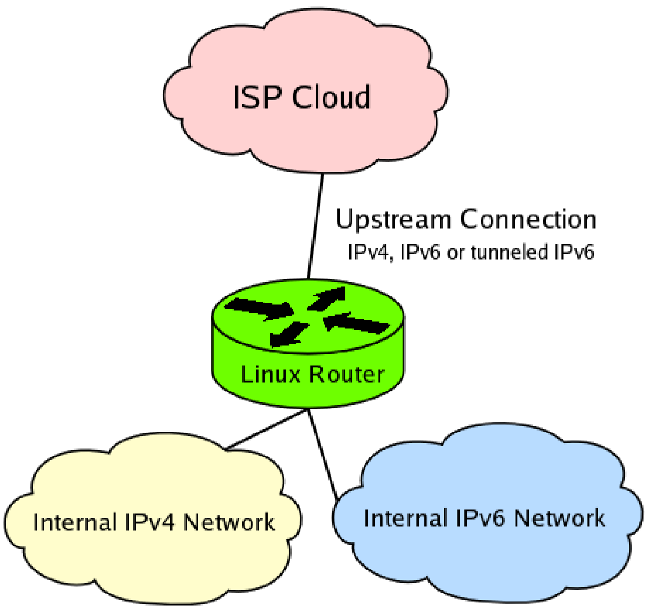
Building an IPv6 router with GNU/Linux

This step-by-step guide is intended to help the novice gain experience with IPv6 by setting up an Linux IPv6 router. It can help more experienced IPv6 users, as it shows how quick and easy building an IPv6 router that will perform advanced functions, like stateful packet translations between IPv4/IPv6 or acting as a DNS proxy, really is. In this paper I will be using Fedora Core 4 as it is an easy to use distribution, perfectly suitable for building an IPv6 router, but I will keep all my examples as generic as possible, so that you should be able to replicate them on almost any modern Linux distribution. This paper is **not** a guide to IPv6. You should know the basics of the protocol itself. If you don't there are hundreds of excellent papers on the Internet explaining the protocol and I don't indent to write Yet Another Guide About What's New In IPv6 (YAGAWNIv6 anyone ?). Enough said. Lets jump to what's important: building our router.

Setup Scenario



Linux is an incredibly versatile operating system that can be easily adapted to different network conditions. In this we assume that the router we will be building will be placed as illustrated above. There is an upstream connection (not necessarily to an ISP, simply to some higher hierarchy router), and there are two LANs we are connecting, one IPv6 only, one IPv4 only. The upstream connection can be a native IPv4 connection (most common), a native IPv6 connection (rarity these days) or a IPv6 connection but using tunneling over IPv4 (quite common among 6bone participants and other early adopters). The type of connection that we have will change some of the configuration options we will need to perform, but I will clearly indicate when that comes into play.

Installation Notes

This is not a guide to installing Fedora Core, but note the following:

- **Install radvd** - found under "Network Servers" in the installation process - this is the daemon that will used to assign IPv6 addresses to your clients.
- **Make sure** all network interface cards are Linux compatible.
- **Don't enable SELinux** - if you have little experience with configuring it. There are problems with radvd when using SELinux in it's default configuration, and I don't want to add to the confusion by introducing the entire new subject like SELinux.

Planning Addressing

Before you start typing out commands write down the specific addresses and networks that you plan to configure on your router. In my scenario I'm going to use an upstream IPv4 connection, call it eth0; An IPv6 interface eth1, and an IPv6 tunnel sit1 that will be my only connection to the global IPv6 cloud. My ISP has assigned me the address block 2001:468:181:f100::/56, and an IPv6 address of my IPv6 tunnel endpoint of 2001:468:181:ff::2 (remote endpoint is 2001:468:181:ff::1). My address block allows me to create 256 subnets, because every network mask is 64 bits long I have 8 bits free for subnetting (64 - 56 = 8) which allows me to create 2^8 = 256 subnets (I don't have to substract 2 from this like with IPv4, because IPv6 doesn't need addresses for broadcasts or network identifiers). I'm going to use the first available subnet to create my internal IPv6 network (2001:468:181:f100::/64) and use the first available IP from that network for my interface IP (2001:468:181:f100::1/64). I have a public IPv4 address on eth0 198.209.98.4, and the IP address of my tunnel endpoint is x.y.z.v (put whatever you want here - I'm just concerned about the privacy of my tunnel provider ;). The following table sums up this information.

Interface Name	Local Address	Remote Address
eth0 (IPv4)	198.209.98.4/29	N/A
eth1 (IPv6)	2001:468:181:f100::1/64	N/A
sit (IPv6-in-IPv4 tunnel)	198.209.98.4 2001:468:181:ff::2	x.y.z.v 2001:468:181:ff::1

Having your interfaces written out like that makes things a lot easier than trying to calculate the necessary values on the spot.

Internal IPv6

First configure your IPv6 interface and setup the appropriate route. In my example this is done like this:

```
ip a a 2001:468:181:f100::1 dev eth1
ip r a 2001:468:181:f100::/64 dev eth1
```

You can add these two lines to the begining of /etc/rc.local to make them execute during boot time.

Assuming that every computer on your internal network already has an IPv6 stack installed (Linux should by default, you may need to do 'modprobe ipv6'. Windows has IPv6 since SP1 - you can enable it through the connection properties -> Network Connections -> Right Click on LAN1 -> Properties) the first thing you should do to get internal connectivity running is install the **radvd** daemon that will allow your workstations to automatically configure their interfaces to appropriate IPv6 addresses.

After installing radvd (during installation, after installation from the 3 CD, or using the command 'yum install radvd' as root) you must configure it to distribute the appropriate information. The basic radvd configuration file (by default found in /etc/radvd.conf) should look like this:

```
interface eth0                    // identifies the interface we are advertising on
{                                 // if you need, you can have multiple interfaces defined

    AdvSendAdvert on;             // this simply means that we are sending advertisements

    MinRtrAdvInterval 5;          // these options control how often advertisements are sent
    MaxRtrAdvInterval 15;         // these aren't mandatory, but valuable settings
                                  // I will discuss them in a moment

    prefix 2001:468:181:f100::/64 // netmask length must be "/64" (see RFC 2462, sect 5.5.3, page 18)
    {
        AdvOnLink on;             // Says to a host: "Everyone sharing this prefix is on the same,"
                                  //                 "local link as you."
        AdvAutonomous on;         // Says to a host: "Use this prefix to autoconfigure your address."
    };
};

interface eth1,eth2,etc .... // if we wish to advertise on a second interface
{
};
```

The reason MinRtrAdvInterval and MaxRtrAdvInterval are necessary is because there is a glitch in the Microsoft implementation of IPv6 (at least at the time of this writing) that prevents the host from sending Router Solicitation messages after boot even if the host has IPv6 enabled (it works as it should, if you bring the interface down and then up again). If these settings would not be present in the config file, the default prefix broadcast interval is 10 minutes, and your users would either have to wait until the next broadcast for their computers to get their IPv6 addresses or bring their network interfaces down and then back up again manually (and that's a hassle, and no one likes hassle). You can find dozens more options to configure by typing 'man radvd.conf'.

For radvd to be able to start IPv6 routing must be enabled. You do this by typing:

```
sysctl -w net.ipv6.conf.all.forwarding=1
```

Add the line 'net.ipv6.conf.all.forwarding = 1' to /etc/sysctl.conf to make this be applied during startup.

Finally make sure radvd starts up every time at boot. You can do this through graphical RedHat service configuration utility, or if you have no graphical environment (as I do) installed, by typing:

```
ln -s /etc/init.d/radvd /etc/rc3.d/S91radvd
```

After this is complete you should be able to open a console on one of your hosts on the IPv6 network and ping the router by typing:

```
ping6 2001:468:181:f100::1
```

If this doesn't work check if radvd is really running 'ps -A|grep radvd' on your server and that your clients have IPv6 addresses autoconfigured from the advertised prefix (ipconfig and ifconfig on Windows and Linux respectively). If the daemon is not functioning, make sure that you have IPv6 forwarding enabled, and check /var/log/messages ('tail -n 50 /var/log/messages') for errors. If radvd is running but your clients are not autoconfiguring themselves, check to make sure you have an IPv6 stack installed. NetBSD users should check that your computer accepts Router Advertisement messages (www.google.com for more information on this - 'setting NetBSD to accept Router Advertisements'). If the stack is installed and you are sure that your computer is accepting Router Advertisements (ICMPv6 packets aren't blocked by any firewall), use a network traffic analazer to see if the actual RA packets are seen on the wire. If not, configure eth1 on the router and your host interface with an IPv4 address and try pinging over IPv4. If this doesn't work, check your cabling, and make sure you're in the same broadcast domain as the IPv6 router. If you are still unable to find a solution to your problem, write me an email.

Global Reachability

There are three different situations (native IPv6, tunneled IPv6, IPv4-only) possible regarding upstream connectivity that will affect the steps you must take to establish global IPv6 connectivity.

Case 1

If you have a native IPv6 connection to your upstream provider your job is simple. Route packets to the next hop router. If the address of your ISP's router is 2001:468:181:ff::1, you type the following command (add to /etc/rc.local to make it be executed at boot time):

```
ip r a 2000::/3 via 2001:468:181:ff::1
```

This assumes that there exists a route to 2001:468:181:ff::1. Normally this shouldn't be problem but I have encountered situation where it was necessary to do:

```
ip r a 2001:468:181:ff::/64 dev eth0
```

Assuming that it's eth0 that is on the same link as the upstream router.

If you have no IPv6 connectivity whatsoever (even a tunneled one) then you can still use NAT-PT to connect your network to the Internet. I discuss setting up NAT-PT later in this document.

Case 2

If you have a tunneled connection (IPv6-in-IPv4), it is relatively easy to create a tunnel under Linux by using the **ip** command. Using the addresses from my example, the following commands will create a IPv6-in-IPv4 tunnel:

```
ip tunnel add sit1 mode sit remote 164.113.235.201 // creates tunnel
ip link set sit0 up // brings up
ip a a 2001:468:181:ff::2 dev sit1
// say our end of the IPv6 tunnel has the address 2001:468:181:ff::2
```

You should be now able ping any global IPv6 address from your router or any other host on your internal IPv6 network.

Case 3: NAT-PT

We have created an internal IPv6 network and achieved global reachability for IPv6 (or not ;). If you are running a dual stack network, you can reach both IPv4 and IPv6 nodes on the Internet - you are basicaly done. But what if you wish to have only IPv6 running on your network? What do you do if you want to drop IPv4 from your network completely? There are a couple of solutions to this problem. I'm only going to talk about the one I've found most suitable for most scenarios: NAT-PT. To find out more about the other possibilities check out my friend's website, it contains a lot of good links (www.google.com search 'IPv6 to IPv4 communication').

The first step to setting up NAT-PT on GNU/Linux is to download it, and install it on your system using RPM. Check the download page, as new version of NAT-PT may be available than at the time of this writing.

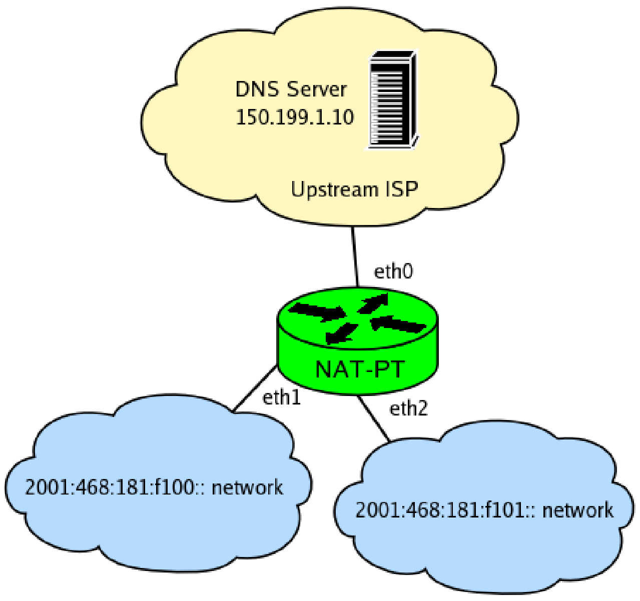
```
[root@localhost ~]# wget http://tomicki.net/download.php?id=76
--11:21:53--  http://tomicki.net/download.php?id=76
           => `download.php?id=76'
Resolving tomicki.net... 195.187.102.64
Connecting to tomicki.net[195.187.102.64]:80... connected.
HTTP request sent, awaiting response... 302 Found
Location: trash/res/naptd/naptd-0.3-1.i386.rpm [following]
--11:21:53--  http://tomicki.net/trash/res/naptd/naptd-0.3-1.i386.rpm
           => `naptd-0.3-1.i386.rpm'
Connecting to tomicki.net[195.187.102.64]:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 71,893 [application/x-rpm]

100%[====================================>] 71,893        --.--K/s

11:21:53 (2.46 MB/s) - `naptd-0.3-1.i386.rpm' saved [71,893/71,893]

[root@localhost ~]# rpm -ivh naptd-0.3-1.i386.rpm
Preparing...                ########################################### [100%]
   1:naptd                  ########################################### [100%]
Please run 'naptd-confmaker' to complete the installation process.
```

The next step is to configure NAT-PT to your needs. There are many advanced features that my implementation of NAT-PT has, but they are described in detail in the documentation, so in this document I will go over only the basics of setting up NAT-PT.

NAT-PT works by simulating a virtual, non-existant IPv6 network (also refered to as the NAT-PT prefix) that contains all of the current IPv4 Internet. You must decide what network number you shall use for this network, and configure NAT-PT appropriately. If you already have a IPv6 block allocated to you and are globaly reachable over IPv6, it is generally a good idea to use a network from your own block. I recommend using the last available subnetwork from you block, so if for example your ISP has allocated you

2001:468:181:f100::/56, you would use 2001:468:181:f1ff::. If you do not have a block of IPv6 addresses or do not wish to use a subnet from your block, you can still use the default network of 2000:ffff:: - this network should not be used anywhere in the global cloud, so there should be no problems with routing (there where none in the testing I've done), but I give no guarantees.

To configure NAT-PT, you must first become root, and then type, 'naptd-confmaker', to start the wizard that will guide you through the setup process. There are only three options that can't be left at their default values. The first two are the inside (IPv6) and outside (IPv4) interfaces, and the third one is a question about your current IPv4 DNS server which will help you calculate the correct IPv6 address for to use for DNS resolution if you choose to use NAT-PTs built-in DNS translation mechanism. This may all sound a bit confusing.



You can configure more than one interface to be considered 'inside' or 'outside'. Assuming that you will use the default address for the 'virtual network' that NAT-PT requires, your current IPv4 DNS server address is 150.199.1.10, and your network topology looks like on the picture above, then this is the way you should answer the setup questions:

```
[root@localhost ~]# naptd-confmaker
Ataga IPv4/IPv6 NAPT Configuration Maker
(c) 2005 by Lukasz Tomicki

Do you want to create a new configuration? [Y/n] {return}

Do you want IPv4 addresses from the outside interfaces to be automatically used
as part of the NAT pool? [Y/n] {return}

Do you want to configure additional address as part of your NAT pool? [y/N] {return}

Do you want to create a pool of public IPv4 addresses that will allow incoming
connections to be dynamically mapped to appropriate IPv6 addresses? [y/N] {return}

Do you want to create static mappings of public IPv4 addresses that will allow
incoming connections to reach IPv6 hosts? [y/N] {return}

Enter the name of the first inside (IPv6) interface that you want NAT-PT to
listen on.
interface (eth0 eth1 eth2 sit0): eth1 {return}

Do you want to enter more interfaces? [y/N]
y {return}
interface (eth0 eth1 eth2 sit0): eth2 {return}

Do you want to enter more interfaces? [y/N] {return}

Enter the name of the first outside (IPv4) interface that you want NAT-PT to
listen on.
interface (eth0 eth1 eth2 sit0): eth0 {return}

Do you want to enter more interfaces? [y/N] {return}

Enter the TCP translation timeout in seconds [86400]: {return}
Enter the UDP translation timeout in seconds [3600]: {return}
Enter the ICMP translation timeout in seconds [30]: {return}

Enter the IPv6 prefix that will be used as the destination for translations.
prefix [2000:ffff::]: {return}

Please enter the IPv4 address of the DNS server you are currently using.
IPv4 DNS server: 150.199.1.10 {return}

You can configure hosts for automatic DNS translation by using the DNS server below.
IPv6 DNS Server: 2000:ffff::96c7:10a

Thank you for choosing Ataga as you IPv4/IPv6 NAT-PT solution.
Setup is now complete. Type 'naptd' to start NAT-PT.
```

After that is done, you must setup iptables and ip6tables, for reasons which are explained in detail in the documentation. Many

Linux distributions don't come with ip6tables today, but you can surely obtain an appropriate package from a RPM mirror like www.rpmfind.net. After installing ip6tables and making sure both iptables and ip6tables are configure to be started at boot (either the Redhat graphical Service configuration tool or ln -s /etc/init.d/iptables /etc/rc3.d/S50iptables, ln -s /etc/init.d/ip6tables /etc/rc3.d/S51ip6tables), add the following to your firewall configuration files:

```
for ip6tables:
-A OUTPUT -p icmpv6 --icmpv6-type 1 -j DROP
```

In order for NAT-PT to work correctly, ip6tables must be configured to drop all outbound ICMP "Destination Unreachable" packets to prevent your system from sending "Route Unreachable" messages. The above accomplish that.

If your router is configured to perform IPv6 forwarding (if net.ipv6.conf.all.forwarding=1, as it should be if you have global IPv6 reachability) you must drop all packets going to the NAT-PT prefix (default: 2000:ffff::) to prevent them from going out to the global cloud. The following rule will do:

```
-A FORWARD -d 2000:ffff:: -j DROP
// remember to set this to whatever you chose as your prefix
```

The second important thing is the configuration of iptables. If you intend to use the outbound IPv4 addresses of your interfaces as part of your translation pool, you must DROP all packets that are not part of NEW, ESTABLISHED, or RELATED connections. This should be part of a healthy firewall policy anyway. A set of rules as the one below will work perfectly.

```
for iptables:
-A INPUT -i lo -j ACCEPT
-A INPUT -m state --state ESTABLISHED -j ACCEPT
// here go all your IPv4 inbound firewall rules
// I suggest having at least:
-A INPUT -m state --state NEW -p tcp -m tcp --dport 22 -j ACCEPT // for SSH remote management
-A INPUT -j DROP // this rule must be last
```

If everything went OK, after starting NAT-PT (just type 'naptd'), iptables and ip6tables, you should be able to ping your DNS server from one of you internal hosts (assuming it has a valid IPv6 address and default-router set).

The RPM package will automatically make naptd run during startup, so you don't need to worry about that.

Because you have entered both your NAT-PT prefix and preferred DNS server while configuring NAT-PT, setup was able to calculate the IPv6 address of the DNS server you should configure your clients to use if you want them to take advantage of the built-in DNS translation mechanism that NAT-PT provides. In your example this was 2000:ffff::96c7:10a, and this is the DNS address you would configure for your clients to have them use IPv6 whenever possible. There are however two situations when you wouldn't want to use the DNS translator that comes with NAT-PT. If you have a global IPv6 reachability, then you'd rather resolve the names of IPv6 servers to their IPv6 addresses and not use NAT-PT to communicate with them, while using NAT-PT only for those servers that do not have IPv6 addresses. The second situation when you can't use the NAT-PT's built-in DNS translator is when you are using an operating system what will simply not resolve DNS over IPv6 transport like most Windows implementation as of the time of this writing. For these two special occasions you must use a DNS proxy server like totd, which brings us to the next section of this document:

Installing DNS Proxy - totd

Totd can be obtained from http://www.vermicelli.pasta.cs.uit.no/ipv6/software.html, but before you run out and get it, let me tell you about the bug found in the version awailable there. Totd was written quite some time ago and is compiled with the -Werror flag. This means that every warning encountered during compilation will be counted as an error and will bring the compilation to a halt. This feature is nice when used during testing, but a bad idea when used with a release version, as older software sometimes generates new warnings with newer, more strict compilers. To make a long story short, the version of totd from the official website does not compile, but worry no more, I've debuged the source code to a state that compiles without problems. It is available on my website in the form of source code: totd-1.5.tar.gz or a more convienient RPM, totd-1.5-1.i386.rpm.

After downloading and installing the RPM package. You must edit the configuration file, found in /etc/totd.conf, before you can start using totd as your DNS proxy. By default it will look like this:

```
; $Id: totd.conf.sample,v 1.9 2003/09/17 15:56:20 dillema Exp $
; Totd sample configuration file
; you can have multiple forwarders, totd will always prefer
; forwarders listed early and only use forwarders listed later
; if the first ones are unresponsive.
forwarder 192.168.0.1 port 5000
forwarder 3ffe:ffff:fff:f::1234:1234 port 53
; you can have multiple prefixes or even no prefixes at all
; totd uses them in round-robin fashion
prefix 3ffe:abcd:1234:9876::
; the port totd listens on for incoming requests
port 5005
; the pidfile to use (default: /var/run/totd.pid)
pidfile /var/run/totd5005.pid
; interfaces totd listens on (UDP only for now and not on Linux)
; If left out totd will only open wildcard sockets.
interfaces lo0 ep0 ex0 an0
; 6to4 reverse lookup
stf
```

The important parts of this file are the:

- **forwarder** - this is the address of the server totd will forward its queries to. Format: forwarder IP_Addr [port default:53]

In our example, forwarder will be set as follows:

```
forwarder 150.199.1.10
```

- **prefix** - remember the NAT-PT prefix I talked so much about when setting up NAT-PT ? Set this to the same address you set the NAT-PT prefix, as this is prefix will be responsible for directing traffic to your NAT-PT translator. If you have multiple NAT-PT machines you can load balance traffic between them by creating multiple "prefix addr" lines, each corresponding to the NAT-PT prefix set for a different NAT-PT machines.
- **port** - simply the port totd is supposed to listen on, if not set, will default to 53.
- The rest of the options aren't important for the basic setup scenario described here. If you wish to learn more about them type 'man totd' or write me an email.

My final /etc/totd.conf looks like this:

```
; $Id: totd.conf.sample,v 1.9 2003/09/17 15:56:20 dillema Exp $
; Totd sample configuration file
; you can have multiple forwarders, totd will always prefer
; forwarders listed early and only use forwarders listed later
; if the first ones are unresponsive.
forwarder 150.199.1.10
forwarder 150.199.8.1
; you can have multiple prefixes or even no prefixes at all
; totd uses them in round-robin fashion
prefix 2001:468:181:f1ff::
; the port totd listens on for incoming requests
port 53
```

Finally add totd to run at startup. Add this to the very end of /etc/rc.local:

```
sleep 1 // for whatever reason totd had trouble running without this
totd
```

Conclusion

Enabling IPv6 on your network is possible today, and with the help of GNU/Linux and community based programs, it's easier than ever. Building an IPv6 router with GNU/Linux is a only a matter of a few a hours but the experience gained about IPv6 while be of great value in the coming years, as IPv6 deployment widens. If you have any questions about IPv6 or this paper, please write me an email.

"If you are not worried, then you must not be paying attention."

Last update: Saturday, 05th May, 2012