Further possibilities are described in the documentation totd(8). My config file looks like this :

```
prefix 3ffe:400:3d0:1000::
forwarder 3ffe:400:3d0::1
```

After filling out the required config file: totd.conf, we can simply start totd, by typing totd. Most simple way to test totd is, trying to reach an IPv4 host with IPv6 capable telnet program on a client machine³ using the FQDN of the host. Before doing that we have to make sure, that our client turns to the machine running totd, during resolving. Unix like systems using typically `/etc/resolv.conf` to setup this. For example:

```
[client]# /usr/local/v6/bin/telnet -6 www.ipv4.com
Trying 3ffe:400:3d0:1000::cfa4:7ca0...
telnet: Unable to connect to remote host: No route to host
```

Where 3ffe:400:3d0:1000 is our "totd prefix", and cfa4:7ca0 is the translated value of 207.164.124.160. The space in-between filled up with zeros. We have got the unable to connect message, since we did not set up anything yet to handle this kind of traffic, but at least we can be sure the name resolution trick works by now.

³It does not matter if the machine is unreachable, or not running the telnet service, because we only want to test the name resolving phase

3.3.2 Faithd

Faithd is one way to handle the situation. Faithd is a TRT solution. First we have to set our kernel state to the following values:

```
sysctl -w net.inet6.ip6.accept_rtadv=0
sysctl -w net.inet6.ip6.forwarding=1
sysctl -w net.inet6.ip6.keepfaith=1
```

Afterwards we can bring up our faith interface, issuing :

```
ifconfig faith0 up
```

We want to route all incoming traffic from our regular interface to the faith0 interface with the predefined prefix, regardless the interface is assigned with that address or not, in order to translate it:

```
route add -inet6 3ffe:400:3d0:1000:: -prefixlen 64 ::1
route change -inet6 3ffe:400:3d0:1000:: -prefixlen 64 -ifp faith0
```

We have a possibility to limit the stations, which can use the faith service, with basic acl. So we set up our faithd.conf like:

```
3ffe:400:3d0::/64 permit 0.0.0.0/0
```

And finally we can start faithd on each service, what we want to relay:

```
/usr/local/v6/sbin/faithd http
/usr/local/v6/sbin/faithd https
/usr/local/v6/sbin/faithd telnet
/usr/local/v6/sbin/faithd irc
/usr/local/v6/sbin/faithd finger
/usr/local/v6/sbin/faithd pop3
/usr/local/v6/sbin/faithd pop3s
/usr/local/v6/sbin/faithd smtp
/usr/local/v6/sbin/faithd smtps
/usr/local/v6/sbin/faithd imap
/usr/local/v6/sbin/faithd imaps
```

The above lines should be added to one of the systems startup scripts in order to start them without interaction after next boot. When faithd running in daemon mode, it listens to the TCPv6 port service, so it is not possible to run local daemons for these services. If we want to run a local service which is relayed as well, we need to use INETD for this purpose. Typically ssh is such a service.

The corresponding configuration line in inetd.conf follows:

```
ssh stream tcp6/faith nowait root /usr/local/v6/sbin/faithd /usr/sbin/sshd -i
```

On the client side, we should insert a static route entry into the routing table in order to ensure all traffic with the "faith prefix" goes to the faith box. For example:

```
[client]# ip -f inet6 route add 3ffe:400:3d0:1000::/64 \
via 3ffe:400:3d0:0:2e0:29ff:fe24:f0a2
```

After all, we are ready to test. Let us try to open the port 80 on a webserver

```
/usr/local/v6/bin/telnet -6 www.fht-esslingen.de 80
Trying 3ffe:400:3d0:1000::866c:2203...
Connected to www.fht-esslingen.de.
Escape character is '^']'.
```

And it works ! Further documentation in faithd manpage section 8.

3.3.3 NAT-PT

While faithd is a userspace implementation, nat-pt is on the kernel level, so you have to recompile your kernel if it does not contain the support for it yet. The /usr/local/v6/etc/natpt.conf file:

```
prefix 3ffe:400:3d0:1000::
# This is our prefix to translate
map from any6 to 134.108.35.42 port 28672 - 32767
# We want to map any IPv6 traffic to our IPv4
# address on the translator box and we want to
# allocate this traffic only to the defined port range
map enable
```

Then we read the config file by the natptconfig program:

```
/usr/local/v6/sbin/natptconfig -f /usr/local/v6/etc/natpt.conf
```

We can check if all went fine:

```
/usr/local/v6/sbin/natptconfig show rules
0: from any6 to 134.108.35.42 port 28672 - 32767
```

On the client we need the same static route entry and resolve setting as we needed with `faithd`, but of course at this time we do not want to reroute the traffic to the `faith0` interface on our gateway. So do the same test as with `faithd` on the client:

```
[client]# /usr/local/v6/bin/telnet -6 www.fht-esslingen.de 80
Trying 3ffe:400:3d0:1000::866c:2203...
Connected to www.fht-esslingen.de.
Escape character is '^]'.

```

Worked out again ! We can have a look on the translation table at the gateway, to check what happend exactly:

```
/usr/local/v6/sbin/natptconfig show xlate
proto Local Address (src)  Local Address (dst)
tcp    3ffe=fe24:f165.1051      3ffe=866c:2203.80

```

```
Remote Address (src) Remote Address (dst)
134.108.35.42.28673  134.108.34.3.80

```

```
Ipkts Opkts  Idle  (state)
1      2    00:01:20 ESTABLISHED

```

This solution is better in the aspect, that we can force the `nat-pt` mechanism between port ranges, so we can run local daemons without restrictions if we choose the ranges carefully. And it is maybe closer to the ones, who already running NAT systems. NAT-PT maybe used to make IPv6 host services available for IPv4 hosts as well. Further documentation in `natpt.conf(5)` and `natptconfig(8)`.


```
20:47:37.264589 3ffe:400:3d0:0:2e0:29ff:fe24:f165.1029 >
3ffe:400:3d0:1000::866c:2203.http:
S 2493680500:2493680500(0) win 5760
<mss 1440,sackOK,timestamp 72623 0,nop,wscale 0>
```

Since these prefixes statically routed to the gateway, the reply comes from the gateway with the fake address back to the client:

```
20:47:37.265557 3ffe:400:3d0:1000::866c:2203.http >
3ffe:400:3d0:0:2e0:29ff:fe24:f165.1029:
S 1772341350:1772341350(0) ack 2493680501 win 5792
<mss 1460,nop,nop,timestamp 278923490 72623>
```

The client thinks everything is alright, so the 3 way handshake is well done, but between the client and gateway, not the client and the webserver:

```
20:47:37.265644 3ffe:400:3d0:0:2e0:29ff:fe24:f165.1029 >
3ffe:400:3d0:1000::866c:2203.http:
. ack 1 win 5760 <nop,nop,timestamp 72623 278923490>
```

Though meanwhile the gateway made a parallel action with the webserver as well, so from the gateway point of view we can see that the gateway tries to connect to the webserver:

```
21:57:47.273713 134.108.35.42.28688 > 134.108.34.3.80:
S 3336234750:3336234750(0) win 5760
<mss 1440,sackOK,timestamp 154876 0,nop,wscale 0>
```

The webserver replies:

```
21:57:47.274176 134.108.34.3.80 > 134.108.35.42.28688:
S 2635875760:2635875760(0) ack 3336234751 win 5792
<mss 1460,nop,nop,timestamp 279005741 154876> (DF)
```

And the other side of the handshake has been done:

```
21:57:47.274570 134.108.35.42.28688 > 134.108.34.3.80:
. ack 1 win 5760 <nop,nop,timestamp 154876 279005741>
```

Of course meanwhile the gateway deals with the name resolving for the client as well.

The above actions are similar to nat-pt and trt, the only difference is on the gateway side, when nat-pt uses only ports in the predefined range.

Appendix A

URL list

Socks implementation by Nec

<http://www.socks.nec.com/>

The USAGI Project

<http://www.linux-ipv6.org/>

Nat-pt on linux by BT

<http://www.ipv6.or.kr/english/natpt-overview.htm>

The FreeBSD Homepage

<http://www.freebsd.org/>

The OpenBSD Homepage

<http://www.openbsd.org/>

The KAME Project

<http://www.kame.net/>

Totd

<http://www.vermicelli.pasta.cs.wit.no/ipv6/software.html>

Appendix B

Daemon configuration files

default USAGI and KAME root tree, where binaries and config files can be found after compiling and installation:

```
/usr/local/v6/
```

```
/usr/local/etc/totd.conf
```

```
prefix 3ffe:400:3d0:1000::  
forwarder 3ffe:400:3d0::1
```

```
/usr/local/etc/faithd.conf
```

```
3ffe:400:3d0::/64 permit 0.0.0.0/0
```

faithd addition to /etc/rc.network

```
sysctl -w net.inet6.ip6.accept_rtadv=0
sysctl -w net.inet6.ip6.forwarding=1
sysctl -w net.inet6.ip6.keepfaith=1
ifconfig faith0 up
route add -inet6 3ffe:400:3d0:1000:: -prefixlen 64 ::1
route change -inet6 3ffe:400:3d0:1000:: -prefixlen 64 -ifp faith0
/usr/local/v6/sbin/faithd http
/usr/local/v6/sbin/faithd https
/usr/local/v6/sbin/faithd telnet
/usr/local/v6/sbin/faithd irc
/usr/local/v6/sbin/faithd finger
/usr/local/v6/sbin/faithd pop3
/usr/local/v6/sbin/faithd pop3s
/usr/local/v6/sbin/faithd smtp
/usr/local/v6/sbin/faithd smtps
/usr/local/v6/sbin/faithd imap
/usr/local/v6/sbin/faithd imaps
```

/usr/local/v6/etc/natpt.conf

```
prefix 3ffe:400:3d0:1000::
map from any6 to 134.108.35.42 port 28672 - 32767
map enable
```

client routing setup on unix like systems:

```
ip -f inet6 route add 3ffe:400:3d0:1000::/64 \
via 3ffe:400:3d0:0:2e0:29ff:fe24:f0a2
```

Appendix C

FreeBSD kernel configuration

```
machine i386
cpu I686_CPU
ident KAME
maxusers 32
options MATH_EMULATE #Support for x87 emulation
options INET #InterNETworking
options INET6 #IPv6 communications protocols
options FFS #Berkeley Fast Filesystem
options FFS_ROOT #FFS usable as root device [keep this!]
options SOFTUPDATES #Enable FFS soft updates support
options MFS #Memory Filesystem
options MD_ROOT #MD is a potential root device
options NFS #Network Filesystem
options NFS_ROOT #NFS usable as root device, NFS required
options MSDOSFS #MSDOS Filesystem
options CD9660 #ISO 9660 Filesystem
options CD9660_ROOT #CD-ROM usable as root, CD9660 required
options PROCFS #Process filesystem
options COMPAT_43 #Compatible with BSD 4.3 [KEEP THIS!]
options UCONSOLE #Allow users to grab the console
options USERCONFIG #boot -c editor
options VISUAL_USERCONFIG #visual boot -c editor
options KTRACE #ktrace(1) support
options SYSVSHM #SYSV-style shared memory
options SYSVMSG #SYSV-style message queues
options SYSVSEM #SYSV-style semaphores
options P1003_1B #Posix P1003_1B real-time extensions
options _KPOSIX_PRIORITY_SCHEDULING
```

```

options ICMP_BANDLIM #Rate limit bad replies
options KBD_INSTALL_CDEV # install a CDEV entry in /dev
device isa
device eisa
device pci
device fdc0 at isa? port IO_FD1 irq 6 drq 2
device fd0 at fdc0 drive 0
device ata0 at isa? port IO_WD1 irq 14
device ata1 at isa? port IO_WD2 irq 15
device ata
device atadisk # ATA disk drives
device atapicd # ATAPI CDROM drives
device atapifd # ATAPI floppy drives
options ATA_STATIC_ID #Static device numbering
device ahc # AHA2940 and onboard AIC7xxx devices
device aic0 at isa?
device scbus # SCSI bus (required)
device da # Direct Access (disks)
device sa # Sequential Access (tape etc)
device cd # CD
device pass # Passthrough device (direct SCSI access)
device atkbd0 at isa? port IO_KBD
device atkbd0 at atkbd? irq 1 flags 0x1
device psm0 at atkbd? irq 12
device vga0 at isa?
pseudo-device splash
device sc0 at isa? flags 0x100
device vt0 at isa?
options XSERVER # support for X server on a vt console
options PCVT_FREEBSD=440
device npx0 at nexus? port IO_NPX irq 13
device apm0 at nexus? disable flags 0x20 # Advanced Power Management
device sio0 at isa? port IO_COM1 flags 0x10 irq 4
device sio1 at isa? port IO_COM2 irq 3
device sio2 at isa? disable port IO_COM3 irq 5
device sio3 at isa? disable port IO_COM4 irq 9
device ppc0 at isa? irq 7
device ppbus # Parallel port bus (required)
device lpt # Printer
device miibus # MII bus support
device tx # SMC EtherPower II (83c170 'EPIC')

```

```
pseudo-device loop # Network loopback
pseudo-device ether # Ethernet support
pseudo-device tun # Packet tunnel.
pseudo-device pty # Pseudo-ttys (telnet etc)
pseudo-device md # Memory "disks"
pseudo-device gif # IPv6 and IPv4 tunneling
pseudo-device faith 1 # IPv6-to-IPv4 relaying (translation)
pseudo-device bpf #Berkeley packet filter
options IPSEC #IP security
options IPSEC_ESP #IP security (crypto; define w/ IPSEC)
options NEW_STRUCT_ROUTE # mandatory
options ALTQ #alternate queueing
options NATPT # IPv6 -> IPv4 translation.
options NATPT_NAT # IPv4 -> IPv4 NAT.
options EXT2FS
options COMPAT_LINUX
device pcm
```

Appendix D

Installation checkpoints

After installing the FreeBSD base system (It is recommended, to install the ports). Checking system settings: `ifconfig` should report at least one more interface, than `lo0` (`tx0` for SMC EtherPower II 10/100). In the line beginning with `inet`, should be the IPv4 properties of the NIC. `netstat -rn` should report the routing table(s) including the default gateway(s). `cat /etc/resolv.conf` to check name resolve settings. If everything looks good, than we can try to ping the default gateway by its IP address. (`ping 134.108.32.254 #` depending on the vlan) If it worked out, we can try to ping machines outside our network (`ping www.uni-stuttgart.de #` for example) If it replies without any error, we can be sure, that the IPv4 settings are good.

Now we should get the latest kame snapshot and extract it. Follow the instructions in the attached README and INSTALL files. If the kernel is compiled and installed, we should reboot the system. Afterwards `uname -a` should indicate the date and path we compiled our kernel. Check again the network interface with `ifconfig` and the routing tables with `netstat -rn`. Now we should have `inet6` line in `ifconfig` output and Internet6 routing tables in `netstat` output. Let us try to reach another IPv6 station with `ping` (`ping6 3ffe:400:3d0:0:2e0:29ff:fe24:f165 #` `itpc45` for example)

If it goes well, we can start setting up the gateway functionality. `cd /usr/ports/net/totd/; make install #` to install `totd`, the dns trick daemon. On one client set the `bsd box` as its nameserver and then `telnet -6 www.ipv4.com` should return something like: `Trying 3ffe:400:3d0:1000::cfa4:7ca0...` After all, we can set up `faithd` or `nat-pt`, as we wish following the instructions in the `Faithd` or `NAT-PT` chapters.

If you get in trouble you can find more answers in the FreeBSD handbook located in `/usr/share/doc/handbook/` with a default install, or the man pages and readme files submitted with the source codes or packages. The only thing, that documentation does not talk about is, that you should set up

a static route entry on the client which tells the client to route all traffic addressed to the predefined "faithd" prefix to the gateway (ip -f inet6 route add 3ffe:400:3d0:1000::/64 via 3ffe:400:3d0:0:2e0:29ff:fe24:f0a2 # for example).