Linux IPv6 HOWTO

Peter Bieringer

pb (at) bieringer.de

Revision History

Revision Release 0.23	2002−03−27	Revised by: PB
See revision history for more		
Revision Release 0.22	2002−03−04	Revised by: PB
See revision history for more		
Revision Release 0.21	2002−02−26	Revised by: PB
See revision history for more		

The goal of the Linux IPv6 HOWTO is to answer both basic and advanced questions about IPv6 on the Linux operating system. This HOWTO will provide the reader with enough information to install, configure, and use IPv6 applications on Linux machines.

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Chapter 1. General

1.1. Copyright, license and others

1.1.1. Copyright

1.1.2. License

1.1.3. About the author

1.1.3.1. Internet/IPv6 history of the author

- 1993: I got in contact with the Internet using console based e−mail and news client (e.g. look for "e91abier" on groups.google.com, that's me).
- 1996: I got a request for designing a course on IPv6, including a workshop with the Linux operating system.
- 1997: Started writing a guide on how to install, configure and use IPv6 on Linux systems, called IPv6 & Linux − HowTo (see IPv6 & Linux − HowTo/History for more information).
- 2001: Started writing this new Linux IPv6 HOWTO.

1.1.3.2. Contact

The author can be contacted via e−mail at <pb at bieringer dot de> and also via his homepage.

He's currently living in Munich [northern part of Schwabing] / Bavaria / Germany (south) / Europe (middle) / Earth (surface/mainland).

1.2. Category

This HOWTO should be listed in category "*Networking/Protocols*".

1.3. Version, History and To−Do

1.3.1. Version

The current version is shown above.

1.3.2. History

1.3.2.1. Major history

2001−11−30: Starting to design new HOWTO.

2002−01−02: A lot of content completed, first public release of chapter 1 (version 0.10).

2002−01−14: More completed, some reviews, public release of the whole document (version 0.14).

1.3.2.2. Full history

See revision history at the end of this document.

1.3.3. To−Do

- Fill in missing content
- Finishing grammar checking

1.4. Translations

Translations always have to contain the URL, version number and copyright of the original document (but yours, too).

1.4.1. To German

A German translation is planned by me (German is my native language), but it won't happen until the document's change frequency is less than once per month, and I get enough free time to do it. If you have more free time than me, please feel free to take over the translation!

1.4.2. To other languages

Please wait until the document's change frequency is less than once per month.

1.5. Technical

1.5.1. Original source of this HOWTO

This HOWTO is currently written with LyX version 1.1.6fix4 on a Red Hat Linux 7.2 system with template SGML (DocBook book). It's available on LDP−CVS / users / Peter−Bieringer for contribution.

1.5.1.1. Code line wrapping

Code line wrapping is done using selfmade utility "lyxcodelinewrapper.pl", you can get it from CVS for your own usage: LDP−CVS / users / Peter−Bieringer

1.5.1.2. SGML generation

SGML is generated using export function in LyX.

Also some fixes are have to be made to create proper SGML code (see also here for the Perl programs LDP−CVS / users / Peter−Bieringer):

- Export of LyX table does not create proper "colspan" tags − tool for fixing: "sgmllyxtabletagfix.pl"
- LyX sometimes uses special left/right entinities for quotes instead the normal one, which will still exist in generated HTML. Some browsers don't parse this very well (known: Opera 6 TP 2 or Konqueror) − tool for fixing: "sgmllyxquotefix.pl"

1.5.2. On−line references to the HTML version of this HOWTO (linking/anchors)

1.5.2.1. Master index page

Generally, a reference to the master index page is recommended.

1.5.2.2. Dedicated pages

Because the HTML pages are generated out of the SGML file, the HTML filenames turn out to be quite random. However, some pages are tagged in LyX, resulting in static names. These tags are useful for references and shouldn't be changed in the future.

If you think that I have forgotten a tag, please let me know, and I will add it.

1.6. Preface

Some things first:

1.6.1. How many versions of a Linux & IPv6 related HOWTO are floating around?

Including this, there are three (3) HOWTO documents available. Apologies, if that is too many ;−)

1.6.1.1. Linux IPv6 FAQ/HOWTO (outdated)

The first IPv6 related document was written by *Eric Osborne* and called Linux IPv6 FAQ/HOWTO (please use it only for historical issues). Latest version was 3.2.1 released 14. July 1997.

Please help: if someone knows the date of birth of this HOWTO, please send me an e−mail (information will be needed in "history").

1.6.1.2. IPv6 & Linux − HowTo (maintained)

There exists a second version called IPv6 & Linux − HowTo written by me (*Peter Bieringer*) in pure HTML. It was born April 1997 and the first English version was published in June 1997. I will continue to maintain it, but it will slowly fade in favour of the Linux IPv6 HOWTO you are currently reading.

1.6.1.3. Linux IPv6 HOWTO (this document)

Because the IPv6 & Linux − HowTo is written in pure HTML it's not really compatible with the Linux Documentation Project (LDP). I (*Peter Bieringer*) got a request in late November 2001 to rewrite the IPv6 & Linux − HowTo in SGML. However, because of the discontinuation of that HOWTO (Future of IPv6 & Linux − HowTo), and as IPv6 is becoming more and more standard, I decided to write a new document covering basic and advanced issues which will remain important over the next few years. Dynamic content will be still found further on in the second HOWTO (IPv6 & Linux − HowTo).

1.7. Used terms

1.7.1. Network related

Link

A link is a layer 2 network packet transport medium, examples are Ethernet, Token Ring, PPP, SLIP, ATM, ISDN, Frame Relay,...

Node

A node is a host or a router.

Host

Generally a single homed host on a link. Normally it has only one active network interface, e.g. Ethernet or (not and) PPP.

Dual homed host

A dual homed host is a node with two network (physical or virtual) interfaces on two different links, but does not forward any packets between the interfaces.

Router

A router is a node with two or more network (physical or virtual) interfaces, capable of forwarding packets between the interfaces.

Tunnel

A tunnel is typically a point−to−point connection over which packets are exchanged which carry the data of another protocol, e.g. an IPv6−in−IPv4 tunnel.

NIC

Network Interface Card

1.7.2. Document related

1.7.2.1. Long code line wrapping signal char

The special character "¬" is used for signaling that this code line is wrapped for better viewing in PDF and PS files.

1.7.2.2. Placeholders

In generic examples you will sometimes find the following:

```
<myipaddress>
```

For real use on your system command line or in scripts this has to be replaced with relevant content (removing the < and > of course), the result would be e.g.

```
1.2.3.4
```

1.7.2.3. Commands in the shell

Commands executable as non–root user begin with $, e.g.

```
$ whoami
```

Commands executable as root user begin with #, e.g.

```
# whoami
```

1.8. Requirements for using this HOWTO

1.8.1. Personal prerequisites

1.8.1.1. Experience with Unix tools

You should be familiar with the major Unix tools e.g. *grep*, *awk*, *find*, ... , and know about their most commonly used command–line options.

1.8.1.2. Experience with networking theory

You should know about layers, protocols, addresses, cables, plugs, etc. If you are new to this field, here is one good starting point for you: linuxports/howto/intro_to_networking

1.8.1.3. Experience with IPv4 configuration

You should definitely have some experience in IPv4 configuration, otherwise it will be hard for you to understand what is really going on.

1.8.1.4. Experience with the Domain Name System (DNS)

Also you should understand what the Domain Name System (DNS) is, what it provides and how to use it.

1.8.1.5. Experience with network debugging strategies

You should at least understand how to use *tcpdump* and what it can show you. Otherwise, network debugging will very difficult for you.

1.8.2. Linux operating system compatible hardware

Surely you wish to experiment with real hardware, and not only read this HOWTO to fall asleep here and there. :)

Chapter 2. Basics

2.1. What is IPv6?

IPv6 is a new layer 3 transport protocol (see linuxports/howto/intro_to_networking/ISO − OSI Model) which will supersede IPv4 (also known as IP). IPv4 was designed long time ago (RFC 760 from January 1980) and since its inception, there have been many requests for more addresses and enhanced capabilities. Major changes in IPv6 are the redesign of the header, including the increase of address size from 32 bits to 128 bits. Because layer 3 is responsible for end−to−end packet transport using packet routing based on addresses, it must include the new IPv6 addresses (source and destination), like IPv4.

For more information about the IPv6 history take a look at older IPv6 related RFCs listed e.g. at SWITCH IPv6 Pilot / References.

2.2. History of IPv6 in Linux

To−do: better time−line, more content...

2.2.1. Beginning

The first IPv6 related network code was added to the Linux kernel 2.1.8 in November 1996 by Pedro Roque. It was based on the BSD API:

```
diff -u --recursive --new-file v2.1.7/linux/include/linux/in6.h
¬ linux/include/linux/in6.h
--- v2.1.7/linux/include/linux/in6.h Thu Jan 1 02:00:00 1970
+++ linux/include/linux/in6.h Sun Nov 3 11:04:42 1996
@@ -0,0 +1,99 @@
+/*
+ * Types and definitions for AF_INET6
+ * Linux INET6 implementation
+ * + * Authors:
+ * Pedro Roque <******>
+ *
+ * Source:
+ * IPv6 Program Interfaces for BSD Systems
+ * <draft-ietf-ipngwg-bsd-api-05.txt>
```

The shown lines were copied from patch−2.1.8 (e−mail address was blanked on copy&paste).

2.2.2. In between

Because of lack of manpower, the IPv6 implementation in the kernel was unable to follow the discussed drafts or newly released RFCs. In October 2000, a project was started in Japan, called USAGI, whose aim was to implement all missing, or outdated IPv6 support in Linux. It tracks the current IPv6 implementation in FreeBSD made by the KAME project. From time to time they create snapshots against current vanilla Linux kernel sources.

2.2.3. Current

Unfortunately, the USAGI patch is so big, that current Linux networking maintainers are unable to include it in the production source of the Linux kernel 2.4.x series. Therefore the 2.4.x series is missing some (many) extensions and also does not confirm to all current drafts and RFCs. This can cause some interoperability problems with other operating systems.

2.2.4. Future

USAGI is now making use of the new Linux kernel development series 2.5.x to insert all of their current extensions into this development release. Hopefully the 2.6.x kernel series will contain a true and up−to−date IPv6 implementation.

2.3. How do IPv6 addresses look like?

As previously mentioned, IPv6 addresses are 128 bits long. This number of bits generates very high decimal numbers with up to 39 digits:

```
2^{128}-1: 340282366920938463463374607431768211455
```

Such numbers are not really addresses that can be memorized. Also the IPv6 address schema is bitwise orientated (just like IPv4, but that's not often recognized). Therefore a better notation of such big numbers is hexadecimal. In hexadecimal, 4 bits (also known as "nibble") are represented by a digit or character from 0−9 and a−f (10−15). This format reduces the length of the IPv6 address to 32 characters.

```
2^{128}-1: 0xffffffffffffffffffffffffffffffff
```

This representation is still not very convenient (possible mix−up or loss of single hexadecimal digits), so the designers of IPv6 chose a hexadecimal format with a colon as separator after each block of 16 bits. In addition, the leading "0x" (a signifier for hexadecimal values used in programming languages) is removed:

```
2^{128}-1: ffff:ffff:ffff:ffff:ffff:ffff:ffff:ffff
```

A usable address (see address types later) is e.g.:

```
3ffe:ffff:0100:f101:0210:a4ff:fee3:9566
```

For simplifications, leading zeros of each 16 bit block can be omitted:

```
3ffe:ffff:0100:f101:0210:a4ff:fee3:9566  ->
¬ 3ffe:ffff:100:f101:210:a4ff:fee3:9566
```

One sequence of 16 bit blocks containing only zeroes can be replaced with "::". But not more than one at a time, otherwise it is no longer a unique representation.

```
3ffe:ffff:100:f101:0:0:0:1  ->  3ffe:ffff:100:f101::1
```

The biggest reduction is seen by the IPv6 localhost address:

```
0000:0000:0000:0000:0000:0000:0000:0001  ->  ::1
```

There is also a so−called *compact* (base85 coded) representation defined RFC 1924 / A Compact Representation of IPv6 Addresses (written 1996), never seen in the wild, but here is an example:

```
# ipv6calc --addr_to_base85 3ffe:ffff:0100:f101:0210:a4ff:fee3:9566
Itu&-ZQ82s>J%s99FJXT
```

> Info: *ipv6calc* is an IPv6 address format calculator and converter program and can be found here: ipv6calc

2.4. FAQ (Basics)

2.4.1. Why is the name IPv6 and not IPv5 as successor for IPv4?

On any IP header, the first 4 bits are reserved for protocol version. So theoretically a protocol number between 0 and 15 is possible:

- 4: is already used for IPv4
- 5: is reserved for the Stream Protocol (STP, RFC 1819) (which never really made it to the public)

The next free number was 6. Hence IPv6 was born!

2.4.2. IPv6 addresses: why such a high number of bits?

During the design of IPv4, people thought that 32 bits were enough for the world. Looking back into the past, 32 bits were enough until now and will perhaps be enough for another few years. However, 32 bits are not enough to provide each network device with a global address in the future. Think about mobile phones, cars (including electronic devices on its CAN−bus), toasters, refrigerators, light switches, and so on...

So designers have chosen 128 bits, 4 times more in length and 2^96 greater in size than in IPv4 today.

The usable size is smaller than it may appear however. This is because in the currently defined address schema, 64 bits are used for interface identifiers. The other 64 bits are used for routing. Assuming the current strict levels of aggregation (/48, /35, ...), it is still possible to "run out" of space, but hopefully not in the near future.

2.4.3. IPv6 addresses: why so small a number of bits on a new design?

While, there are (possibly) some people on the Internet who are thinking about IPv8 and IPv16, their design is far away from acceptance and implementation. In the meantime 128 bits was the best choice regarding header overhead and data transport. Consider the minimum Maximum Transfer Unit (MTU) in IPv4 (576 octets) and in IPv6 (1280 octets), the header length in IPv4 is 20 octets (minimum, can increase to 60 octets with IPv4 options) and in IPv6 is 48 octets (fixed). This is 3.4 % of MTU in IPv4 and 3.8 % of MTU in IPv6. This means the header overhead is almost equal. More bits for addresses would require bigger headers and therefore more overhead. Also, consider the maximum MTU on normal links (like Ethernet today): it's 1500 octets (in special cases: 9k octets using Jumbo frames). Ultimately, it wouldn't be a proper design if 10 % or 20 % of transported data in a Layer−3 packet were used for addresses and not for payload.

Chapter 3. Address types

Like IPv4, IPv6 addresses can be split into network and host parts using subnet masks.

IPv4 has shown that sometimes it would be nice, if more than one IP address can be assigned to an interface, each for a different purpose (aliases, multi−cast). To remain extensible in the future, IPv6 is going further and allows more than one IPv6 address to be assigned to an interface. There is currently no limit defined by an RFC, only in the implementation of the IPv6 stack (to prevent DoS attacks).

Using this large number of bits for addresses, IPv6 defines address types based on some leading bits, which are hopefully never going to be broken in the future (unlike IPv4 today and the history of class A, B, and C).

Also the number of bits are separated into a network part (upper 64 bits) and a host part (lower 64 bits), to facilitate auto−configuration .

3.1. Addresses without a special prefix

3.1.1. Localhost address

This is a special address for the loopback interface, similiar to IPv4 with its "127.0.0.1". With IPv6, the localhost address is:

```
0000:0000:0000:0000:0000:0000:0000:0001
```

or compressed:

```
::1
```

Packets with this address as source or destination should never leave the sending host.

3.1.2. Unspecified address

This is a special address like "any" or "0.0.0.0" in IPv4 . For IPv6 it's:

```
0000:0000:0000:0000:0000:0000:0000:0000
```

or:

```
::
```

These addresses are mostly used/seen in socket binding (to any IPv6 address) or routing tables.

Note: the unspecified address cannot be used as destination address.

3.1.3. IPv6 address with embedded IPv4 address

There are two addresses which contain an IPv4 address.

3.1.3.1. IPv4−mapped IPv6 address

IPv4−only IPv6−compatible addresses are sometimes used/shown for sockets created by an IPv6−enabled daemon, but only binding to an IPv4 address.

These addresses are defined with a special prefix of length 96 (a.b.c.d is the IPv4 address):

```
0:0:0:0:0:ffff:a.b.c.d/96
```

or in compressed format

```
::ffff:a.b.c.d/96
```

For example, the IPv4 address 1.2.3.4 looks like this:

```
::ffff:1.2.3.4
```

3.1.3.2. IPv4−compatible IPv6 address

Also for sockets, in this case it is for a dual purpose and looks like:

```
0:0:0:0:0:0:a.b.c.d/96
```

or in compressed format

```
::a.b.c.d/96
```

These addresses are also used by automatic tunneling, which is being replaced by 6to4 tunneling.

3.2. Network part, also known as prefix

Designers defined some address types and left a lot of scope for future definitions as currently unknown requirements arise. RFC 2373 [July 1998] / IP Version 6 Addressing Architecture defines the current addressing scheme but there is already a new draft available: draft−ietf−ipngwg−addr−arch−*.txt.

Now lets take a look at the different types of prefixes (and therefore address types):

3.2.1. Link local address type

These are special addresses which will only be valid on a link of an interface. Using this address as destination the packet would never pass through a router. It's used for link communications such as:

- anyone else here on this link?
- anyone here with a special address (e.g. looking for a router)?

They begin with (where *"x"* is any hex character, normally *"0"*)

```
fe8<emphasis>x:  <- currently the only one in use.</emphasis>
fe9<emphasis>x:</emphasis>
fea<emphasis>x:</emphasis>
feb<emphasis>x:</emphasis>
```

An address with this prefix is found on each IPv6−enabled interface after stateless auto−configuration (which is normally always the case).

Note: only fe80 is currently in use.

3.2.2. Site local address type

These are addresses similar to the RFC 1918 / Address Allocation for Private Internets in IPv4 today, with the added advantage that everyone who use this address type has the capability to use the given 16 bits for a maximum number of 65536 subnets. Comparable with the 10.0.0.0/8 in IPv4 today.

Another advantage: because it's possible to assign more than one address to an interface with IPv6, you can also assign such a site local address in addition to a global one.

It begins with:

```
fec<emphasis>x:  <- most commonly used.</emphasis>
fed<emphasis>x:</emphasis>
fee<emphasis>x:</emphasis>
fef<emphasis>x:</emphasis>
```

(where *"x"* is any hex character, normally *"0"*)

3.2.3. Global address type "Aggregatable global unicast"

Today, there is one global address type defined (the first design, called "provider based," was thrown away some years ago RFC 1884 / IP Version 6 Addressing Architecture [obsolete], you will find some remains in older Linux kernel sources).

It begins with (*x* are hex characters)

```
2<emphasis>xxx</emphasis>:
3<emphasis>xxx</emphasis>:
```

There are some further subtypes defined, see below:

3.2.3.1. 6bone test addresses

These were the first global addresses which were defined and in use. They all start with

```
3ffe:
```

Example:

```
3ffe:ffff:100:f102::1
```

A special 6bone test address which will be never be globally unique begins with

```
3ffe:ffff:
```

and is mostly shown in examples, because if real addresses are shown, its possible for someone to do a copy & paste to their configuration files. Thus inadvertently causing duplicates on a globally unique address. This would cause serious problems for the original host (e.g. getting answer packets for request that were never sent). You can still apply for one of these prefixes, see here How to join 6bone. Also some tunnel brokers still distribute 6bone test address prefixes.

3.2.3.2. 6to4 addresses

These addresses, designed for a special tunneling mechanism [RFC 3056 / Connection of IPv6 Domains via IPv4 Clouds and RFC 2893 / Transition Mechanisms for IPv6 Hosts and Routers], encode a given IPv4 address and a possible subnet and begin with

```
2002:
```

For example, representing 192.168.1.1/5:

```
2002:c0a8:0101:5::1
```

A small shell command line can help you generating such address out of a given IPv4 one:

```
ipv4="1.2.3.4"; sla="5"; printf "2002:%02x%02x:%02x%02x:%04x::1" `echo $ipv4 | tr "." " "` $sla
```

See also tunneling using 6to4 and information about 6to4 relay routers.

3.2.3.3. Assigned by provider for hierarchical routing

These addresses are delegated to Internet service providers (ISP) and begin with

```
2001:
```

Prefixes to major (backbone owning) ISPs are delegated by local registries and currently they assign to them a prefix with length 35.

Major ISPs normally delegate to minor ISPs a prefix with length 48.

3.2.4. Multicast addresses

Multicast addresses are used for related services.

They alway start with (*xx* is the scope value)

```
ff<emphasis>x</emphasis>y:
```

They are split into scopes and types:

3.2.4.1. Multicast scopes

Multicast scope is a parameter to specify the maximum distance a multicast packet can travel from the sending entity.

Currently, the following regions (scopes) are defined:

- ffx1: node−local, packets never leave the node.
- ffx2: link−local, packets are never forwarded by routers, so they never leave the specified link.
- ffx5: site−local, packets never leave the site.
- ffx8: organization−local, packets never leave the organization (not so easy to implement, must be covered by routing protocol).
- ffxe: global scope.
- others are reserved

3.2.4.2. Multicast types

There are many types already defined/reserved (see RFC 2373 / IP Version 6 Addressing Architecture for details). Some examples are:

- All Nodes Address: ID = 1h, addresses all hosts on the local node (ff01:0:0:0:0:0:0:1) or the connected link (ff02:0:0:0:0:0:0:1).
- All Routers Address: ID = 2h, addresses all routers on the local node (ff01:0:0:0:0:0:0:2), on the connected link (ff02:0:0:0:0:0:0:2), or on the local site (ff05:0:0:0:0:0:0:2)

3.2.4.3. Solicited node link−local multicast address

Special multicast address used as destination address in neighborhood discovery, because unlike in IPv4, ARP no longer exists in IPv6.

An example of this address looks like

```
ff02::1:ff00:1234
```

Used prefix shows that this is a link−local multicast address. The suffix is generated from the destination address. In this example, a packet should be sent to address "fe80::1234", but the network stack doesn't know the current layer 2 MAC address. It replaces the upper 104 bits with "ff02:0:0:0:0:1:ff00::/104" and leaves the lower 24 bits untouched. This address is now used `on−link' to find the corresponding node which has to send a reply containing its layer 2 MAC address.

3.2.5. Anycast addresses

Anycast addresses are special addresses and are used to cover things like nearest DNS server, nearest DHCP server, or similar dynamic groups. Addresses are taken out of the unicast address space (aggregatable global or site−local at the moment). The anycast mechanism (client view) will be handled by dynamic routing protocols.

Note: Anycast addresses cannot be used as source addresses, they are only used as destination addresses.

3.2.5.1. Subnet−router anycast address

A simple example for an anycast addresses is the subnet−router anycast address. Assuming that a node has the following global assigned IPv6 address:

```
3ffe:ffff:100:f101:210:a4ff:fee3:9566/64  <- Node's address
```

The subnet−router anycast address will be created blanking the suffix (least significant 64 bits) completely:

```
3ffe:ffff:100:f101::/64  <- subnet-router anycast address
```

3.3. Address types (host part)

For auto−configuration and mobility issues, it was decided to use the lower 64 bits as host part of the address in most of the current address types. Therefore each single subnet can hold a large amount of addresses.

This host part can be inspected differently:

3.3.1. Automatically computed (also known as stateless)

With auto−configuration, the host part of the address is computed by converting the MAC address of an interface (if available), with the EUI−64 method, to a unique IPv6 address. If no MAC address is available (happens e.g. on virtual devices), something else (like the IPv4 addresses or the MAC address of a physical interface) is used instead.

Consider again the first example

```
3ffe:ffff:100:f101:210:a4ff:fee3:9566
```

here,

```
210:a4ff:fee3:9566
```

is the host part and computed from the NIC's MAC address

```
00:10:A4:E3:95:66
```

using the IEEE−Tutorial EUI−64 design for EUI−48 identifiers.

3.3.1.1. Privacy problem with automatically computed and solution

Because the "automatically computed" host part is globally unique (except when a vendor of a NIC uses the same MAC address on more than one NIC), client tracking is possible on the host when not using a proxy of any kind.

This is a known problem, and a solution was defined: privacy extension, defined in RFC 3041 / Privacy Extensions for Stateless Address Autoconfiguration in IPv6 (there is also already a newer draft available: draft−ietf−ipngwg−temp−addresses−*.txt). Using a random and a static value a new suffix is generated from time to time. Note: this is only reasonable for outgoing client connections and isn't really useful for well−known servers.

3.3.2. Manually set

For servers it's probably easier to remember simpler addresses, this can also be accommodated. It is possible to assign an additional IPv6 address to an interface, e.g.

```
3ffe:ffff:100:f101::1
```

For manual suffixes like "::1" shown in the above example it's required that the 6th most significant bit is set to 0 (the universal/local bit of the automatically generated identifier). Also some other (otherwise unchosen) bit combinations are reserved for anycast addresses, too.

3.4. Prefix lengths for routing

In the early design phase it was planned to use a fully hierarchical routing approach to reduce the size of the routing tables maximally. The reasoning behind this approach were the number of current IPv4 routing entries in core routers (> 104 thousand in May 2001), reducing the need of memory in hardware routers (ASIC driven) to hold the routing table and increase speed (fewer entries hopefully result in faster lookups).

Todays view is that routing will be mostly hierarchically designed for networks with only one service provider. With more than one ISP connections, this is not possible, and subject to an issue named multi−homing.

3.4.1. Prefix lengths (also known as "netmasks")

Similar to IPv4, the routable network path for routing to take place. Because standard netmask notation for 128 bits doesn't look nice, designers employed the IPv4 Classless Inter Domain Routing (CIDR, RFC 1519 / Classless Inter−Domain Routing) scheme, which specifies the number of bits of the IP address to be used for routing. It is also called the "slash" notation.

An example:

```
3ffe:ffff:100:1:2:3:4:5/48
```

This notation will be expanded:

- Network:

```
3ffe:ffff:0100:0000:0000:0000:0000:0000
```

- Net−mask:

```
ffff:ffff:ffff:0000:0000:0000:0000:0000
```

3.4.2. Matching a route

Under normal circumstances (no QoS) a lookup in a routing table results in the route with the most significant number of address bits means the route with the biggest prefix length matches first.

For example if a routing table shows following entries (list is not complete):

```
3ffe:ffff:100::/48     ::             U  1 0 0 sit1
3ffe::/16              ::192.88.99.1 UG 1 0 0 tun6to4
2000::/3               ::192.88.99.1 UG 1 0 0 tun6to4
```

Shown destination addresses of IPv6 packets will be routed through shown device

```
3ffe:ffff:100:1:2:3:4:5/48  ->  routed through device sit1
3ffe:ffff:200:1:2:3:4:5/48  ->  routed through device tun6to4
```

Chapter 4. IPv6−ready system check

Before you can start using IPv6 on a Linux host, you have to test, whether your system is IPv6−ready. You may have to do some work to enable it first.

4.1. IPv6−ready kernel

Modern Linux distributions already contain IPv6−ready kernels, the IPv6 capability is generally compiled as a module, but it's possible that this module is not loaded automatically on startup.

See IPv6+Linux−Status−Distribution page for most up−to−date information.

4.1.1. Check for IPv6 support in the current running kernel

To check, whether your current running kernel supports IPv6, take a look into your /proc−file−system. Following entry must exists:

```
/proc/net/if_inet6
```

A short automatical test looks like:

```
# test −f /proc/net/if_inet6 && echo "Running kernel is IPv6 ready"
```

If this fails, it is quite likely, that the IPv6 module is not loaded.

4.1.2. Try to load IPv6 module

You can try to load the IPv6 module executing

```
# modprobe ipv6
```

If this is successful, this module should be listed, testable with following auto−magically line:

```
# lsmod |grep −w 'ipv6' && echo "IPv6 module successfully loaded"
```

And the check shown above should now run successfully.

Note: unloading the module is currently not supported and can result, under some circumstances, in a kernel crash.

4.1.2.1. Automatically loading of module

Its possible to automatically load the IPv6 module on demand. You only have to add following line in the configuration file of the kernel module loader (normally /etc/modules.conf or /etc/conf.modules):

```
alias net-pf-10 ipv6  # automatically load IPv6 module on demand
```

It's also possible to disable automatically loading of the IPv6 module using following line

```
alias net-pf-10 off   # disable automatically load of IPv6 module on demand
```

4.1.3. Compile kernel with IPv6 capabilities

If both above shown results were negative and your kernel has no IP6 support, than you have the following options:

- Update your distribution to a current one which supports IPv6 out−of−the−box (recommended for newbies), see here again: IPv6+Linux−Status−Distribution
- Compile a new vanilla kernel (easy, if you know which options you needed)
- Recompile kernel sources given by your Linux distribution (sometimes not so easy)
- Compile a kernel with USAGI extensions

If you decide to compile a kernel, you should have previous experience in kernel compiling and read the Linux Kernel HOWTO.

A mostly up−to−time comparison between vanilla and USAGI extended kernels is available on IPv6+Linux−Status−Kernel.

4.1.3.1. Compiling a vanilla kernel

More detailed hints about compiling an IPv6−enabled kernel can be found e.g. on IPv6−HOWTO−2#kernel.

Note: you should use whenever possible kernel series 2.4.x or above, because the IPv6 support in series 2.2.x is not so in current state and needs some patches for ICMPv6 and 6to4 support (can be found on kernel series 2.2.x IPv6 patches).

4.1.3.2. Compiling a kernel with USAGI extensions

Same as for vanilla kernel, only recommend for advanced users, which are already familiar with IPv6 and kernel compilation. See also USAGI project / FAQ.

4.1.4. IPv6−ready network devices

Not all existing network devices have already (or ever) the capability to transport IPv6 packets. A current status can be found at IPv6+Linux−status−kernel.html#transport.

A major issue is that because of the network layer structure of kernel implementation an IPv6 packet isn't really recognized by it's IP header number (6 instead of 4). It's recognized by the protocol number of the Layer 2 transport protocol. Therefore any transport protocol which doesn't use such protocol number cannot dispatch IPv6 packets. Note: the packet is still transported over the link, but on receivers side, the dispatching won't work (you can see this e.g. using tcpdump).

4.1.4.1. Currently known never "IPv6 capable links"

- Serial Line IP (SLIP, RFC 1055), should be better called now to SLIPv4, device named: slX
- Parallel Line IP (PLIP), same like SLIP, device names: plipX
- ISDN with encapsulation *rawip*, device names: isdnX

4.1.4.2. Currently known "not supported IPv6 capable links"

- ISDN with encapsulation *syncppp*, device names: ipppX (design issue of the ipppd, will be merged into more general PPP layer in kernel series 2.5.x)

4.2. IPv6−ready network configuration tools

You wont get very far, if you are running an IPv6−ready kernel, but have no tools to configure IPv6. There are several packages in existence which can configure IPv6.

4.2.1. net−tools package

The net−tool package includes some tools like ifconfig and route, which helps you to configure IPv6 on an interface. Look at the output of ifconfig −? or route −?, if something is shown like IPv6 or inet6, then the tool is IPv6−ready.

Auto−magically check:

```
# /sbin/ifconfig -? 2>& 1|grep -qw 'inet6' && echo "utility 'ifconfig' is
¬ IPv6-ready"
```

Same check can be done for route:

```
# /sbin/route -? 2>& 1|grep -qw 'inet6' && echo "utility 'route' is IPv6-ready"
```

4.2.2. iproute package

Alexey N. Kuznetsov (current a maintainer of the Linux networking code) created a tool−set which configures networks through the netlink device. Using this tool−set you have more functionality than net−tools provides, but its not very well documented and isn't for the faint of heart.

```
# /sbin/ip 2>&1 |grep −qw 'inet6' && echo "utility 'ip' is IPv6−ready"
```

If the program /sbin/ip isn't found, then I strongly recommend you install the iproute package.

- You can get it from your Linux distribution (if contained)
- You can download the tar−ball and recompile it: Original FTP source and mirror (missing)
- You've able to look for a proper RPM package at RPMfind/iproute (sometimes rebuilding of a SRPMS package is recommended)

4.3. IPv6−ready test/debug programs

After you have prepared your system for IPv6, you now want to use IPv6 for network communications. First you should learn how to examine IPv6 packets with a sniffer program. This is strongly recommended because for debugging/troubleshooting issues this can aide in providing a diagnosis very quickly.

4.3.1. IPv6 ping

This program is normally included in package *iputils*. It is designed for simple transport tests sending ICMPv6 echo−request packets and wait for ICMPv6 echo−reply packets.

Usage

```
# ping6 <hostwithipv6address>
# ping6 <ipv6address>
# ping6 [−I <device>] <link−local−ipv6address>
```

Example

```
# ping6 −c 1 ::1
PING ::1(::1) from ::1 : 56 data bytes
64 bytes from ::1: icmp_seq=0 hops=64 time=292 usec
--- ::1 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round−trip min/avg/max/mdev = 0.292/0.292/0.292/0.000 ms
```

Hint: ping6 needs raw access to socket and therefore root permissions. So if non−root users cannot use ping6 then there are two possible problems:

1. ping6 is not in users path (probably, because ping6 is generally stored in /usr/sbin −> add path (not really recommended)

2. ping6 doesn't execute properly, generally because of missing root permissions −> chmod u+s /usr/sbin/ping6

4.3.1.1. Specifying interface for IPv6 ping

Using link−local addresses for an IPv6 ping, the kernel does not know through which (physically or virtual) device it must send the packet − each device has a link−local address. A try will result in following error message:

```
# ping6 fe80::212:34ff:fe12:3456
connect: Invalid argument
```

In this case you have to specify the interface additionally like shown here:

```
# ping6 −I eth0 −c 1 fe80::2e0:18ff:fe90:9205
PING fe80::212:23ff:fe12:3456(fe80::212:23ff:fe12:3456) from
¬ fe80::212:34ff:fe12:3478 eth0: 56 data bytes
64 bytes from fe80::212:23ff:fe12:3456: icmp_seq=0 hops=64 time=445 usec
--- fe80::2e0:18ff:fe90:9205 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss round-trip
¬ min/avg/max/mdev = 0.445/0.445/0.445/0.000 ms
```

4.3.2. IPv6 traceroute6

This program is normally included in package *iputils*. It's a program similar to IPv4 traceroute. But unlike modern IPv4 versions, the IPv6 one still doesn't understand to traceroute using ICMP echo−request packets (which is more accepted by firewalls around than UDP packets to high ports). Below you will see an example:

```
# traceroute6 www.6bone.net
traceroute to 6bone.net (3ffe:b00:c18:1::10) from 3ffe:ffff:0000:f101::2, 30
¬ hops max, 16 byte packets
 1 localipv6gateway (3ffe:ffff:0000:f101::1) 1.354 ms 1.566 ms 0.407 ms
 2 swi6T1−T0.ipv6.switch.ch (3ffe:2000:0:400::1) 90.431 ms 91.956 ms 92.377 ms
 3 3ffe:2000:0:1::132 (3ffe:2000:0:1::132) 118.945 ms 107.982 ms 114.557 ms
 4 3ffe:c00:8023:2b::2 (3ffe:c00:8023:2b::2) 968.468 ms 993.392 ms 973.441 ms
 5 3ffe:2e00:e:c::3 (3ffe:2e00:e:c::3) 507.784 ms 505.549 ms 508.928 ms
 6 www.6bone.net (3ffe:b00:c18:1::10) 1265.85 ms * 1304.74 ms
```

4.3.3. IPv6 tracepath6

This program is normally included in package *iputils*. It's a program like traceroute6 and traces the path to a given destination discovering the MTU along this path. Below you will see an example:

```
# tracepath6 www.6bone.net
 1?: [LOCALHOST] pmtu 1480
 1: 3ffe:401::2c0:33ff:fe02:14 150.705ms
 2: 3ffe:b00:c18::5 267.864ms
```

```
 3: 3ffe:b00:c18::5 asymm 2 266.145ms pmtu 1280
 3: 3ffe:3900:5::2 asymm 4 346.632ms
 4: 3ffe:28ff:ffff:4::3 asymm 5 365.965ms
 5: 3ffe:1cff:0:ee::2 asymm 4 534.704ms
 6: 3ffe:3800::1:1 asymm 4 578.126ms !N
Resume: pmtu 1280
```

4.3.4. IPv6 tcpdump

On Linux, tcpdump is the major tool for packet capturing. Below you find some examples. IPv6 support is normally built−in in current releases of version 3.6.

tcpdump uses expressions for filtering packets to minimize the noise:

- icmp6: filters native ICMPv6 traffic
- ip6: filters native IPv6 traffic (including ICMPv6)
- proto ipv6: filters tunneled IPv6−in−IPv4 traffic
- not port ssh: to suppress displaying SSH packets for running tcpdump in a remote SSH session

Also some command line options are very useful to catch and print more information in a packet, mostly interesting for digging into ICMPv6 packets:

- "−s 512": increase the snap length during capturing of a packet to 512 bytes
- "−vv": really verbose output
- "−n": don't resolve addresses to names, useful if reverse DNS resolving isn't working proper

4.3.4.1. IPv6 ping to 3ffe:ffff:100:f101::1 native over a local link

```
# tcpdump −t −n −i eth0 −s 512 −vv ip6 or proto ipv6
tcpdump: listening on eth0
3ffe:ffff:100:f101:2e0:18ff:fe90:9205 > 3ffe:ffff:100:f101::1: icmp6: echo
¬ request (len 64, hlim 64)
3ffe:ffff:100:f101::1 > 3ffe:ffff:100:f101:2e0:18ff:fe90:9205: icmp6: echo
¬ reply (len 64, hlim 64)
```

4.3.4.2. IPv6 ping to 3ffe:ffff:100::1 routed through an IPv6−in−IPv4−tunnel

1.2.3.4 and 5.6.7.8 are tunnel endpoints (all addresses are examples)

```
# tcpdump −t −n −i ppp0 −s 512 −vv ip6 or proto ipv6
tcpdump: listening on ppp0
1.2.3.4 > 5.6.7.8: 2002:ffff:f5f8::1 > 3ffe:ffff:100::1: icmp6: echo request
¬ (len 64, hlim 64) (DF) (ttl 64, id 0, len 124)
5.6.7.8 > 1.2.3.4: 3ffe:ffff:100::1 > 2002:ffff:f5f8::1: icmp6: echo reply (len
¬ 64, hlim 61) (ttl 23, id 29887, len 124)
1.2.3.4 > 5.6.7.8: 2002:ffff:f5f8::1 > 3ffe:ffff:100::1: icmp6: echo request
¬ (len 64, hlim 64) (DF) (ttl 64, id 0, len 124)
5.6.7.8 > 1.2.3.4: 3ffe:ffff:100::1 > 2002:ffff:f5f8::1: icmp6: echo reply (len
¬ 64, hlim 61) (ttl 23, id 29919, len 124)
```

4.4. IPv6−ready programs

Current distributions already contain the most needed IPv6 enabled client and servers. See first on IPv6+Linux−Status−Distribution. If still not included, you can check IPv6 & Linux − Current Status − Applications whether the program is already ported to IPv6 and usable with Linux. For common used programs there are some hints available at IPv6 & Linux − HowTo − Part 3 and IPv6 & Linux − HowTo − Part 4.

4.5. IPv6−ready client programs (selection)

To run the following shown tests, it's required that your system is IPv6 enabled, and some examples show addresses which only can be reached if a connection to the 6bone is available.

4.5.1. Checking DNS for resolving IPv6 addresses

Because of security updates in the last years every Domain Name System (DNS) server should run newer software which already understands the (intermediate) IPv6 address−type AAAA (the newer one named A6 isn't still common at the moment because only supported using BIND9 and newer and also the non−existent support of root domain IP6.ARPA). A simple whether the used system can resolve IPv6 addresses is

```
# host −t AAAA www.join.uni−muenster.de
```

and should show something like following:

```
www.join.uni−muenster.de. is an alias for ns.join.uni−muenster.de.
ns.join.uni−muenster.de. has AAAA address 3ffe:400:10:100:201:2ff:feb5:3806
```

4.5.2. IPv6−ready telnet clients

IPv6−ready telnet clients are available. A simple test can be done with

```
$ telnet 3ffe:400:100::1 80
Trying 3ffe:400:100::1...
Connected to 3ffe:400:100::1.
Escape character is '^]'.
HEAD / HTTP/1.0
HTTP/1.1 200 OK
Date: Sun, 16 Dec 2001 16:07:21
GMT Server: Apache/2.0.28 (Unix)
Last−Modified: Wed, 01 Aug 2001 21:34:42 GMT
ETag: "3f02−a4d−b1b3e080"
Accept−Ranges: bytes
Content−Length: 2637
```

```
Connection: close
Content-Type: text/html; charset=ISO-8859-1
Connection closed by foreign host.
```

If the telnet client don't understand the IPv6 address and says something like "cannot resolve hostname", then it's not IPv6–enabled.

4.5.3. IPv6–ready ssh clients

4.5.3.1. openssh

Current versions of openssh are IPv6–ready. Depending on configuring before compiling it has two behavior.

* ––without–ipv4–default: the client tries an IPv6 connect first automatically and fall back to IPv4 if not working
* ––with–ipv4–default: default connection is IPv4, IPv6 connection must be force like following example shows

```
$ ssh -6 ::1
user@::1's password: ******
[user@ipv6host user]$
```

If your ssh client doesn't understand the option "–6" then it's not IPv6–enabled, like most ssh version 1 packages.

4.5.3.2. ssh.com

SSH.com's SSH client and server is also IPv6 aware now and is free for all Linux and FreeBSD machine regardless if used for personal or commercial use.

4.5.4. IPv6–ready web browsers

A current status of IPv6 enabled web browsers is available at IPv6+Linux–status–apps.html#HTTP.

Most of them have unresolved problems at the moment

1. If using an IPv4 only proxy in the settings, IPv6 requests will be sent to the proxy, but the proxy will fail to understand the request and the request fails. Solution: update proxy software (see later).
2. Automatic proxy settings (*.pac) cannot be extended to handle IPv6 requests differently (e.g. don't use proxy) because of their nature (written in Java–script and well hard coded in source like to be seen in Maxilla source code).

Also older versions don't understand an URL with IPv6 encoded addresses like http://[3ffe:400:100::1]/ (this given URL only works with an IPv6–enabled browser!).

A short test is to try shown URL with a given browser and using no proxy.

4.5.4.1. URLs for testing

A good starting point for browsing using IPv6 is http://www.kame.net/. If the turtle on this page is animated, the connection is via IPv6, otherwise the turtle is static.

4.6. IPv6−ready server programs

In this part of this HOWTO, more client specific issues are mentioned. Therefore hints for IPv6−ready servers like sshd, httpd, telnetd, etc. are shown below in Hints for IPv6−enabled daemons.

4.7. FAQ (IPv6−ready system check)

4.7.1. Using tools

4.7.1.1. Q: Cannot ping6 to link−local addresses

Error message: "*connect: Invalid argument*"

Kernel doesn't know, which physical or virtual link you want to use to send such ICMPv6 packets. Therefore it displays this error message.

Solution: Specify interface like: "ping6 −I eth0 fe80::2e0:18ff:fe90:9205", see also program ping6 usage.

4.7.1.2. Q: Cannot ping6 or traceroute6 as normal user

Error message: "*icmp socket: Operation not permitted*"

These utilities create special ICMPv6 packets and send them out. This is done by using raw sockets in the kernel. But raw sockets can only be used by the "root" user. Therefore normal users get such error message.

Solution: If it's really needed that all users should be able to use this utilities, you can add the "suid" bit using "chmod u+s /path/to/program", see also program ping6 usage. If not all users should be able to, you can change the group of the program to e.g. "wheel", add this power users to this group and remove the execution bit for other users using "chmod o−rwx /path/to/program". Or configure "sudo" to enable your security policy.

Chapter 5. Configuring interfaces

5.1. Different network devices

On a node, there exist different network devices. They can be collected in classes

- Physically bounded, like eth0, tr0
- Virtually existing, like ppp0, tun0, tap0, sit0, isdn0, ippp0

5.1.1. Physically bounded

Physically bounded interfaces like Ethernet or Token−Ring are normal ones and need no special treatment.

5.1.2. Virtually bounded

Virtually bounded interfaces always need special support

5.1.2.1. IPv6−in−IPv4 tunnel interfaces

This interfaces are normally named sit*x*. The name *sit* is a shortcut for Simple Internet Transition. This device has the capability to encapsulate IPv6 packets into IPv4 ones and tunnel them to a foreign endpoint.

sit0 has a special meaning and cannot be used for dedicated tunnels.

5.1.2.2. PPP interfaces

PPP interfaces get their IPv6 capability from an IPv6 enabled PPP daemon.

5.1.2.3. ISDN HDLC interfaces

IPv6 capability for HDLC with encapsulation ip is already built−in in the kernel

5.1.2.4. ISDN PPP interfaces

ISDN PPP interfaces (ippp) aren't IPv6 enabled by kernel. Also there are also no plans to do that because in kernel 2.5.+ they will be replaced by a more generic ppp interface layer.

5.1.2.5. SLIP + PLIP

Like mentioned earlier, this interfaces don't support IPv6 transport (sending is OK, but dispatching on receiving don't work).

5.1.2.6. Ether−tap device

Ether−tap devices are IPv6−enabled and also stateless configured. For use, the module "ethertap" has to be loaded before.

5.1.2.7. tun devices

Currently not tested by me.

5.1.2.8. ATM

01/2002: Aren't currently supported by vanilla kernel, supported by USAGI extension

5.1.2.9. Others

Did I forget an interface?...

5.2. Bringing interfaces up/down

Two methods can be used to bring interfaces up or down.

5.2.1. Using "ip"

Usage:

```
# ip link set dev <interface> up
# ip link set dev <interface> down
```

Example:

```
# ip link set dev eth0 up
# ip link set dev eth0 down
```

5.2.2. Using "ifconfig"

Usage:

```
# /sbin/ifconfig <interface> up
# /sbin/ifconfig <interface> down
```

Example:

```
# /sbin/ifconfig eth0 up
# /sbin/ifconfig eth0 down
```

Chapter 6. Configuring IPv6 addresses

There are different ways to configure an IPv6 address on an interface. You can use use "ifconfig" or "ip".

6.1. Displaying existing IPv6 addresses

First you should check, whether and which IPv6 addresses are already configured (perhaps auto−magically during stateless auto−configuration).

6.1.1. Using "ip"

Usage:

```
# /sbin/ip −6 addr show dev <interface>
```

Example for a static configured host:

```
# /sbin/ip −6 addr show dev eth0
2: eth0: <BROADCAST,MULTICAST,UP&gt; mtu 1500 qdisc pfifo_ fast qlen 100
inet6 fe80::210:a4ff:fee3:9566/10 scope link
inet6 3ffe:ffff:0:f101::1/64 scope global
inet6 fec0:0:0:f101::1/64 scope site
```

Example for a host which is auto−configured

Here you see some auto−magically configured IPv6 addresses and their lifetime.

```
# /sbin/ip −6 addr show dev eth0
3: eth0: <BROADCAST,MULTICAST,PROMISC,UP&gt; mtu 1500 qdisc pfifo_fast qlen
¬ 100
inet6 2002:d950:f5f8:f101:2e0:18ff:fe90:9205/64 scope global dynamic
valid_lft 16sec preferred_lft 6sec
inet6 3ffe:400:100:f101:2e0:18ff:fe90:9205/64 scope global dynamic
valid_lft 2591997sec preferred_lft 604797sec inet6 fe80::2e0:18ff:fe90:9205/10
¬ scope link
```

6.1.2. Using "ifconfig"

Usage:

```
# /sbin/ifconfig <interface>
```

Example (output filtered with grep to display only IPv6 addresses). Here you see different IPv6 addresses with different scopes.

```
# /sbin/ifconfig eth0 |grep "inet6 addr:"
inet6 addr: fe80::210:a4ff:fee3:9566/10 Scope:Link
inet6 addr: 3ffe:ffff:0:f101::1/64 Scope:Global
inet6 addr: fec0:0:0:f101::1/64 Scope:Site
```

6.2. Add an IPv6 address

Adding an IPv6 address is similar to the mechanism of "IP ALIAS" addresses in Linux IPv4 addressed interfaces.

6.2.1. Using "ip"

Usage:

```
# /sbin/ip -6 addr add <ipv6address>/<prefixlength> dev <interface>
```

Example:

```
# /sbin/ip -6 addr add 3ffe:ffff:0:f101::1/64 dev eth0
```

6.2.2. Using "ifconfig"

Usage:

```
# /sbin/ifconfig <interface> inet6 add <ipv6address>/<prefixlength>
```

Example:

```
# /sbin/ifconfig eth0 inet6 add 3ffe:ffff:0:f101::1/64
```

6.3. Removing an IPv6 address

Not so often needed, be carefully with removing non existent IPv6 address, sometimes using older kernels it results in a crash.

6.3.1. Using "ip"

Usage:

```
# /sbin/ip -6 addr del <ipv6address>/<prefixlength> dev <interface>
```

Example:

```
# /sbin/ip -6 addr del 3ffe:ffff:0:f101::1/64 dev eth0
```

6.3.2. Using "ifconfig"

Usage:

```
# /sbin/ifconfig <interface> inet6 del <ipv6address>/<prefixlength>
```

Example:

```
# /sbin/ifconfig eth0 inet6 del 3ffe:ffff:0:f101::1/64
```

Chapter 7. Configuring normal IPv6 routes

If you want to leave your link and want to send packets in the world wide IPv6−Internet, you need routing. If there is already an IPv6 enabled router on your link, it's possible enough to add IPv6 routes.

Also here there are different ways to configure an IPv6 address on an interface. You can use use "ifconfig" or "ip"

7.1. Displaying existing IPv6 routes

First you should check, whether and which IPv6 addresses are already configured (perhaps auto−magically during auto−configuration).

7.1.1. Using "ip"

Usage:

```
# /sbin/ip −6 route show [dev <device>]
```

Example:

```
# /sbin/ip −6 route show dev eth0
3ffe:ffff:0:f101::/64 proto kernel metric 256 mtu 1500 advmss 1440
fe80::/10             proto kernel metric 256 mtu 1500 advmss 1440
ff00::/8              proto kernel metric 256 mtu 1500 advmss 1440
default               proto kernel metric 256 mtu 1500 advmss 1440
```

7.1.2. Using "route"

Usage:

```
# /sbin/route −A inet6
```

Example (output is filtered for interface eth0). Here you see different IPv6 routes for different addresses on a single interface.

```
# /sbin/route −A inet6 |grep −w "eth0"
3ffe:ffff:0:f101 ::/64 :: UA  256 0 0 eth0 <− Interface route for global
¬ address
fe80::/10        ::       UA  256 0 0 eth0 <− Interface route for link−local
¬ address
ff00::/8         ::       UA  256 0 0 eth0 <− Interface route for all multicast
¬ addresses
::/0             ::       UDA 256 0 0 eth0 <− Automatic default route
```

7.2. Add an IPv6 route through a gateway

Mostly needed to reach the outside with IPv6 using an IPv6−enabled router on your link.

7.2.1. Using "ip"

Usage:

```
# /sbin/ip −6 route add <ipv6network>/<prefixlength> via <ipv6address>
¬ [dev <device>]
```

Example:

```
# /sbin/ip −6 route add 2000::/3 via 3ffe:ffff:0:f101::1
```

7.2.2. Using "route"

Usage:

```
# /sbin/route −A inet6 add <ipv6network>/<prefixlength> gw
¬ <ipv6address> [dev <device>]
```

A device can be needed, too, if the IPv6 address of the gateway is a link local one.

Following shown example adds a route for all currently global addresses (2000::/3) through gateway 3ffe:ffff:0:f101::1

```
# /sbin/route −A inet6 add 2000::/3 gw 3ffe:ffff:0:f101::1
```

7.3. Removing an IPv6 route through a gateway

Not so often needed manually, mostly done by network configure scripts on shutdown (full or per interface)

7.3.1. Using "ip"

Usage:

```
# /sbin/ip −6 route del <ipv6network>/<prefixlength> via <ipv6address>
¬ [dev <device>]
```

Example:

```
# /sbin/ip -6 route del 2000::/3 via 3ffe:ffff:0:f101::1
```

7.3.2. Using "route"

Usage:

```
# /sbin/route -A inet6 del <network>/<prefixlength> [dev <device>]
```

Example for removing upper added route again:

```
# /sbin/route -A inet6 del 2000::/3 gw 3ffe:ffff:0:f101::1
```

7.4. Add an IPv6 route through an interface

Not often needed, sometimes in cases of dedicated point−to−point links.

7.4.1. Using "ip"

Usage:

```
# /sbin/ip -6 route add <ipv6network>/<prefixlength> dev <device>
¬ metric 1
```

Example:

```
# /sbin/ip -6 route add 2000::/3 dev eth0 metric 1
```

Metric "1" is used here to be compatible with the metric used by route, because the default metric on using "ip" is "1024".

7.4.2. Using "route"

Usage:

```
# /sbin/route -A inet6 add <network>/<prefixlength> dev <device>
```

Example:

```
# /sbin/route -A inet6 add 2000::/3 dev eth0
```

7.5. Removing an IPv6 route through an interface

Not so often needed to use by hand, configuration scripts will use such on shutdown.

7.5.1. Using "ip"

Usage:

```
# /sbin/ip -6 route del <ipv6network>/<prefixlength> dev <device>
```

Example:

```
# /sbin/ip -6 route del 2000::/3 dev eth0
```

7.5.2. Using "route"

Usage:

```
# /sbin/route -A inet6 del <network>/<prefixlength> dev <device>
```

Example:

```
# /sbin/route -A inet6 del 2000::/3 dev eth0
```

7.6. FAQ for IPv6 routes

7.6.1. Support of an IPv6 default route

One idea of IPv6 was a hierachical routing, therefore only less routing entries are needed in routers.

There are some issues in current Linux kernels:

7.6.1.1. Clients

Client can setup a default route like prefix "::/0", they also learn such route on autoconfiguration e.g. using radvd on the link like following example shows:

```
# ip -6 route show | grep ^default
default via fe80::212:34ff:fe12:3450 dev eth0 proto kernel metric 1024 expires
¬ 29sec mtu 1500 advmss 1440
```

7.6.1.2. Routers on packet forwarding

Current mainstream Linux kernel (at least <= 2.4.17) don't support default routes. You can set them up, but the route lookup fails when a packet should be forwarded (normal intention of a router).

Therefore at this time "default routing" can be setup using the currently only global address prefix "2000::/3".

The USAGI project already supports this in their extension with a hack.

Note: take care about default routing without address filtering on edge routers. Otherwise unwanted multicast or site−local traffic leave the edge.

Chapter 8. Configuring IPv6−in−IPv4 tunnels

If you want to leave your link you have no IPv6 capable network around you, you need IPv6−in−IPv4 tunneling to reach the World Wide IPv6−Internet.

There are some kind of tunnel mechanism and also some possibilities to setup tunnels.

8.1. Types of tunnels

There are more than one possibility to tunnel IPv6 packets over IPv4−only links.

8.1.1. Static point−to−point tunneling: 6bone

A point−to−point tunnel is a dedicated tunnel to an endpoint, which knows about your IPv6 network (for backward routing) and the IPv4 address of your tunnel endpoint and defined in RFC 2893 / Transition Mechanisms for IPv6 Hosts and Routers. Requirements:

- IPv4 address of your local tunnel endpoint must be static, global unique and reachable from the foreign tunnel endpoint
- A global IPv6 prefix assigned to you (see 6bone registry)
- A foreign tunnel endpoint which is capable to route your IPv6 prefix to your local tunnel endpoint (mostly remote manual configuration required)

8.1.2. Automatically tunneling

Automatic tunneling occurs, when a node directly connects another node gotten the IPv4 address of the other node before.

8.1.3. 6to4−Tunneling

6to4 tunneling (RFC 3056 / Connection of IPv6 Domains via IPv4 Clouds) uses a simple mechanism to create automatic tunnels. Each node with a global unique IPv4 address is able to be a 6to4 tunnel endpoint (if no IPv4 firewall prohibits traffic). 6to4 tunneling is mostly not a one−to−one tunnel. This case of tunneling can be divided into upstream and downstream tunneling. Also, a special IPv6 address indicates that this node will use 6to4 tunneling for connecting the world−wide IPv6 network

8.1.3.1. Generation of 6to4 prefix

The 6to4 address is defined like following (schema is taken from RFC 3056 / Connection of IPv6 Domains via IPv4 Clouds):

```
|   3+13   |    32     |    16  |            64 bits             |
+---+------+-----------+--------+--------------------------------+
```

```
|  FP+TLA  |  V4ADDR   | SLA ID |           Interface ID          |
|  0x2002  |           |        |                                 |
+---+------+-----------+--------+---------------------------------+
```

Where FP is the known prefix for global addresses, TLA is the top level aggregator. V4ADDR is the node's global unique IPv4 address (in hexadecimal notation). SLA is the subnet identifier (65536 local subnets possible).

Such prefix is generated and normally using SLA "0000" and suffix "::1" assigned to the 6to4 tunnel interface.

8.1.3.2. 6to4 upstream tunneling

The node has to know to which foreign tunnel endpoint its in IPv4 packed IPv6 packets should be send to. In "early" days of 6to4 tunneling, dedicated upstream accepting routers were defined. See NSayer's 6to4 information for a list of routers.

Nowadays, 6to4 upstream routers can be found auto−magically using the anycast address 192.88.99.1. In the background routing protocols handle this, see RFC 3068 / An Anycast Prefix for 6to4 Relay Routers for details.

8.1.3.3. 6to4 downstream tunneling

The downstream (6bone −> your 6to4 enabled node) is not really fix and can vary from foreign host which originated packets were send to. There exist two possibilities:

- Foreign host uses uses 6to4 and sends packet direct back to your node (see below)
- Foreign host sends packets back to the world−wide IPv6 network and depending on the dynamic routing a relay router create a automatic tunnel back to your node.

8.1.3.4. Possible 6to4 traffic

- from 6to4 to 6to4: is normally directly tunneled between the both 6to4 enabled hosts
- from 6to4 to non−6to4: is sent via upstream tunneling
- non−6to4 to 6to4: is sent via downstream tunneling

8.2. Displaying existing tunnels

8.2.1. Using "ip"

Usage:

```
# /sbin/ip −6 tunnel show [<device>]
```

Example:

```
# /sbin/ip -6 tunnel show
sit0: ipv6/ip remote any local any ttl 64 nopmtudisc
sit1: ipv6/ip remote 195.226.187.50 local any ttl 64
```

8.2.2. Using "route"

Usage:

```
# /sbin/route -A inet6
```

Example (output is filtered to display only tunnels through virtual interface sit0):

```
# /sbin/route -A inet6 | grep "\Wsit0\W*$"
::/96      ::              U    256  2  0  sit0
2002::/16  ::              UA   256  0  0  sit0
2000::/3   ::193.113.58.75  UG     1  0  0  sit0
fe80::/10  ::              UA   256  0  0  sit0
ff00::/8   ::              UA   256  0  0  sit0
```

8.3. Setup of point−to−point tunnel

There are 3 possibilities to add or remove point−to−point tunnels.

8.3.1. Add point−to−point tunnels

8.3.1.1. Using "ip" and "route"

Common method at the moment for a small amount of tunnels

Usage for creating a tunnel device (but it's not up afterward, also a TTL must be specified because the default value is 0).

```
# /sbin/ip tunnel add <device> mode sit ttl <ttldefault> remote
¬ <ipv4addressofforeigntunnel> local <ipv4addresslocal>
```

Usage (generic example for three tunnels):

```
# /sbin/ip tunnel add sit1 mode sit ttl <ttldefault> remote
¬ <ipv4addressofforeigntunnel1> local <ipv4addresslocal>
# /sbin/ifconfig sit1 up
# /sbin/route -A inet6 add <prefixtoroute1> dev sit1
# /sbin/ip tunnel add sit2 mode sit ttl <ttldefault>
¬ <ipv4addressofforeigntunnel2> local <ipv4addresslocal>
# /sbin/ifconfig sit2 up
```

```
# /sbin/route -A inet6 add <prefixtoroute2> dev sit2
# /sbin/ip tunnel add sit3 mode sit ttl <ttldefault>
¬ <ipv4addressofforeigntunnel3> local <ipv4addresslocal>
# /sbin/ifconfig sit3 up
# /sbin/route -A inet6 add <prefixtoroute3> dev sit3
```

8.3.1.2. Using "ifconfig" and "route" (deprecated)

This not very recommended way to add a tunnel because it's a little bit strange. No problem if adding only one, but if you setup more than one, you cannot easy shutdown the first ones and leave the others running.

Usage (generic example for three tunnels):

```
# /sbin/ifconfig sit0 up
# /sbin/ifconfig sit0 tunnel <ipv4addressofforeigntunnel1>
# /sbin/ifconfig sit1 up
# /sbin/route -A inet6 add <prefixtoroute1> dev sit1
# /sbin/ifconfig sit0 tunnel <ipv4addressofforeigntunnel2>
# /sbin/ifconfig sit2 up
# /sbin/route -A inet6 add <prefixtoroute2> dev sit2
# /sbin/ifconfig sit0 tunnel <ipv4addressofforeigntunnel3>
# /sbin/ifconfig sit3 up
# /sbin/route -A inet6 add <prefixtoroute3> dev sit3
```

Important: DON'T USE THIS, because this setup implicit enable "automatic tunneling" from anywhere in the Internet, this is a risk, and it should not be advocated.

8.3.1.3. Using "route" only

It's also possible to setup tunnels in Non Broadcast Multiple Access (NBMA) style, it's a easy way to add many tunnels at once. But none of the tunnel can be numbered (which is a not required feature).

Usage (generic example for three tunnels):

```
# /sbin/ifconfig sit0 up
# /sbin/route -A inet6 add <prefixtoroute1> gw
¬ ::<ipv4addressofforeigntunnel1> dev sit0
# /sbin/route -A inet6 add <prefixtoroute2> gw
¬ ::<ipv4addressofforeigntunnel2> dev sit0
# /sbin/route -A inet6 add <prefixtoroute3> gw
¬ ::<ipv4addressofforeigntunnel3> dev sit0
```

Important: DON'T USE THIS, because this setup implicit enable "automatic tunneling" from anywhere in the Internet, this is a risk, and it should not be advocated.

8.3.2. Removing point-to-point tunnels

Manually not so often needed, but used by scripts for clean shutdown or restart of IPv6 configuration.

8.3.2.1. Using "ip" and "route"

Usage for removing a tunnel device:

```
# /sbin/ip tunnel del <device>
```

Usage (generic example for three tunnels):

```
# /sbin/route −A inet6 del <prefixtoroute1> dev sit1
# /sbin/ifconfig sit1 down
# /sbin/ip tunnel del sit1
# /sbin/route −A inet6 del <prefixtoroute2> dev sit2
# /sbin/ifconfig sit2 down
# /sbin/ip tunnel del sit2
# /sbin/route −A inet6 del <prefixtoroute3> dev sit3
# /sbin/ifconfig sit3 down
# /sbin/ip tunnel del sit3
```

8.3.2.2. Using "ifconfig" and "route" (deprecated because not very funny)

Not only the creation is strange, the shutdown also...you have to remove the tunnels in backorder, means the latest created must be removed first.

Usage (generic example for three tunnels):

```
# /sbin/route −A inet6 del <prefixtoroute3> dev sit3
# /sbin/ifconfig sit3 down
# /sbin/route −A inet6 del <prefixtoroute2> dev sit2
# /sbin/ifconfig sit2 down
# /sbin/route −A inet6 add <prefixtoroute1> dev sit1
# /sbin/ifconfig sit1 down
# /sbin/ifconfig sit0 down
```

8.3.2.3. Using "route"

This is like removing normal IPv6 routes

Usage (generic example for three tunnels):

```
# /sbin/route −A inet6 del <prefixtoroute1> gw
¬ ::<ipv4addressofforeigntunnel1> dev sit0
# /sbin/route −A inet6 del <prefixtoroute2> gw
¬ ::<ipv4addressofforeigntunnel2> dev sit0
# /sbin/route −A inet6 del <prefixtoroute3> gw
¬ ::<ipv4addressofforeigntunnel3> dev sit0
# /sbin/ifconfig sit0 down
```

8.3.3. Numbered point−to−point tunnels

Sometimes it's needed to configure a point−to−point tunnel with IPv6 addresses like in IPv4 today. This is only possible with the first (ifconfig+route − deprecated) and third (ip+route) tunnel setup. In such cases, you can add the IPv6 address to the tunnel interface like shown on interface configuration.

8.4. Setup of 6to4 tunnels

Pay attention that the support of 6to4 tunnels currently lacks on vanilla kernel series 2.2.x (see systemcheck/kernel for more information). Also note that that the prefix length for a 6to4 address is 16 because of from network point of view, all other 6to4 enabled hosts are on the same layer 2.

8.4.1. Add a 6to4 tunnel

First, you have to calculate your 6to4 prefix using your local assigned global routable IPv4 address (if your host has no global routable IPv4 address, in special cases NAT on border gateways is possible):

Assuming your IPv4 address is

```
1.2.3.4
```

the generated 6to4 prefix will be

```
2002:0102:0304::
```

Local 6to4 gateways should always assigned the manual suffix "::1", therefore your local 6to4 address will be

```
2002:0102:0304::1
```

Use e.g. following for automatic generation:

```
ipv4="1.2.3.4"; printf "2002:%02x%02x:%02x%02x::1" `echo $ipv4 | tr "." " "`
```

There are two ways possible to setup 6to4 tunneling now.

8.4.1.1. Using "ip" and a dedicated tunnel device

This is now the recommended way.

Create a new tunnel device

```
# /sbin/ip tunnel add tun6to4 mode sit remote any local <localipv4address>
```

Bring interface up

```
# /sbin/ip link set dev tun6to4 up
```

Add local 6to4 address to interface

```
# /sbin/ip -6 addr add <local6to4address>/16 dev tun6to4
```

Add (default) route to the global IPv6 network using the all−6to4−routers IPv4 anycast address

```
# /sbin/ip -6 route add 2000::/3 via ::192.88.99.1 dev tun6to4 metric 1
```

8.4.1.2. Using "ifconfig" and "route" and generic tunnel device "sit0" (deprecated)

This is now deprecated because using the generic tunnel device sit0 doesn't let specify filtering per device.

Bring generic tunnel interface sit0 up

```
# /sbin/ifconfig sit0 up
```

Add local 6to4 address to interface

```
# /sbin/ifconfig sit0 add <local6to4address>/16
```

Add (default) route to the global IPv6 network using the all−6to4−relays IPv4 anycast address

```
# /sbin/route -A inet6 add 2000::/3 gw ::192.88.99.1 dev sit0
```

8.4.2. Remove a 6to4 tunnel

8.4.2.1. Using "ip" and a dedicated tunnel device

Remove all routes through this dedicated tunnel device

```
# /sbin/ip -6 route flush dev tun6to4
```

Shut down interface

```
# /sbin/ip link set dev tun6to4 down
```

Remove created tunnel device

```
# /sbin/ip tunnel del tun6to4
```

8.4.2.2. Using "ifconfig" and "route" and generic tunnel device "sit0" (deprecated)

Remove (default) route through the 6to4 tunnel interface

```
# /sbin/route -A inet6 del 2000::/3 gw ::192.88.99.1 dev sit0
```

Remove local 6to4 address to interface

```
# /sbin/ifconfig sit0 del <local6to4address>/16
```

Shut down generic tunnel device (take care about this, perhaps it's still in use...)

```
# /sbin/ifconfig sit0 down
```

Chapter 9. Configuring IPv4−in−IPv6 tunnels

This will be filled in the future. At the moment, such tunnels are more used in test environments.

More information in the meantime: RFC 2473 / Generic Packet Tunneling in IPv6 Specification

Chapter 10. Kernel settings

To be filled...

10.1. /proc filesystem

To be filled with following content next: switches forwarding and autoconf behavior, acceptance of router advertisements and more.

10.1.1. Entries in /proc/net/

To be filled...

10.1.2. Entries in /proc/sys/net/

To be filled...

10.2. Netlink

To be filled...

Chapter 11. Network debugging

11.1. Server socket binding

11.1.1. Using "netstat" for server socket binding check

It's always interesting which server sockets are currently active on a node. Using "netstat" is a short way to get such information:

Used options: –nlptu

Example:

```
# netstat -nlptu
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
¬ PID/Program name
tcp        0      0 0.0.0.0:32768           0.0.0.0:*               LISTEN
¬ 1258/rpc.statd
tcp        0      0 0.0.0.0:32769           0.0.0.0:*               LISTEN
¬ 1502/rpc.mountd
tcp        0      0 0.0.0.0:515             0.0.0.0:*               LISTEN
¬ 22433/lpd Waiting
tcp        0      0 1.2.3.1:139             0.0.0.0:*               LISTEN
¬ 1746/smbd
tcp        0      0 0.0.0.0:111             0.0.0.0:*               LISTEN
¬ 1230/portmap
tcp        0      0 0.0.0.0:6000            0.0.0.0:*               LISTEN
¬ 3551/X
tcp        0      0 1.2.3.1:8081            0.0.0.0:*               LISTEN
¬ 18735/junkbuster
tcp        0      0 1.2.3.1:3128            0.0.0.0:*               LISTEN
¬ 18822/(squid)
tcp        0      0 127.0.0.1:953           0.0.0.0:*               LISTEN
¬ 30734/named
tcp        0      0 ::ffff:1.2.3.1:993      :::*                    LISTEN
¬ 6742/xinetd-ipv6
tcp        0      0 :::13                   :::*                    LISTEN
¬ 6742/xinetd-ipv6
tcp        0      0 ::ffff:1.2.3.1:143      :::*                    LISTEN
¬ 6742/xinetd-ipv6
tcp        0      0 :::53                   :::*                    LISTEN
¬ 30734/named
tcp        0      0 :::22                   :::*                    LISTEN
¬ 1410/sshd
tcp        0      0 :::6010                 :::*                    LISTEN
¬ 13237/sshd
udp        0      0 0.0.0.0:32768           0.0.0.0:*
¬ 1258/rpc.statd
udp        0      0 0.0.0.0:2049            0.0.0.0:*
¬ -
udp        0      0 0.0.0.0:32770           0.0.0.0:*
¬ 1502/rpc.mountd
udp        0      0 0.0.0.0:32771           0.0.0.0:*
¬ -
udp        0      0 1.2.3.1:137             0.0.0.0:*
¬ 1751/nmbd
```

```
udp        0      0 0.0.0.0:137             0.0.0.0:*
¬ 1751/nmbd
udp        0      0 1.2.3.1:138             0.0.0.0:*
¬ 1751/nmbd
udp        0      0 0.0.0.0:138             0.0.0.0:*
¬ 1751/nmbd
udp        0      0 0.0.0.0:33044           0.0.0.0:*
¬ 30734/named
udp        0      0 1.2.3.1:53              0.0.0.0:*
¬ 30734/named
udp        0      0 127.0.0.1:53            0.0.0.0:*
¬ 30734/named
udp        0      0 0.0.0.0:67              0.0.0.0:*
¬ 1530/dhcpd
udp        0      0 0.0.0.0:67              0.0.0.0:*
¬ 1530/dhcpd
udp        0      0 0.0.0.0:32858           0.0.0.0:*
¬ 18822/(squid)
udp        0      0 0.0.0.0:4827            0.0.0.0:*
¬ 18822/(squid)
udp        0      0 0.0.0.0:111             0.0.0.0:*
¬ 1230/portmap
udp        0      0 :::53                   :::*
¬ 30734/named
```

11.2. Examples for tcpdump packet dumps

Here some examples of captured packets are shown, perhaps useful for your own debugging...

...more coming next...

11.2.1. Router discovery

11.2.1.1. Router advertisement

```
15:43:49.484751 fe80::212:34ff:fe12:3450 > ff02::1: icmp6: router
¬ advertisement(chlim=64, router_ltime=30, reachable_time=0,
¬ retrans_time=0)(prefix info: AR valid_ltime=30, preffered_ltime=20,
¬ prefix=2002:0102:0304:1::/64)(prefix info: LAR valid_ltime=2592000,
¬ preffered_ltime=604800, prefix=3ffe:ffff:0:1::/64)(src lladdr:
¬ 0:12:34:12:34:50) (len 88, hlim 255)
```

Router with link−local address "fe80::212:34ff:fe12:3450" send an advertisement to the all−node−on−link multicast address "ff02::1" containing two prefixes "2002:0102:0304:1::/64" (lifetime 30 s) and "3ffe:ffff:0:1::/64" (lifetime 2592000 s) including its own layer 2 MAC address "0:12:34:12:34:50"

11.2.1.2. Router solicitation

```
15:44:21.152646 fe80::212:34ff:fe12:3456 > ff02::2: icmp6: router solicitation
¬ (src lladdr: 0:12:34:12:34:56) (len 16, hlim 255)
```

```
```

Node with link−local address "fe80::212:34ff:fe12:3456" and layer 2 MAC address "0:12:34:12:34:56" is looking for a router on−link, therefore sending this solicitation to the all−router−on−link multicast address "ff02::2".

11.2.2. Neighbor discovery

11.2.2.1. Neighbor discovery solicitation for duplicate address detection

Following packets are sent by a node with layer 2 MAC address "0:12:34:12:34:56" during autoconfiguration to check whether a potential address is already used by another node on the link sending this to the solicited−node link−local multicast address

* Node wants to configure its link−local address "fe80::212:34ff:fe12:3456", checks for duplicate now

```
15:44:17.712338 :: > ff02::1:ff12:3456: icmp6: neighbor sol: who has
¬ fe80::212:34ff:fe12:3456(src lladdr: 0:12:34:12:34:56) (len 32, hlim 255)
```

* Node wants to configure its global address "2002:0102:0304:1:212:34ff:fe12:3456" (after receiving advertisement shown above), checks for duplicate now

```
15:44:21.905596 :: > ff02::1:ff12:3456: icmp6: neighbor sol: who has
¬ 2002:0102:0304:1:212:34ff:fe12:3456(src lladdr: 0:12:34:12:34:56) (len 32,
¬ hlim 255)
```

* Node wants to configure its global address "3ffe:ffff:0:1:212:34ff:fe12:3456" (after receiving advertisement shown above), checks for duplicate now

```
15:44:22.304028 :: > ff02::1:ff12:3456: icmp6: neighbor sol: who has
¬ 3ffe:ffff:0:1:212:34ff:fe12:3456(src lladdr: 0:12:34:12:34:56) (len 32, hlim
¬ 255)
```

11.2.2.2. Neighbor discovery solicitation for looking for host or gateway

* Note wants to send packages to "3ffe:ffff:0:1::10" but has no layer 2 MAC address to send packet, so send solicitation now

```
13:07:47.664538 2002:0102:0304:1:2e0:18ff:fe90:9205 > ff02::1:ff00:10: icmp6:
¬ neighbor sol: who has 3ffe:ffff:0:1::10(src lladdr: 0:e0:18:90:92:5) (len 32,
¬ hlim 255)
```

* Node looks for "fe80::10" now

```
13:11:20.870070 fe80::2e0:18ff:fe90:9205 > ff02::1:ff00:10: icmp6: neighbor
¬ sol: who has fe80::10(src lladdr: 0:e0:18:90:92:5) (len 32, hlim 255)
```

Chapter 12. Support for persistent IPv6 configuration in Linux distributions

Some Linux distribution contain already support of a persistent IPv6 configuration using existing or new configuration and script files and some hook in the IPv4 script files.

12.1. Red Hat Linux and "clones"

Since starting writing the IPv6 & Linux − HowTo it was my intention to enable a persistent IPv6 configuration which catch most of the wished cases like host−only, router−only, dual−homed−host, router with second stub network, normal tunnels, 6to4 tunnels, and so on. Nowadays there exists a set of configuration and script files which do the job very well (never heard about real problems, but I don't know how many use the set. Because this configuration and scrips files are extended from time to time, they got their own HOWTO page: IPv6−HOWTO/scripts/current. Because I began my IPv6 experience using a Red Hat Linux 5.0 clone, my IPv6 development systems are mostly Red Hat Linux based now, it's kind a logic that the scripts are developed for this kind of distribution (so called *historic issue*). Also it was very easy to extend some configuration files, create new ones and create some simple hook for calling IPv6 setup during IPv4 setup.

Fortunately, in Red Hat Linux since 7.1 a snapshot of my IPv6 scripts is included, this was and is still further on assisted by Pekka Savola.

Mandrake since version 8.0 also includes an IPv6−enabled initscript package, but a minor bug still prevents usage ("ifconfig" misses "inet6" before "add").

12.1.1. Test for IPv6 support of network configuration scripts

You can test, whether your Linux distribution contain support for persistent IPv6 configuration using my set. Following script library should exist:

```
/etc/sysconfig/network-scripts/network-functions-ipv6
```

Auto−magically test:

```
# test -f /etc/sysconfig/network-scripts/network-functions-ipv6 && echo "Main
¬ IPv6 script libary exists"
```

The version of the library is important if you miss some features. You can get it executing following (or easier look at the top of the file):

```
# source /etc/sysconfig/network-scripts/network-functions-ipv6 &&
¬ getversion_ipv6_functions
20011124
```

In shown example, the used version is 20011124. Check this against latest information on IPv6−HOWTO/scripts/current to see what has been changed. There is also a change−log available in the distributed tar−ball.

12.1.2. Short hint for enabling IPv6 on current RHL 7.1, 7.2, ...

- Check whether running system has already IPv6 module loaded

```
# modprobe −c | grep net−pf−10
alias net−pf−10 off
```

- If result is "off", then enable IPv6 networking by editing /etc/sysconfig/network, add following new line

```
NETWORKING_IPV6=yes
```

- Reboot or restart networking using

```
# service network restart
```

- Now IPv6 module should be loaded

```
# modprobe −c | grep ipv6
alias net−pf−10 ipv6
```

If your system is on a link which provides router advertisement, autoconfiguration will be done automatically. For more information which settings are supported see /usr/share/doc/initscripts−$version/sysconfig.txt.

12.2. SuSE Linux

In newer versions there is a really rudimentary support available, see /etc/rc.config for details.

Because of the really different configuration and script file structure it is hard (or impossible) to use the set for Red Hat Linux and clones with this distribution.

12.2.1. Further information

- How to setup 6to4 IPv6 with SuSE 7.3

12.3. Debian Linux

I still don't have any information weather a persistent IPv6 configuration can stored somewhere.

12.3.1. Further information

* IPv6 on Debian Linux

Chapter 13. Auto−configuration and mobility

13.1. Stateless auto−configuration

Is supported and seen on the assigned link−local address after an IPv6−enabled interface is up.

13.2. Stateful auto−configuration using Router Advertisement Daemon (radvd)

to be filled. See radvd daemon autoconfiguration below.

13.3. Dynamic Host Configuration Protocol v6 (DHCPv6)

to be filled.

13.4. Mobility

to be filled.

For the moment, see Mobile IPv6 for Linux(MIPL) homepage for more details

Chapter 14. Firewalling and security issues

IPv6 firewalling is important, especially if using IPv6 on internal networks with global IPv6 addresses. Because unlike at IPv4 networks where in common internal hosts are protected automatically using private IPv4 addresses like RFC 1918 / Address Allocation for Private Internets or APIPA / Automatic Private IP Addressing, in IPv6 normally global addresses are used and someone with IPv6 connectivity can reach all internal IPv6 enabled nodes.

14.1. Firewalling

14.1.1. Firewalling using netfilter6

Native IPv6 firewalling is only supported in kernel versions 2.4+. In older 2.2− you can only filter IPv6−in−IPv4 by protocol 41.

Attention: no warranty that described rules or examples are really protect your system!

14.1.1.1. More information

- Netfilter project
- maillist archive of netfilter users
- maillist archive of netfilter developers
- Unofficial status informations

14.1.2. Preparation

14.1.2.1. Get sources

Get the latest kernel source: http://www.kernel.org/

Get the latest iptables package:

- Source tarball (for kernel patches): http://www.netfilter.org/
- Source RPM for rebuild of binary (for RedHat systems): ftp://ftp.redhat.com/redhat/linux/rawhide/SRPMS/SRPMS/ or perhaps also at http://www.netcore.fi/pekkas/linux/ipv6/

14.1.2.2. Extract sources

Change to source directory:

```
# cd /path/to/src
```

Unpack and rename kernel sources

```
# tar z|jxf kernel-version.tar.gz|bz2
# mv linux linux-version-iptables-version+IPv6
```

Unpack iptables sources

```
# tar z|jxf iptables-version.tar.gz|bz2
```

14.1.2.3. Apply latest iptables/IPv6-related patches to kernel source

Change to iptables directory

```
# cd iptables-version
```

Apply pending patches

```
# make pending-patches KERNEL_DIR=/path/to/src/linux-version-iptables-version/
```

Apply additional IPv6 related patches (still not in the vanilla kernel included)

```
# make patch-o-matic KERNEL_DIR=/path/to/src/linux-version-iptables-version/
```

Say yes at following options (iptables-1.2.2)

- ah-esp.patch
- masq-dynaddr.patch (only needed for systems with dynamic IP assigned WAN connections like PPP or PPPoE)
- ipv6-agr.patch.ipv6
- ipv6-ports.patch.ipv6
- LOG.patch.ipv6
- REJECT.patch.ipv6

Check IPv6 extensions

```
# make print-extensions
Extensions found: IPv6:owner IPv6:limit IPv6:mac IPv6:multiport
```

14.1.2.4. Configure, build and install new kernel

Change to kernel sources

```
# cd /path/to/src/linux-version-iptables-version/
```

Edit Makefile

```
- EXTRAVERSION =
+ EXTRAVERSION = -iptables-version+IPv6-try
```

Run configure, enable IPv6 related

```
            Code maturity level options
                  Prompt for development and/or incomplete code/drivers : yes
            Networking options
                  Network packet filtering: yes
                  The IPv6 protocol: module
                       IPv6: Netfilter Configuration
                             IP6 tables support: module
                             All new options like following:
                                   limit match support: module
                                   MAC address match support: module
                                   Multiple port match support: module
                                   Owner match support: module
                                   netfilter MARK match support: module
                                   Aggregated address check: module
                                   Packet filtering: module
                                        REJECT target support: module
                                        LOG target support: module
                                   Packet mangling: module
                                   MARK target support: module
```

Configure other related to your system, too

Compilation and installing: see the kernel section here and other HOWTOs

14.1.2.5. Rebuild and install binaries of iptables

Make sure, that upper kernel source tree is also available at /usr/src/linux/

Rename older directory

```
# mv /usr/src/linux /usr/src/linux.old
```

Create a new softlink

```
# ln /path/to/src/linux-version-iptables-version /usr/src/linux
```

Rebuild SRPMS

```
# rpm --rebuild /path/to/SRPMS/iptables-version-release.src.rpm
```

Install new iptables packages (iptables + iptables−ipv6)

* On RH 7.1 systems, normally, already an older version is installed, therefore use "freshen"

```
# rpm -Fhv /path/to/RPMS/cpu/iptables*-version-release.cpu.rpm
```

* If not already installed, use "install"

```
# rpm -ihv /path/to/RPMS/cpu/iptables*-version-release.cpu.rpm
```

* On RH 6.2 systems, normally, no kernel 2.4.x is installed, therefore the requirements don't fit. Use "−−nodeps" to install it

```
# rpm -ihv --nodep /path/to/RPMS/cpu/iptables*-version-release.cpu.rpm
```

Perhaps it's necessary to create a softlink for iptables libraries where iptables looks for them

```
# ln -s /lib/iptables/ /usr/lib/iptables
```

14.1.3. Usage

14.1.3.1. Check for support

Load module, if so compiled

```
# modprobe ip6_tables
```

Check for capability

```
# [ ! -f /proc/net/ip6_tables_names ] && echo "Current kernel doesn't support
¬ 'ip6tables' firewalling (IPv6)!"
```

14.1.3.2. Learn how to use ip6tables

List all IPv6 netfilter entries

* Short

```
# ip6tables -L
```

* Extended

```
# ip6tables -n -v --line-numbers -L
```

List specified filter

```
# ip6tables -n -v --line-numbers -L INPUT
```

Insert a log rule at the input filter with options

```
# ip6tables --table filter --append INPUT  -j LOG --log-prefix "INPUT:"
¬ --log-level 7
```

Insert a drop rule at the input filter

```
# ip6tables --table filter --append INPUT  -j DROP
```

Delete a rule by number

```
# ip6tables --table filter --delete INPUT 1
```

Allow ICMPv6, at the moment, with unpatched kernel 2.4.5 and iptables−1.2.2 no type can be specified

- Accept incoming ICMPv6 through tunnels

```
# ip6tables -A INPUT -i sit+ -p icmpv6 -j ACCEPT
```

- Allow outgoing ICMPv6 through tunnels

```
# ip6tables -A OUTPUT -o sit+ -p icmpv6 -j ACCEPT
```

Allow incoming SSH, here an example is shown for a ruleset which allows incoming SSH connection from a specified IPv6 address

- Allow incoming SSH from 3ffe:400:100::1/128

```
# ip6tables -A INPUT -i sit+ -p tcp -s 3ffe:400:100::1/128 --sport 512:65535
¬ --dport 22 -j ACCEPT
```

- Allow response packets (at the moment IPv6 connection tracking isn't in mainstream netfilter6 implemented)

```
# ip6tables -A OUTPUT -o sit+ -p tcp -d 3ffe:400:100::1/128 --dport 512:65535
¬ --sport 22 ! --syn j ACCEPT
```

Enable tunneled IPv6−in−IPv4, to accept tunneled IPv6−in−IPv4 packets, you have to insert rules in your IPv4 firewall setup relating to such packets, for example

- Accept incoming IPv6−in−IPv4 on interface ppp0

```
# iptables -A INPUT -i ppp0 -p ipv6 -j ACCEPT
```

- Allow outgoing IPv6−in−IPv4 to interface ppp0

```
# iptables -A OUTPUT -o ppp0 -p ipv6 -j ACCEPT
```

If you have only a static tunnel, you can specify the IPv4 addresses, too, like

- Accept incoming IPv6−in−IPv4 on interface ppp0 from tunnel endpoint 1.2.3.4

```
# iptables -A INPUT -i ppp0 -p ipv6 -s 1.2.3.4 -j ACCEPT
```

* Allow outgoing IPv6−in−IPv4 to interface ppp0 to tunnel endpoint 1.2.3.4

```
# iptables −A OUTPUT −o ppp0 −p ipv6 −d 1.2.3.4 −j ACCEPT
```

Protect against incoming TCP connection requests (VERY RECOMMENDED!), for security issues you should really insert a rule which blocks incoming TCP connection requests. Adapt "−i" option, if other interface names are in use!

* Block incoming TCP connection requests to this host

```
# ip6tables −I INPUT −i sit+ −p tcp --syn −j DROP
```

* Block incoming TCP connection requests to hosts behind this router

```
# ip6tables −I FORWARD −i sit+ −p tcp --syn −j DROP
```

Perhaps the rules have to be placed below others, but that is work you have to think about it. Best way is to create a script and execute rules in a specified way.

Protect against incoming UDP connection requests (ALSO RECOMMENDED!), like mentioned on my firewall information it's possible to control the ports on outgoing UDP/TCP sessions. So if all of your local IPv6 systems are use local ports e.g. from 32768 to 60999 you are able to filter UDP connections also (until connection tracking works) like:

* Block incoming UDP packets which cannot be responses of outgoing requests of this host

```
# ip6tables −I INPUT −i sit+ −p udp ! --dport 32768:60999 −j DROP
```

* Block incoming UDP packets which cannot be responses of forwarded requests of hosts behind this router

```
ip6tables −I FORWARD −i sit+ −p udp ! --dport 32768:60999 −j DROP
```

14.1.3.3. Demonstration example

Following lines show a more sophisticated setup as an example. Happy netfilter6 ruleset creation....

```
# ip6tables −n −v −L
Chain INPUT (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 extIN      all      sit+   *       ::/0                 ::/0
    4   384 intIN      all      eth0   *       ::/0                 ::/0
    0     0 ACCEPT     all      *      *       ::1/128              ::1/128
    0     0 ACCEPT     all      lo     *       ::/0                 ::/0
    0     0 LOG        all      *      *       ::/0                 ::/0
¬        LOG flags 0 level 7 prefix `INPUT−default:'
    0     0 DROP       all      *      *       ::/0                 ::/0

Chain FORWARD (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
¬
    0     0 int2ext    all      eth0   sit+    ::/0                 ::/0
    0     0 ext2int    all      sit+   eth0    ::/0                 ::/0
    0     0 LOG        all      *      *       ::/0                 ::/0
```

```
¬         LOG flags 0 level 7 prefix `FORWARD-default:'
    0     0 DROP       all      *      *       ::/0                 ::/0

Chain OUTPUT (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
¬
    0     0 extOUT     all      *      sit+    ::/0                 ::/0
    4   384 intOUT     all      *      eth0    ::/0                 ::/0
    0     0 ACCEPT     all      *      *       ::1/128              ::1/128
    0     0 ACCEPT     all      *      lo      ::/0                 ::/0
    0     0 LOG        all      *      *       ::/0                 ::/0
¬         LOG flags 0 level 7 prefix `OUTPUT-default:'
    0     0 DROP       all      *      *       ::/0                 ::/0

Chain ext2int (1 references)
 pkts bytes target     prot opt in     out     source               destination
¬
    0     0 ACCEPT     icmpv6    *      *       ::/0                 ::/0
    0     0 ACCEPT     tcp      *      *       ::/0                 ::/0
¬         tcp spts:1:65535 dpts:1024:65535 flags:!0x16/0x02
    0     0 LOG        all      *      *       ::/0                 ::/0
¬         LOG flags 0 level 7 prefix `ext2int-default:'
    0     0 DROP       tcp      *      *       ::/0                 ::/0
    0     0 DROP       udp      *      *       ::/0                 ::/0
    0     0 DROP       all      *      *       ::/0                 ::/0

Chain extIN (1 references)
 pkts bytes target     prot opt in     out     source               destination
¬
    0     0 ACCEPT     tcp      *      *       3ffe:400:100::1/128  ::/0
¬         tcp spts:512:65535 dpt:22
    0     0 ACCEPT     tcp      *      *       3ffe:400:100::2/128  ::/0
¬         tcp spts:512:65535 dpt:22
    0     0 ACCEPT     icmpv6    *      *       ::/0                 ::/0
    0     0 ACCEPT     tcp      *      *       ::/0                 ::/0
¬         tcp spts:1:65535 dpts:1024:65535 flags:!0x16/0x02
    0     0 ACCEPT     udp      *      *       ::/0                 ::/0
¬         udp spts:1:65535 dpts:1024:65535
    0     0 LOG        all      *      *       ::/0                 ::/0
¬         limit: avg 5/min burst 5 LOG flags 0 level 7 prefix `extIN-default:'
    0     0 DROP       all      *      *       ::/0                 ::/0

Chain extOUT (1 references)
 pkts bytes target     prot opt in     out     source               destination
¬
    0     0 ACCEPT     tcp      *      *       ::/0
¬ 3ffe:400:100::1/128tcp spt:22 dpts:512:65535 flags:!0x16/0x02
    0     0 ACCEPT     tcp      *      *       ::/0
¬ 3ffe:400:100::2/128tcp spt:22 dpts:512:65535 flags:!0x16/0x02
    0     0 ACCEPT     icmpv6    *      *       ::/0                 ::/0
    0     0 ACCEPT     tcp      *      *       ::/0                 ::/0
¬         tcp spts:1024:65535 dpts:1:65535
    0     0 ACCEPT     udp      *      *       ::/0                 ::/0
¬         udp spts:1024:65535 dpts:1:65535
    0     0 LOG        all      *      *       ::/0                 ::/0
¬         LOG flags 0 level 7 prefix `extOUT-default:'
    0     0 DROP       all      *      *       ::/0                 ::/0

Chain int2ext (1 references)
 pkts bytes target     prot opt in     out     source               destination
¬
    0     0 ACCEPT     icmpv6    *      *       ::/0                 ::/0
```

```
    0     0 ACCEPT     tcp      *      *       ::/0                 ::/0
¬        tcp spts:1024:65535 dpts:1:65535
    0     0 LOG        all      *      *       ::/0                 ::/0
¬        LOG flags 0 level 7 prefix `int2ext:'
    0     0 DROP       all      *      *       ::/0                 ::/0
    0     0 LOG        all      *      *       ::/0                 ::/0
¬        LOG flags 0 level 7 prefix `int2ext-default:'
    0     0 DROP       tcp      *      *       ::/0                 ::/0
    0     0 DROP       udp      *      *       ::/0                 ::/0
    0     0 DROP       all      *      *       ::/0                 ::/0

Chain intIN (1 references)
 pkts bytes target     prot opt in     out     source               destination
¬
    0     0 ACCEPT     all      *      *       ::/0
¬ fe80::/ffc0::
    4   384 ACCEPT     all      *      *       ::/0                 ff02::/16

Chain intOUT (1 references)
 pkts bytes target     prot opt in     out     source               destination
¬
    0     0 ACCEPT     all      *      *       ::/0
¬ fe80::/ffc0::
    4   384 ACCEPT     all      *      *       ::/0                 ff02::/16
    0     0 LOG        all      *      *       ::/0                 ::/0
¬        LOG flags 0 level 7 prefix `intOUT-default:'
    0     0 DROP       all      *      *       ::/0                 ::/0
```

14.2. Security

14.2.1. Node security

It's very recommend to apply all available patches and disable all not necessary services.

More to be filled...

14.2.2. Access limitations

Many services uses the tcp_wrapper library for access control. Below is described the use of tcp_wrapper.

More to be filled...

Chapter 15. Encryption and Authentication

Unlike in IPv4 encryption and authentication is a mandatory feature of IPv6. This features are normally implemented using IPsec (which can be also used by IPv4).

But because of the independence of encryption and authentication from the key exchange protocol there exists currently some interoperability problems regarding this issue.

15.1. Support in kernel

15.1.1. Support in vanilla Linux kernel

Currently missing in 2.4, perhaps in 2.5 (see below). There is an issue about keeping the Linux kernel source free of export/import−control−laws regarding encryption code. This is also one case why FreeS/WAN project (IPv4 only IPsec) isn't still contained in vanilla source.

15.1.2. Support in USAGI kernel

The USAGI project has taken over in July 2001 the IPv6 enabled FreeS/WAN code from the IABG / IPv6 Project and included in their kernel extensions, but still work in progress, means that not all IABG features are already working in USAGI extension.

15.2. Usage

to be filled, mostly like FreeS/WAN for IPv4. For the meantime look for documentation at FreeS/WAN / Online documentation.

Chapter 16. Quality of Service (QoS)

IPv6 supports QoS with use of Flow Labels and Traffic Classes. This can be controlled using "tc" (contained in package "iproute").

more to be filled...

Chapter 17. Hints for IPv6−enabled daemons

Here some hints are shown for IPv6−enabled daemons.

17.1. Berkeley Internet Name Daemon BIND (named)

IPv6 is supported since version 9. Always use newest available version. At least version 9.1.3 must be used, older versions can contain remote exploitable security holes.

17.1.1. Listening on IPv6 addresses

Note: unlike in IPv4 current versions doesn't allow to bind a server socket to dedicated IPv6 addresses, so only *any* or *none* are valid. Because this can be a security issue, check the Access Control List (ACL) section below, too!

17.1.1.1. Enable BIND named for listening on IPv6 address

To enable IPv6 for listening, following options are requested to change

```
options {
        # sure other options here, too
        listen-on-v6 { any; };
};
```

This should result after restart in e.g.

```
# netstat -lnptu |grep "named\W*$"
tcp 0 0 :::53         :::*      LISTEN 1234/named # incoming TCP requests
udp 0 0 1.2.3.4:53    0.0.0.0:*        1234/named # incoming UDP requests to IPv4 1.2.3.4
udp 0 0 127.0.0.1:53  0.0.0.0:*        1234/named # incoming UDP requests to IPv4 localhost
udp 0 0 0.0.0.0:32868 0.0.0.0:*        1234/named # dynamic chosen port for outgoing queries
udp 0 0 :::53         :::*             1234/named # incoming UDP request to any IPv6
```

And a simple test looks like

```
# dig localhost @::1
```

and should show you a result.

17.1.1.2. Disable BIND named for listening on IPv6 address

To disable IPv6 for listening, following options are requested to change

```
options {
```

```
        # sure other options here, too
        listen-on-v6 { none; };
};
```

17.1.2. IPv6 enabled Access Control Lists (ACL)

IPv6 enabled ACLs are possible and should be used whenever it's possible. An example looks like following:

```
acl internal-net {
        127.0.0.1;
        1.2.3.0/24;
        3ffe:ffff:100::/56;
        ::1/128;
        ::ffff:1.2.3.4/128;
};
acl ns-internal-net {
        1.2.3.4;
        1.2.3.5;
        3ffe:ffff:100::4/128;
        3ffe:ffff:100::5/128;
};
```

This ACLs can be used e.g. for queries of clients and transfer zones to secondary name−servers. This prevents also your caching name−server to be used from outside using IPv6.

```
options {
        # sure other options here, too
        listen-on-v6 { none; };
        allow-query { internal-net; };
        allow-transfer { ns-internal-net; };
};
```

It's also possible to set the *allow−query* and *allow−transfer* option for most of single zone definitions, too.

17.1.3. Sending queries with dedicated IPv6 address

This option is not required, but perhaps needed:

```
query-source-v6 address <ipv6address|*> port <port|*>;
```

17.1.4. Per zone defined dedicated IPv6 addresses

It's also possible to define per zone some IPv6 addresses.

17.1.4.1. Transfer source address

Transfer source address is used for outgoing zone transfers:

```
transfer-source-v6 <ipv6addr|*> [port port];
```

17.1.4.2. Notify source address

Notify source address is used for outgoing notify messages:

```
notify-source-v6 <ipv6addr|*> [port port];
```

17.1.5. Serving IPv6 related DNS data

For IPv6 new types and root zones for reverse lookups are defined:

- AAAA and reverse IP6.INT: specified in RFC 1886 / DNS Extensions to support IP version 6, usable since BIND version 4.9.6
- A6, DNAME and reverse IP6.ARPA: specified in RFC 2874 / DNS Extensions to Support IPv6 Address Aggregation and Renumbering, usable since BIND 9, but see also an information about the current state at draft−ietf−dnsext−ipv6−addresses−00.txt

Perhaps filled later more content, for the meantime take a look at given RFCs and

- AAAA and reverse IP6.INT: IPv6 DNS Setup Information
- A6, DNAME and reverse IP6.ARPA: take a look into chapter 4 and 6 of the BIND 9 Administrator Reference Manual (ARM) distributed which the bind−package or get this here: BIND version 9 ARM (PDF)

Because IP6.INT is deprecated (but still in use), a DNS server which will support IPv6 information has to serve both reverse zones.

17.1.5.1. Current best practice

Because there are some troubles around using the new formats, current best practice is:

Forward lookup support:

- AAAA
- A6 without chaining, means prefix length value set to 0

Reverse lookup support:

- Reverse nibble format for zone ip6.int
- Reverse nibble format for zone ip6.arpa

17.1.6. Checking IPv6−enabled connect

To check, whether BIND is listening on an IPv6 socket and serving data see following examples.

17.1.6.1. IPv6 connect, but denied by ACL

Specifying a dedicated server for the query, an IPv6 connect can be forced:

```
$ host -t aaaa www.6bone.net 3ffe:ffff:200:f101::1
Using domain server:
Name: 3ffe:ffff:200:f101::1
Address: 3ffe:ffff:200:f101::1#53
Aliases:
Host www.6bone.net. not found: 5(REFUSED)
```

Related log entry looks like following:

```
Jan 3 12:43:32 gate named[12347]: client
¬ 3ffe:ffff:200:f101:212:34ff:fe12:3456#32770:
 query denied
```

If you see such entries in the log, check whether requests from this client should be allowed and perhaps review your ACL configuration.

17.1.6.2. Successful IPv6 connect

A successful IPv6 connect looks like following:

```
$ host -t aaaa www.6bone.net 3ffe:ffff:200:f101::1
Using domain server:
Name: 3ffe:ffff:200:f101::1
Address: 3ffe:ffff:200:f101::1#53
Aliases:
www.6bone.net. is an alias for 6bone.net.
6bone.net. has AAAA address 3ffe:b00:c18:1::10
```

17.2. Internet super daemon (xinetd)

IPv6 is supported since version around 1.8.9. Always use newest available version. At least version 2.3.3 must be used, older versions can contain remote exploitable security holes.

Some Linux distribution contain an extra package for the IPv6 enabled xinetd, some others start the IPv6−enabled xinetd if following variable is set: NETWORKING_IPV6="yes", mostly done by /etc/sysconfig/network (only valid for Red Hat like distributions).

If you enable a built−in service like e.g. daytime by modifying the configuration file in /etc/xinetd.d/daytime like

```
# diff −u /etc/xinetd.d/daytime.orig /etc/xinetd.d/daytime
--- /etc/xinetd.d/daytime.orig Sun Dec 16 19:00:14 2001
+++ /etc/xinetd.d/daytime Sun Dec 16 19:00:22 2001
@@ −10,5 +10,5 @@
        protocol = tcp
        user = root
        wait = no
−       disable = yes
+       disable = no
 }
```

After restarting the xinetd you should get a positive result like:

```
# netstat −lnptu −A inet6 |grep "xinetd*"
tcp 0 0 ::ffff:192.168.1.1:993  :::*  LISTEN  12345/xinetd−ipv6
tcp 0 0 :::13                   :::*  LISTEN  12345/xinetd−ipv6 <− service
¬ daytime/tcp
tcp 0 0 ::ffff:192.168.1.1:143  :::*  LISTEN  12345/xinetd−ipv6
```

Shown example also displays an IMAP and IMAP−SSL IPv4−only listening xinetd.

Note: An IPv4−only xinetd won't start on an IPv6−enabled node and also the IPv6−enabled won't start on an IPv4−only node (will be hopefully fixed in the future).

17.3. Webserver Apache2 (httpd2)

Apache web server supports IPv6 native by maintainers since 2.0.14. Available patches for the older 1.3.x series are not current and shouldn't be used in public environment, but available at KAME / Misc.

17.3.1. Listening on IPv6 addresses

Note: virtual hosts on IPv6 addresses are broken in versions until 2.0.28 (a patch is available for 2.0.28).

17.3.1.1. Virtual host listen on an IPv6 address only

```
Listen [3ffe:ffff:100::1]:80
<VirtualHost [3ffe:ffff:100::1]:80>
        ServerName ipv6only.yourdomain.yourtopleveldomain
        # ...sure more config lines
</VirtualHost>
```

17.3.1.2. Virtual host listen on an IPv6 and on an IPv4 address

```
Listen [3ffe:ffff:100::2]:80
Listen 1.2.3.4:80
<VirtualHost [3ffe:ffff:100::2]:80 1.2.3.4:80>
        ServerName ipv6andipv4.yourdomain.yourtopleveldomain
        # ...sure more config lines
</VirtualHost>
```

This should result after restart in e.g.

```
# netstat -lnptu |grep "httpd2\W*$"
tcp 0 0 1.2.3.4:80          0.0.0.0:* LISTEN 12345/httpd2
tcp 0 0 3ffe:ffff:100::1:80 :::*      LISTEN 12345/httpd2
tcp 0 0 3ffe:ffff:100::2:80 :::*      LISTEN 12345/httpd2
```

For simple tests use the telnet example already shown.

17.4. Router Advertisement Daemon (radvd)

The router advertisement daemon is very useful on a LAN, if clients should be auto−configured. The daemon itself should run a Linux router (not necessary the default IPv4 gateway).

You can specify some information and flags which should be contained in the advertisement. Common used are

- Prefix (needed)
- Lifetime of the prefix
- Frequency of sending advertisements (optional)

After a proper configuration, the daemon sends advertisements through specified interfaces and clients are hopefully receive them and auto−magically configure addresses with received prefix and the default route.

17.4.1. Configuring radvd

17.4.1.1. Simple configuration

Radvd's config file is normally /etc/radvd.conf. An simple example looks like following:

```
interface eth0 {
        AdvSendAdvert on;
        MinRtrAdvInterval 3;
        MaxRtrAdvInterval 10;
        prefix 3ffe:ffff:0100:f101::/64 {
                AdvOnLink on;
                AdvAutonomous on;
                AdvRouterAddr on;
        };
};
```

This results on client side in

```
# ip -6 addr show eth0
3: eth0: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast qlen 100
    inet6 3ffe:ffff:100:f101:2e0:12ff:fe34:1234/64 scope global dynamic
       valid_lft 2591992sec preferred_lft 604792sec
    inet6 fe80::2e0:12ff:fe34:1234/10 scope link
```

Because no lifetime was defined, a very high value was used.

17.4.1.2. Special 6to4 configuration

Version since 0.6.2pl3 support the automatic (re)−generation of the prefix depending on an IPv4 address of a specified interface. This can be used to distribute advertisements in a LAN after the 6to4 tunneling has changed. Mostly used behind a dynamic dial−on−demand Linux router. Because of the sure shorter lifetime of such prefix (after each dial−up, another prefix is valid), the lifetime configured to minimal values:

```
interface eth0 {
        AdvSendAdvert on;
        MinRtrAdvInterval 3;
        MaxRtrAdvInterval 10;
        prefix 0:0:0:f101::/64 {
                AdvOnLink off;
                AdvAutonomous on;
                AdvRouterAddr on;
                Base6to4Interface ppp0;
                AdvPreferredLifetime 20;
                AdvValidLifetime 30;
        };
};
```

This results on client side in (assuming, ppp0 has currently 1.2.3.4 as local IPv4 address):

```
# ip -6 addr show eth0
3: eth0: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast qlen 100
   inet6 2002:0102:0304:f101:2e0:12ff:fe34:1234/64 scope global dynamic
      valid_lft 22sec preferred_lft 12sec
   inet6 fe80::2e0:12ff:fe34:1234/10 scope link
```

Because a small lifetime was defined, such prefix will be thrown away quickly, if no related advertisement was received.

17.4.2. Debugging

A program called "radvdump" can help you looking into sent or received advertisements. Simple to use:

```
# radvdump
Router advertisement from fe80::280:c8ff:feb9:cef9 (hoplimit 255)
```

```
        AdvCurHopLimit: 64
        AdvManagedFlag: off
        AdvOtherConfigFlag: off
        AdvHomeAgentFlag: off
        AdvReachableTime: 0
        AdvRetransTimer: 0
        Prefix 2002:0102:0304:f101::/64
                AdvValidLifetime: 30
                AdvPreferredLifetime: 20
                AdvOnLink: off
                AdvAutonomous: on
                AdvRouterAddr: on
        Prefix 3ffe:ffff:100:f101::/64
                AdvValidLifetime: 2592000
                AdvPreferredLifetime: 604800
                AdvOnLink: on
                AdvAutonomous: on
                AdvRouterAddr: on
        AdvSourceLLAddress: 00 80 12 34 56 78
```

Output shows you each advertisement package in readable format. You should see your configured values here again, if not, perhaps it's not your radvd which sends the advertisement...look for another router on the link (and take the LLAddress, which is the MAC address for tracing).

17.5. tcp_wrapper

tcp_wrapper is a library which can help you to protect service against misuse.

17.5.1. Filtering capabilities

You can use tcp_wrapper for

- Filtering against source addresses (IPv4 or IPv6)
- Filtering against users (requires a running ident daemon on the client)

17.5.2. Which program uses tcp_wrapper

Following are known:

- Each service which is called by xinetd (if xinetd is compiled using tcp_wrapper library)
- sshd (if compiled using tcp_wrapper)

17.5.3. Usage

tcp_wrapper is controlled by two files name /etc/hosts.allow and /etc/hosts.deny. For more information see

```
$ man hosts.all
```

17.5.3.1. Example for /etc/hosts.allow

In this file, each service which should be positive filtered (means connects are accepted) need a line.

```
sshd:           1.2.3. [3ffe:ffff:100:200::]/64
daytime-stream: 1.2.3. [3ffe:ffff:100:200::]/64
```

17.5.3.2. Example for /etc/hosts.deny

This file contains all negative filter entries and should normally deny the rest using

```
ALL: ALL
```

If this node is a more sensible one you can replace the standard line above with this one, but this can cause a DoS attack (load of mailer and spool directory), if too many connects were made in short time. Perhaps a logwatch is better for such issues.

```
ALL: ALL: spawn (echo "Attempt from %h %a to %d at `date`"
 | tee -a /var/log/tcp.deny.log | mail root@localhost)
```

17.5.4. Logging

Depending on the entry in the syslog daemon configuration file /etc/syslog.conf the tcp_wrapper logs normally into /var/log/secure.

17.5.4.1. Refused connection

A refused connection via IPv4 to an xinetd covered daytime service produces a line like following example

```
Jan 2 20:40:44 gate xinetd-ipv6[12346]: FAIL: daytime-stream libwrap
¬ from=::ffff:1.2.3.4
Jan 2 20:32:06 gate xinetd-ipv6[12346]: FAIL: daytime-stream libwrap
 from=3ffe:ffff:100:200::212:34ff:fe12:3456
```

A refused connection via IPv4 to an dual−listen sshd produces a line like following example

```
Jan 2 20:24:17 gate sshd[12345]: refused connect from ::ffff:1.2.3.4
¬ (::ffff:1.2.3.4)
Jan 2 20:39:33 gate sshd[12345]: refused connect
 from 3ffe:ffff:100:200::212:34ff:fe12:3456
¬ (3ffe:ffff:100:200::212:34ff:fe12:3456)
```

17.5.4.2. Permitted connection

A permitted connection via IPv4 to an xinetd covered daytime service produces a line like following example

```
Jan 2 20:37:50 gate xinetd-ipv6[12346]: START: daytime-stream pid=0
¬ from=::ffff:1.2.3.4
Jan 2 20:37:56 gate xinetd-ipv6[12346]: START: daytime-stream pid=0
 from=3ffe:ffff:100:200::212:34ff:fe12:3456
```

A permitted connection via IPv4 to an dual−listen sshd produces a line like following example

```
Jan 2 20:43:10 gate sshd[21975]: Accepted password for user from ::ffff:1.2.3.4
¬ port 33381 ssh2
Jan 2 20:42:19 gate sshd[12345]: Accepted password for user
 from 3ffe:ffff:100:200::212:34ff:fe12:3456 port 33380 ssh2
```

Chapter 18. Programming (using API)

I have no experience in IPv6 programming, perhaps this chapter will be filled by others or moved away to another HOWTO.

Chapter 19. Interoperability

There are some projects around the world which checks the interoperability of different operating systems regarding the implementation of IPv6 features. Here some URLs:

- TAHI Project

More coming next...

Chapter 20. Further information and URLs

20.1. Paper printed books, articles, online reviews (mixed)

20.1.1. German language

- Technik der IP−Netze (TCP/IP incl. IPv6) bei Amazon.de Anatol Badach, Erwin Hoffmann Carl Hanser Verlag München, Wien, 2001 ISBN: 3−446−21501−8 Kap. 6: Protokoll IPv6 S.205−242 Kap. 7: Plug&Play−Unterstützung bei IPv6 S.243−276 Kap. 8: Migration zum IPv6−Einsatz S.277−294 Kap. 9.3.4: RIP für das Protokoll IPv6 (RIPng) S.349−351 Kap. 9.4.6: OSPF für IPv6 S.384−385 Kommentar: tw. nicht ganz up−to−date bzw. nicht ganz fehlerfreie Abbildungen Homepage des Buches und Tabelle mit Fixes
- Internet−Sicherheit (Browser, Firewalls und Verschlüsselung) bei Amazon.de Kai Fuhrberg 2. akt. Auflage 2000 Carl Hanser Verlag München, Wien, ISBN: 3−446−21333−3 Kap.2.3.1.4. IPv6 S.18−22 Kurz angerissen werden: RFC1825 − Security Association Konzept RFC1826 − IP authentication Header RFC1827 − IP Encapsulation Security Payload

---

20.1.2. Articles, Books, Online Reviews (mixed)

- Getting Connected with 6to4 by Huber Feyrer, 06/01/2001
- How Long the Aversion to IP Version 6 − Review of META Group, Inc., full access needs (free) registration at META Group, Inc.
- O'reilly Network search for keyword IPv6 results in 29 hits (28. January 2002)
- Wireless boosting IPv6 by Carolyn Duffy Marsan, 10/23/2000
- IPv6, théorie et pratique (french) 2e édition, mars 1999, O'Reilly (??? no newer one available ???) ISBN: 2−84177−085−0
- Internetworking IPv6 with Cisco Routers by Silvano Gai, McGrawHill Italia, 1997 13 chapters and appendix A−D are downloadable as PDF−documents.
- Secure and Dynamic Tunnel Broker by Vegar Skaerven Wang, Master of Engineering Thesis in Computer Science, 2.June 2000, Faculty of Science, Dep.of Computer Science, University of Tromso, Norway.
- Aufbruch in die neue Welt − IPv6 in IPv4 Netzen von Dipl.Ing. Ralf Döring, TU Illmenau, 1999
- Migration and Co−existence of IPv4 and IPv6 in Residential Networks by Pekka Savola, CSC/FUNET, 2002
- Book IPv6 Essentials written by Silvia Hagen, release planned for April 2002

20.1.3. Others

See following URL for more: SWITCH IPv6 Pilot / References

20.2. Online information

20.2.1. Join the IPv6 backbone

More to be filled later...suggestions are welcome!

20.2.1.1. Global registries

- IPv6 test backbone: 6bone, How to join 6bone

20.2.1.2. Major regional registries

- America: ARIN, ARIN / registration page, ARIN / IPv6 guidelines
- EMEA: Ripe NCC, Ripe NCC / registration page, Ripe NCC / IPv6 registration
- Asia/Pacific: APNIC, APNIC / IPv6 information
- Latin America and Caribbea: LACNIC
- Africa: AfriNIC

Also a list of major (prefix length 35) allocations per local registry is available here: Ripe NCC / IPv6 allocations.

20.2.1.3. Tunnel brokers

- Freenet6, Canada
- Hurricane Electric, US backbone
- Centro Studi e Laboratory Telecomunicazioni, Italy
- Wanadoo, Belgium
- CERTNET−Nokia, China
- Tunnelbroker Leipzig, Germany − DialupUsers with dynamic IP's can get a fix IPv6 IP...
- Internet Initiative Japan, Japan − with IPv6 native line service and IPv6 tunneling Service
- XS26 − "Access to Six", Netherland − with POPs in Slovak Republic, Czech Republic, Netherlands, Germany and Hungary.
- IPng Netherland, Netherland − Intouch, SurfNet, AMS−IX, UUNet, Cistron, RIPE NCC and AT&T are connected at the AMS−IX. It is possible (there are requirements...) to get an static tunnel.
- UNINETT, Norway − Pilot IPv6 Service (for Customers): tunnelbroker & address allocation
- NTT Europe, NTT Euroope, United Kingdom − IPv6 Trial. IPv4 Tunnel and native IPv6 leased Line connections. POPs are located in London, UK Dusseldorf, Germany New Jersey, USA (East Coast) Cupertino, USA (West Coast) Tokyo, Japan
- ESnet, USA − Energy Sciences Network: Tunnel Registry & Address Delegation for directly connected ESnet sites and ESnet collaborators.
- 6REN, USA − The 6ren initiative is being coordinated by the Energy Sciences Network (ESnet), the network for the Energy Research program of the US Dept. of Energy, located at the University of California's Lawrence Berkeley National Laboratory

See also here for more information and URLs: ipv6−net.org.

20.2.1.4. 6to4

- NSayer's 6to4 information
- RFC 3068 / An Anycast Prefix for 6to4 Relay Routers

20.2.2. Latest news

More to be filled later...suggestions are welcome!

- hs247 / IPv6 news and information, also homepage for #ipv6 channel on EFnet
- bofh.st / latest IPv6 news (but currently [Jan 2002] outdated...), also homepage for #IPv6 channel on IRCnet
- ipv6−net.org, German forum

20.2.3. Protocol references

More to be filled later...suggestions are welcome!

- Current IEFT drafts of IP Version 6 Working Group (ipv6)
- Network Sorcery / IPv6, Internet Protocol version 6, IPv6 protocol header
- SWITCH IPv6 Pilot / References, big list of IPv6 references maintained by Simon Leinen

20.2.4. More information

More to be filled later...suggestions are welcome!

20.2.4.1. Linux related

- IPv6−HowTo for Linux by Peter Bieringer − Germany, and his Bieringer / IPv6 − software archive
- Linux+IPv6 status by Peter Bieringer − Germany
- USAGI project − Japan, and their USAGI project − software archive
- Gav's Linux IPv6 Page
- Project6 − IPv6 Networking For Linux − Italy, and their Project6 − software archive

20.2.4.2. Linux related per distribution

PLD

Polish(ed) Linux Distribution ("market leader" in containing IPv6 enabled packages)

Red Hat

Red Hat Linux, Pekka Savola's IPv6 packages

Debian

Debian Linux, Craig Small's IPv6 information and status

SuSE

SuSE Linux

Mandrake

Linux Mandrake

For more see the distribution status page

20.2.4.3. General

- IPv6.org
- 6bone
- UK IPv6 Resource Centre − UK
- JOIN: IPv6 information − Germany, by the JOIN project team maintaining also Links to external WWW pages comprising IPv6/IPng
- TIPSTER6 project − Hungary, "Testing Experimental IPv6 Technology and Services in Hungary"
- WIDE project − Japan
- SWITCH IPv6 Pilot − Switzerland
- IPv6 Corner of Hubert Feyrer − Germany
- Vermicelli Project − Norway
- IPv6 Forum − a world−wide consortium of leading Internet vendors, Research & Education Networks...
- Playground.sun.com / IPv6 Info Page − maintained by Robert Hinden, Nokia
- NASA Ames Research Center (old content)
- 6INIT − IPv6 Internet Initiative − an EU Fifth Framework Project under the IST Programme

Something missing? Suggestions are welcome!

20.2.4.4. In Portuguese

- IPv6 pages of Miguel Rosa
- FCCN (National Foundation for the Scientific Computation)
- Grupo de Pesquisa em IPv6 do Brasil
- University of Algarve, Portugal
- IPv6 − MFA

20.2.5. By countries

20.2.5.1. Australia

* Carl's Australian IPv6 Pages (old content)

20.2.5.2. Belgium

* BELNET − the Belgian Research Network
* Euronet − one of the biggest ISP's of Belgium...

20.2.5.3. Germany

* Completel IPv6 information, German ISP
* IPv6−net.org, German IPv6 forum

20.2.5.4. France

* Renater − Renater IPv6 Project Page

20.2.5.5. Italy

* Edisontel − IPv6 Portal of Edisontel

20.2.5.6. Japan

* Yamaha IPv6 (sorry, all in japanese native ...)

20.2.5.7. Korea

* IPv6 Forum Korea − Korean IPv6 Deployment Project

20.2.5.8. Mexico

* IPv6 Mexico (spain & english version) − IPv6 Project Hompeage of The National Autonomous University of Mexico (UNAM)

20.2.5.9. Netherland

* SURFnet − SURFnet IPv6 Backbone
* STACK, STACK (IPv6) − Students' computer association of the Eindhoven University of Technology, Netherland.
* IPng.nl, collaboration between WiseGuys and Intouch.

20.2.5.10. United Kingdom

* British Telecom IPv6 Home − BT's ISP IPv6 Trial, UK's first IPv6 Internet Exchange etc.

20.2.6. By operating systems

20.2.6.1. Cisco IOS

* Cisco IOS IPv6 Entry Page

20.2.6.2. Compaq

* IPv6 at Compaq − Presentations, White Papers, Documentation...

20.2.6.3. Microsoft

* Microsoft Windows 2000 IPv6
* MSRIPv6 − Microsoft Research Network − IPv6 Homepage
* Getting Started with the Microsoft IPv6 Technology Preview for Windows 2000

20.2.6.4. *BSD

* KAME project − Japan, (*BSD)
* NetBSD's IPv6 Networking FAQ

20.2.6.5. Solaris

* Sun Microsystems IPv6 Page for Solaris 8

20.2.6.6. Sumitoma

* Sumitomo Electric has implemented IPv6 on Suminet 3700 family routers

20.2.7. Application lists

* IPv6.org / IPv6 enabled applications
* Freshmeat / IPv6 search, currently (26. January 2001) 46 projects

20.2.7.1. Analyzer tools

- Etherreal – Ethereal is a free network protocol analyzer for Unix and Windows
- Radcom RC100−WL – Download Radcom RC100−WL protocol analyzer version 3.20

20.2.8. Request For Comments (RFCs)

See here for more

- IPv6.org / IPv6 Related Specifications
- Playground.sun.com / IPng Standardization Status

20.3. Online test tools

More to be filled later...suggestions are welcome!

- finger, nslookup, ping, traceroute, whois: UK IPv6 Resource Centre / The test page
- ping, traceroute, tracepath, 6bone registry, DNS: JOIN / Testtools (German language only, but should be no problem for non German speakers)
- traceroute6, whois: IPng.nl

20.4. Maillists

Request e−mail address	What to subscribe	Maillist e−mail address	Language	Focus	Access through WWW
majordomo (at) oss.sgi.com	netdev	netdev (at) oss.sgi.com	English	Linux kernel networking including IPv6	Archive
majordomo (at) list.mipl.mediapoli.com	mipl	mipl (at) list.mipl.mediapoli.com	English	Mobile IP for Linux	Info, Archive
majordomo (at) atlan.uni−muenster.de	ipv6	ipv6 (at) uni−muenster.de	German/English	IPv6/6bone in Germany	Info, Archive
majordomo (at) isi.edu	6bone	6bone (at) isi.edu	English	6bone	Info, Threaded archive, Mirror of archive
majordomo (at) sunroof.eng.sun.com	ipng	ipng (at) sunroof.eng.sun.com	English	IPv6 discussions	Info, Archive, Mirror of archive
majordomo (at) ipv6.org	users	users (at) ipv6.org	English	IPv6 users	Info
usagi−users−ctl (at)		usagi−users (at)	English	Linux IPv6	Info / Search,

linux−ipv6.org		linux−ipv6.org		users using USAGI extension	Archive
bugtraq−subscribe (at) securityfocus.com		bugtraq (at) securityfocus.com (moderated)	English	Bugtracking of Internet applications*	Info, Archive
Web−based, see URL		ipv6 (at) ipng.nl	English	IPv6 in general	Info/Subscription, Archive
majordomo (at) mfa.eti.br	ipv6	ipv6 (at) mfa.eti.br	Portuguese		Info

*very recommended if you provide server applications.

Something missing? Suggestions are welcome!

Another list is available at JOIN Project / List of IPv6−related maillists.

Chapter 21. Revision history / Credits / The End

21.1. Revision history

Versions x.y are published on the Internet.

Versions x.y.z are work−in−progress and only published as LyX file on CVS.

21.1.1. Releases 0.x

0.23

2002−03−27/PB: Add entries in URL list and at maillists, add a label and minor information about IPv6 on RHL

0.22

2002−03−04/PB: Add info about 6to4 support in kernel series 2.2.x and add an entry in URL list and at maillists

0.21

2002−02−26/PB: Migrate next grammar checks submitted by John Ronan

0.20.4

2002−02−21/PB: Migrate more grammar checks submitted by John Ronan, add some additional hints at DNS section

0.20.3

2002−02−12/PB: Migrate a minor grammar check patch submitted by John Ronan

0.20.2

2002−02−05/PB: Add mipl to maillist table

0.20.1

2002−01−31/PB: Add a hint how to generate 6to4 addresses

0.20

2002−01−30/PB: Add a hint about default route problem, some minor updates

0.19.2

2002−01−29/PB: Add many new URLs

0.19.1

2002−01−27/PB: Add some forgotten URLs

0.19

2002−01−25/PB: Add two German books, fix quote entinities in exported SGML code

0.18.2

2002−01−23/PB: Add a FAQ on the program chapter

0.18.1

2002−01−23/PB: Move "the end" to the end, add USAGI to maillists

0.18

2002−01−22/PB: Fix bugs in explanation of multicast address types

0.17.2

2002−01−22/PB: Cosmetic fix double existing text in history (at 0.16), move all credits to the end of the document

0.17.1

2002−01−20/PB: Add a reference, fix URL text in online−test−tools

0.17

2002−01−19/PB: Add some forgotten information and URLs about global IPv6 addresses

0.16

2002−01−19/PB: Minor fixes, remove "bold" and "emphasize" formats on code lines, fix "too long unwrapped code lines" using selfmade utility, extend list of URLs.

0.15

2002−01−15/PB: Fix bug in addresstype/anycast, move content related credits to end of document

0.14

2002−01−14/PB: Minor review at all, new chapter "debugging", review "addresses", spell checking, grammar checking (from beginning to 3.4.1) by Martin Krafft, add tcpdump examples, copy firewalling/netfilter6 from IPv6+Linux−HowTo, minor enhancements

0.13

2002−01−05/PB: Add example BIND9/host, move revision history to end of document, minor extensions

0.12

2002−01−03/PB: Merge review of David Ranch

0.11

2002−01−02/PB: Spell checking and merge review of Pekka Savola

0.10

2002−01−02/PB: First public release of chapter 1

21.2. Credits

The quickest way to be added to this nice list is to send bug fixes, corrections, and/or updates to me ;−).

If you want to do a major review, you can use the native LyX file (see original source) and send diffs against it, because diffs against SGML don't help too much.

21.2.1. Major credits

- David Ranch <dranch at trinnet dot net>: For encouraging me to write this HOWTO, his editorial comments on the first few revisions, and his contributions to various IPv6 testing results on my IPv6 web site. Also for his major reviews and suggestions.
- Pekka Savola <pekkas at netcore dot fi>: For major reviews, input and suggestions.
- Martin F. Krafft <madduck at madduck dot net>: For grammar checks and general reviewing of the document.
- John Ronan <j0n at tssg dot wit dot ie>: For grammar checks.

21.2.2. Other credits

21.2.2.1. Document technique related

Writing a LDP HOWTO as a newbie (in LyX and exporting this to DocBook to conform to SGML) isn't as easy as some people say. There are some strange pitfalls... Nevertheless, thanks to:

- Authors of the LDP Author Guide
- B. Guillon: For his DocBook with LyX HOWTO

21.2.2.2. Content related credits

Credits for fixes and hints are listed here, will grow sure in the future

- S .P. Meenakshi <meena at cs dot iitm dot ernet dot in>: For a hint using a "send mail" shell program on tcp_wrapper/hosts.deny
- Georg Käfer <gkaefer at salzburg dot co dot at>: For detection of no proper PDF creation (fixed now by LDP maintainer Greg Ferguson), input for German books, big list of URLs and some other suggestions
- Frank Dinies <FrankDinies at web dot de>: For a bugfix on IPv6 address explanation
- John Freed <jfreed at linux−mag dot com>: For finding a bug in in IPv6 multicast address explanation
- Craig Rodrigues <crodrigu at bbn dot com>: For suggestion about RHL IPv6 setup

---

21.3. The End

Thanks for reading. Hope it helps!

If you have any questions, subscribe to proper maillist and describe your problem.