Chapter 23. Introduction to TCP/IP Networking

Table of Contents

23.1. Audience

This section explains various aspects of networking. It is intended to help people with little knowledge about networks to get started. It is divided into three big parts. We start by giving a general overview of how networking works and introduce the basic concepts. Then we go into details for setting up various types of networking in the second parts, and the third part of the networking section covers a number of “advanced” topics that go beyond the basic operation as introduced in the first two sections.

The reader is assumed to know about basic system administration tasks: how to become root, edit files, change permissions, stop processes, etc. See the other chapters of this NetBSD guide and, e.g., [AeleenFrisch] for further information on this topic. Besides that, you should know how to handle the utilities we're going to set up here, i.e., you should know how to use telnet, FTP, ... I will not explain the basic features of those utilities, please refer to the appropriate manual pages, the references listed or, of course, the other parts of this document instead.

This introduction to TCP/IP networking was written with the intention in mind to give starters a basic knowledge. If you really want to know what it's all about, read [CraigHunt]. This book does not only cover the basics, but goes on and explains all the concepts, services and how to set them up in detail. It's great, I love it! :-)

23.2. Supported Networking Protocols

There are several protocol suites supported by NetBSD, most of which were inherited from NetBSD's predecessor, 4.4BSD, and subsequently enhanced and improved. The first and most important one today is DARPA's Transmission Control Protocol/Internet Protocol (TCP/IP). Other protocol suites available in NetBSD include the Stream Control Transmission Protocol (SCTP), and Apple's AppleTalk protocol suite. They are only used in some special applications.

Today, TCP/IP is the most widespread protocol of the ones mentioned above. It is implemented on almost every hardware and operating system, and it is also the most-used protocol in heterogenous environments. So, if you just want to connect your computer running NetBSD to some other machine at home or you want to integrate it into your company's or university's network, TCP/IP is the right choice.

23.3. Supported Media

The TCP/IP protocol stack behaves the same regardless of the underlying media used, and NetBSD supports a wide range of these, among them are Ethernet (10/100Mb/1/10/40/100Gb), USB, serial line and FireWire (IEEE 1394).

23.3.1. Ethernet

Ethernet is the medium commonly used to build local area networks (LANs) of interconnected machines within a limited area such as an office, company or university campus. Ethernet is based on a bus structure to which many machines can connect to, and communication always happens between two nodes at a time. When two or more nodes want to talk at the same time, both will restart communication after some timeout. The technical term for this is CSMA/CD (Carrier Sense w/ Multiple Access and Collision Detection).

Initially, Ethernet hardware consisted of a thick (yellow) cable that machines tapped into using special connectors that poked through the cable's outer shielding. The successor of this was called 10base5, which used BNC-type connectors for tapping in special T-connectors and terminators on both ends of the bus. Today, ethernet is mostly used with twisted pair lines which are used in a collapsed bus system that are contained in switches or hubs. The twisted pair lines give this type of media its name - 10baseT for 10 Mbit/s networks, and 100baseT for 100 MBit/s ones. In switched environments there's also the distinction if communication between the node and the switch can happen in half- or in full duplex mode.

23.3.2. IEEE 802.11 (Wi-Fi)

IEEE 802.11 (commonly known as Wi-Fi) is the primary means by which mobile devices are connected over a Local Area Network.

IEEE 802.11 primarily operates over two radio bands, 2.4 GHz (modes b, g, and n), and 5 GHz (modes a, and ac). The 2.4 GHz band is typically more congested, but loses less performance through walls and other barriers.

The main protocol used for securing Wi-Fi connections is WPA (Wi-Fi Protected Access). In a typical configuration, this encrypts the connection between the access point and its clients with a password.

23.3.3. Serial Line

The disadvantage of a serial connection is that it's slower than other methods. NetBSD can use at most 115200 bit/s, making it a lot slower than e.g. Ethernet's minimum 10 Mbit/s.

There are two possible protocols to connect a host running NetBSD to another host using a serial line (possibly over a phone-line):

- Serial Line IP (SLIP)
- Point to Point Protocol (PPP)

The choice here depends on whether you use a dial-up connection through a modem or if you use a static connection (null-modem or leased line). If you dial up for your IP connection, it's wise to use PPP as it offers some possibilities to auto-negotiate IP-addresses and routes, which can be quite painful to do by hand. If you want to connect to another machine which is directly connected, use SLIP, as this is supported by about every operating system and more easy to set up with fixed addresses and routes.

PPP on a direct connection is a bit difficult to setup, as it's easy to timeout the initial handshake; with SLIP, there's no such initial handshake, i.e. you start up one side, and when the other site has its first packet, it will send it over the line.

[RFC1331] and [RFC1332] describe PPP and TCP/IP over PPP. SLIP is defined in [RFC1055].

23.4. TCP/IP Address Format

TCP/IP uses 4-byte (32-bit) addresses in the current implementations (IPv4), also called IP-numbers (Internet-Protocol numbers), to address hosts.

TCP/IP allows any two machines to communicate directly. To permit this all hosts on a given network must have a unique IP address. To assure this, IP addresses are administrated by one central organisation, the InterNIC. They give certain ranges of addresses (network-addresses) directly to sites which want to participate in the internet or to internet-providers, which give the addresses to their customers.

If your university or company is connected to the Internet, it has (at least) one such network-address for its own use, usually not assigned by the InterNIC directly, but rather through an Internet Service Provider (ISP).

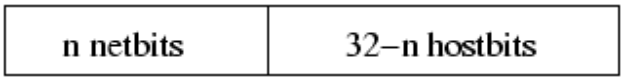
If you just want to run your private network at home, see below on how to “build” your own IP addresses. However, if you want to connect your machine to the (real :-) Internet, you should get an IP addresses from your local network-administrator or -provider.

IP addresses are usually written in “dotted quad”-notation - the four bytes are written down in decimal (most significant byte first), separated by dots. For example, 132.199.15.99 would be a valid address. Another way to write down IP-addresses would be as one 32-bit hex-word, e.g. 0x84c70f63. This is not as convenient as the dotted-quad, but quite useful at times, too. (See below!)

Being assigned a network means nothing else but setting some of the above-mentioned 32 address-bits to certain values. These bits that are used for

identifying the network are called network-bits. The remaining bits can be used to address hosts on that network, therefore they are called host-bits. Figure 23.1, “IPv4-addresses are divided into more significant network- and less significant hostbits” illustrates the separation.

Figure 23.1. IPv4-addresses are divided into more significant network- and less significant hostbits



In the above example, the network-address is 132.199.0.0 (host-bits are set to 0 in network-addresses) and the host's address is 15.99 on that network.

How do you know that the host's address is 16 bit wide? Well, this is assigned by the provider from which you get your network-addresses. In the classless inter-domain routing (CIDR) used today, host fields are usually between as little as 2 to 16 bits wide, and the number of network-bits is written after the network address, separated by a “/”, e.g. 132.199.0.0/16 tells that the network in question has 16 network-bits. When talking about the “size” of a network, it's usual to only talk about it as “/16”, “/24”, etc.

Before CIDR was used, there used to be four classes of networks. Each one starts with a certain bit-pattern identifying it. Here are the four classes:

- Class A starts with “0” as most significant bit. The next seven bits of a class A address identify the network, the remaining 24 bit can be used to address hosts. So, within one class A network there can be 2^{24} hosts. It's not very likely that you (or your university, or company, or whatever) will get a whole class A address.

  The CIDR notation for a class A network with its eight network bits is an “/8”.

- Class B starts with “10” as most significant bits. The next 14 bits are used for the networks address and the remaining 16 bits can be used to address more than 65000 hosts. Class B addresses are very rarely given out today, they used to be common for companies and universities before IPv4 address space went scarce.

  The CIDR notation for a class B network with its 16 network bits is an “/16”.

  Returning to our above example, you can see that 132.199.15.99 (or 0x84c70f63, which is more appropriate here!) is on a class B network, as 0x84... = **10**00... (base 2).

  Therefore, the address 132.199.15.99 can be split into an network-address of 132.199.0.0 and a host-address of 15.99.

- Class C is identified by the MSBs being “110”, allowing only 256 (actually: only 254, see below) hosts on each of the 2^{21} possible class C networks. Class C addresses are usually found at (small) companies.

  The CIDR notation for a class C network with its 24 network bits is an “/24”.

- There are also other addresses, starting with “111”. Those are used for special purposes (e. g. multicast-addresses) and are not of interest here.

Please note that the bits which are used for identifying the network-class are part of the network-address.

When separating host-addresses from network-addresses, the “netmask” comes in handy. In this mask, all the network-bits are set to “1”, the host-bits are “0”. Thus, putting together IP-address and netmask with a logical AND-function, the network-address remains.

To continue our example, 255.255.0.0 is a possible netmask for 132.199.15.99. When applying this mask, the network-address 132.199.0.0 remains.

For addresses in CIDR notation, the number of network-bits given also says how many of the most significant bits of the address must be set to “1” to get the netmask for the corresponding network. For classful addressing, every network-class has a fixed default netmask assigned:

- Class A (/8): default-netmask: 255.0.0.0, first byte of address: 1-127
- Class B (/16): default-netmask: 255.255.0.0, first byte of address: 128-191
- Class C (/24): default-netmask: 255.255.255.0, first byte of address: 192-223

Another thing to mention here is the “broadcast-address”. When sending to this address, *all* hosts on the corresponding network will receive the message sent. The broadcast address is characterized by having all host-bits set to “1”.

Taking 132.199.15.99 with its netmask 255.255.0.0 again, the broadcast-address would result in 132.199.255.255.

You'll ask now: But what if I want a host's address to be all bits “0” or “1”? Well, this doesn't work, as network- and broadcast-address must be present! Because of this, a class B (/16) network can contain at most 2^{16}-2 hosts, a class C (/24) network can hold no more than 2^{8}-2 = 254 hosts.

Besides all those categories of addresses, there's the special IP-address 127.0.0.1 which always refers to the “local” host, i.e. if you talk to 127.0.0.1 you'll talk to yourself without starting any network-activity. This is sometimes useful to use services installed on your own machine or to play around if you don't have other hosts to put on your network.

Let's put together the things we've introduced in this section:

IP-address

: 32 bit-address, with network- and host-bits.

Network-address

: IP-address with all host bits set to “0”.

Netmask

: 32-bit mask with “1” for network- and “0” for host-bits.

Broadcast

: IP-address with all host bits set “1”.

localhost's address

The local host's IP address is always 127.0.0.1.

23.5. Subnetting and Routing

After talking so much about netmasks, network-, host- and other addresses, I have to admit that this is not the whole truth.

Imagine the situation at your university, which usually has a class B (/16) address, allowing it to have up to 2^{16} ~= 65534 hosts on that net. Maybe it would be a nice thing to have all those hosts on one single network, but it's simply not possible due to limitations in the transport media commonly used today.

For example, when using thinwire ethernet, the maximum length of the cable is 185 meters. Even with repeaters in between, which refresh the signals, this is not enough to cover all the locations where machines are located. Besides that, there is a maximum number of 1024 hosts on one ethernet wire, and you'll lose quite a bit of performance if you go to this limit.

So, are you hosed now? Having an address which allows more than 60000 hosts, but being bound to media which allows far less than that limit?

Well, of course not! :-)

The idea is to divide the “big” class B net into several smaller networks, commonly called sub-networks or simply subnets. Those subnets are only allowed to have, say, 254 hosts on them (i.e. you divide one big class B network into several class C networks!).

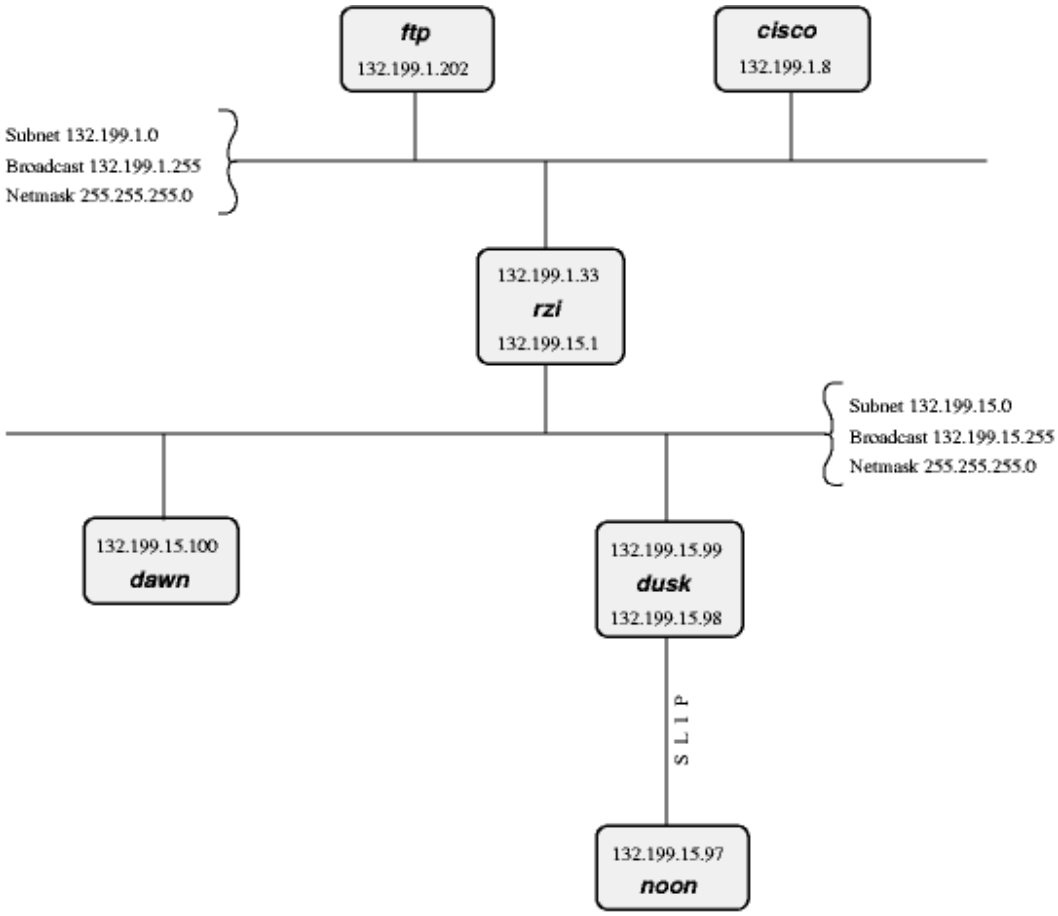
To do this, you adjust your netmask to have more network- and less host-bits on it. This is usually done on a byte-boundary, but you're not forced to do it there. So, commonly your netmask will not be 255.255.0.0 as supposed by a class B network, but it will be set to 255.255.255.0.

In CIDR notation, you now write a “/24” instead of the “/16” to show that 24 bits of the address are used for identifying the network and subnet, instead of the 16 that were used before.

This gives you one additional network-byte to assign to each (physical!) network. All the 254 hosts on that subnet can now talk directly to each other, and you can build 256 such class C nets. This should fit your needs.

To explain this better, let's continue our above example. Say our host 132.199.15.99 (I'll call him dusk from now; we'll talk about assigning hostnames later) has a netmask of 255.255.255.0 and thus is on the subnet 132.199.15.0/24. Let's furthermore introduce some more hosts so we have something to play around with, see Figure 23.2, “Our demo-network”.

Figure 23.2. Our demo-network



In the above network, dusk can talk directly to dawn, as they are both on the same subnet. (There are other hosts attached to the 132.199.15.0/24-subnet but they are not of importance for us now)

But what if dusk wants to talk to a host on another subnet?

Well, the traffic will then go through one or more gateways (routers), which are attached to two subnets. Because of this, a router always has two different addresses, one for each of the subnets it is on. The router is functionally transparent, i.e. you don't have to address it to reach hosts on the “other” side. Instead, you address that host directly and the packets will be routed to it correctly.

Example. Let's say dusk wants to get some files from the local ftp-server. As dusk can't reach `ftp` directly (because it's on a different subnet), all its packets will be forwarded to its "defaultrouter" `rzi` (132.199.15.1), which knows where to forward the packets.

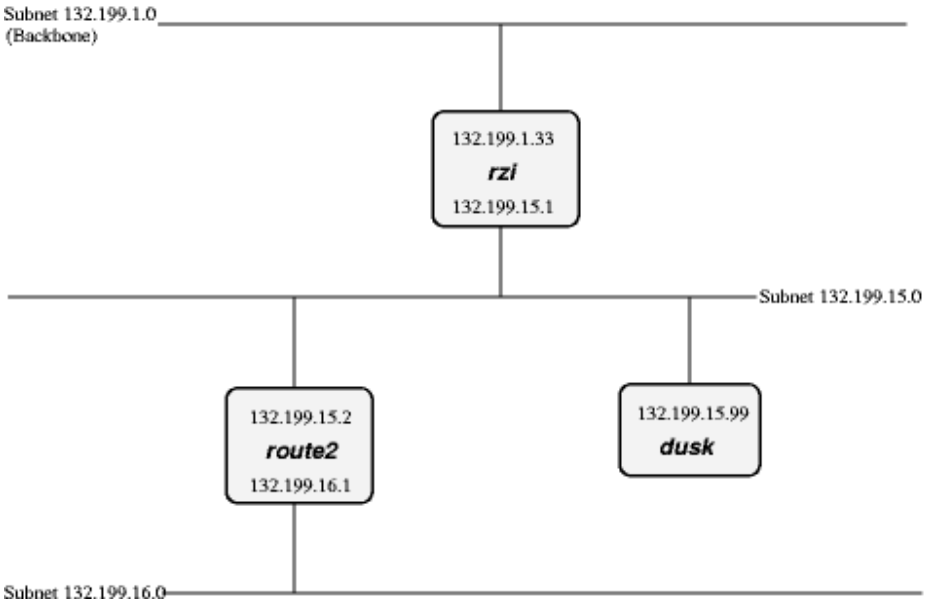
Dusk knows the address of its defaultrouter in its network (`rzi`, 132.199.15.1), and it will forward any packets to it which are not on the same subnet, i.e. it will forward all IP-packets in which the third address-byte isn't 15.

The (default)router then gives the packets to the appropriate host, as it's also on the FTP-server's network.

In this example, *all* packets are forwarded to the 132.199.1.0/24-network, simply because it's the network's backbone, the most important part of the network, which carries all the traffic that passes between several subnets. Almost all other networks besides 132.199.15.0/24 are attached to the backbone in a similar manner.

But what if we had hooked up another subnet to 132.199.15.0/24 instead of 132.199.1.0/24? Maybe something the situation displayed in Figure 23.3, “Attaching one subnet to another one”.

Figure 23.3. Attaching one subnet to another one



When we now want to reach a host which is located in the 132.199.16.0/24-subnet from dusk, it won't work routing it to `rzi`, but you'll have to send it directly to `route2` (132.199.15.2). Dusk will have to know to forward those packets to `route2` and send all the others to `rzi`.

When configuring dusk, you tell it to forward all packets for the 132.199.16.0/24-subnet to `route2`, and all others to `rzi`. Instead of specifying this default as 132.199.1.0/24, 132.199.2.0/24, etc., 0.0.0.0 can be used to set the default-route.

Returning to Figure 23.2, “Our demo-network”, there's a similar problem when dawn wants to send to noon, which is connected to dusk via a serial line running. When looking at the IP-addresses, noon seems to be attached to the 132.199.15.0-network, but it isn't really. Instead, dusk is used as gateway, and dawn will have to send its packets to dusk, which will forward them to noon then. The way dusk is forced into accepting packets that aren't destined at it but for a different host (noon) instead is called “proxy arp”.

The same goes when hosts from other subnets want to send to noon. They have to send their packets to dusk (possibly routed via `rzi`),

23.6. Name Service Concepts

In the previous sections, when we talked about hosts, we referred to them by their IP-addresses. This was necessary to introduce the different kinds of addresses. When talking about hosts in general, it's more convenient to give them “names”, as we did when talking about routing.

Most applications don't care whether you give them an IP address or a hostname. However, they'll use IP addresses internally, and there are several methods for them to map hostnames to IP addresses, each one with its own way of configuration. In this section we'll introduce the idea behind each method, in the next chapter, we'll talk about the configuration-part.

The mapping from hostnames (and domainnames) to IP-addresses is done by a piece of software called the “resolver”. This is not an extra service, but some library routines which are linked to every application using networking-calls. The resolver will then try to resolve (hence the name ;-) the hostnames you give into IP addresses. See [RFC1034] and [RFC1035] for details on the resolver.

Hostnames are usually up to 256 characters long, and contain letters, numbers and dashes (“-”); case is ignored.

Just as with networks and subnets, it's possible (and desirable) to group hosts into domains and subdomains. When getting your network-address, you usually also obtain a domainname by your provider. As with subnets, it's up to you to introduce subdomains. Other as with IP-addresses, (sub)domains are not directly related to (sub)nets; for example, one domain can contain hosts from several subnets.

Figure 23.2, “Our demo-network” shows this: Both subnets 132.199.1.0/24 and 132.199.15.0/24 (and others) are part of the subdomain “`rz.uni-regensburg.de`”. The domain the University of Regensburg got from its IP-provider is “`uni-regensburg.de`” (“`.de`” is for Deutschland, Germany), the subdomain “`rz`” is for Rechenzentrum, computing center.

Hostnames, subdomain- and domainnames are separated by dots (“.”). It's also possible to use more than one stage of subdomains, although this is not very common. An example would be `fox_in.socs.uts.edu.au`.

A hostname which includes the (sub)domain is also called a fully qualified domain name (FQDN). For example, the IP-address 132.199.15.99 belongs to the host with the FQDN `dusk.rz.uni-regensburg.de`.

Further above I told you that the IP-address 127.0.0.1 always belongs to the local host, regardless what's the “real” IP-address of the host. Therefore, 127.0.0.1 is always mapped to the name “`localhost`”.

The three different ways to translate hostnames into IP addresses are: `/etc/hosts`, the Domain Name Service (DNS) and the Network Information Service (NIS).

23.6.1. /etc/hosts

The first and simplest way to translate hostnames into IP-addresses is by using a table telling which IP address belongs to which hostname(s). This table is stored in the file `/etc/hosts` and has the following format:

```
IP-address        hostname [nickname [...]]
```

Lines starting with a hash mark (“#”) are treated as comments. The other lines contain one IP-address and the corresponding hostname(s).

It's not possible for a hostname to belong to several IP addresses, even if I made you think so when talking about routing. `rzi` for example has really two distinct names for each of its two addresses: `rzi` and `rzia` (but please don't ask me which name belongs to which address!).

Giving a host several nicknames can be convenient if you want to specify your favorite host providing a special service with that name, as is commonly done with FTP-servers. The first (leftmost) name is usually the real (canonical) name of the host.

Besides giving nicknames, it's also convenient to give a host's full name (including domain) as its canonical name, and using only its hostname (without domain) as a nickname.

Important: There *must* be an entry mapping localhost to 127.0.0.1 in `/etc/hosts`!

23.6.2. Domain Name Service (DNS)

`/etc/hosts` bears an inherent problem, especially in big networks: when one host is added or one host's address changes, all the `/etc/hosts` files on all machines have to be changed! This is not only time-consuming, it's also very likely that there will be some errors and inconsistencies, leading to problems.

Another approach is to hold only one hostnames-table (-database) for a network, and make all the clients query that “nameserver”. Updates will be made only on the nameserver.

This is the basic idea behind the Domain Name Service (DNS).

Usually, there's one nameserver for each domain (hence DNS), and every host (client) in that domain knows which domain it is in and which nameserver to query for its domain.

When the DNS gets a query about a host which is not in its domain, it will forward the query to a DNS which is either the DNS of the domain in question or knows which DNS to ask for the specified domain. If the DNS forwarded the query doesn't know how to handle it, it will forward that query again to a DNS one step higher. This is not ad infinitum, there are several “root”-servers, which know about any domain.

See Chapter 26, *The Domain Name System* for details on DNS.

23.6.3. Network Information Service (NIS/YP)

Yellow Pages (YP) was invented by Sun Microsystems. The name has been changed into Network Information Service (NIS) because YP was already a trademark of the British telecom. So, when I'm talking about NIS you'll know what I mean. ;-)

There are quite some configuration files on a Unix-system, and often it's desired to maintain only one set of those files for a couple of hosts. Those hosts are grouped together in a NIS-domain (which has *nothing* to do with the domains built by using DNS!) and are usually contained in one workstation cluster.

Examples for the config-files shared among those hosts are `/etc/passwd`, `/etc/group` and - last but not least - `/etc/hosts`.

So, you can “abuse” NIS for getting a unique name-to-address-translation on all hosts throughout one (NIS-)domain.

There's only one drawback, which prevents NIS from actually being used for that translation: In contrast to the DNS, NIS provides no way to resolve hostnames which are not in the hosts-table. There's no hosts “one level up” which the NIS-server can query, and so the translation will fail! Suns NIS+ takes measures against that problem, but as NIS+ is only available on Solaris-systems, this is of little use for us now.

Don't get me wrong: NIS is a fine thing for managing e.g. user-information (`/etc/passwd`, ...) in workstation-clusters, it's simply not too useful for resolving hostnames.

23.6.4. Other

The name resolving methods described above are what's used commonly today to resolve hostnames into IP addresses, but they aren't the only ones. Basically, every database mechanism would do, but none is implemented in NetBSD. Let's have a quick look what you may encounter.

With NIS lacking hierarchy in data structures, NIS+ is intended to help out in that field. Tables can be setup in a way so that if a query cannot be answered by a domain's server, there can be another domain “above” that might be able to do so. E.g. you could choose to have a domain that lists all the hosts (users, groups, ...) that are valid in the whole company, one that defines the same for each division, etc. NIS+ is not used a lot today, even Sun went back to ship back NIS by default.

Last century, the X.500 standard was designed to accommodate both simple databases like `/etc/hosts` as well as complex, hierarchical systems as can be found e.g. in DNS today. X.500 wasn't really a success, mostly due to the fact that it tried to do too much at the same time. A cut-down version is available today as the Lightweight Directory Access Protocol (LDAP), which is becoming popular in the last years to manage data like users but also hosts and others in small to medium sized organisations.

23.7. IPv6

23.7.1. What good is IPv6?

When telling people to migrate from IPv4 to IPv6, the question you usually hear is “why?”. There are actually a few good reasons to move to the new version:

- Bigger address space
- Support for mobile devices
- Built-in security

23.7.1.1. Bigger Address Space

The bigger address space that IPv6 offers is the most obvious enhancement it has over IPv4. While today's internet architecture is based on 32-bit wide addresses, the new version has 128 bit available for addressing. Thanks to the enlarged address space, work-arounds like NAT don't have to be used any more. This allows full, unconstrained IP connectivity for today's mobile phones and IoT devices.

23.7.1.2. Mobility

When mentioning mobile devices and IP, another important point to note is that some special protocol is needed to support mobility, and implementing this protocol - called “Mobile IP” - is one of the requirements for every IPv6 stack. Thus, if you have IPv6 going, you have support for roaming between different networks,

with everyone being updated when you leave one network and enter the other one. Support for roaming is possible with IPv4 too, but there are a number of hoops that need to be jumped in order to get things working. With IPv6, there's no need for this, as support for mobility was one of the design requirements for IPv6. See [RFC3024] for some more information on the issues that need to be addressed with Mobile IP on IPv4.

23.7.1.3. Security

Besides support for mobility, security was another requirement for the successor to today's Internet Protocol version. As a result, IPv6 protocol stacks are required to include IPsec. IPsec allows authentication, encryption and compression of any IP traffic. Unlike application level protocols like SSL or SSH, all IP traffic between two nodes can be handled, without adjusting any applications. The benefit of this is that all applications on a machine can benefit from encryption and authentication, and that policies can be set on a per-host (or even per-network) base, not per application/service. An introduction to IPsec with a roadmap to the documentation can be found in [RFC2411], the core protocol is described in [RFC2401].

23.7.2. Changes to IPv4

After giving a brief overview of all the important features of IPv6, we'll go into the details of the basics of IPv6 here. A brief understanding of how IPv4 works is assumed, and the changes in IPv6 will be highlighted. Starting with IPv6 addresses and how they're split up we'll go into the various types of addresses there are, what became of broadcasts, then after discussing the IP layer go into changes for name resolving and what's new in DNS for IPv6.

23.7.2.1. Addressing

An IPv4 address is a 32 bit value, that's usually written in “dotted quad” representation, where each “quad” represents a byte value between 0 and 255, for example:

127.0.0.1

This allows a theoretical number of 2^{32} or ~4 billion hosts to be connected on the internet today. Due to grouping, not all addresses are available today.

IPv6 addresses use 128 bit, which results in 2^{128} theoretically addressable hosts. This allows for a Really Big number of machines to addressed, and it sure fits all of today's requirements plus all those nifty PDAs and cell phones with IP phones in the near future without any sweat. When writing IPv6 addresses, they are usually divided into groups of 16 bits written as four hex digits, and the groups are separated by colons. An example is:

fe80::2a0:d2ff:fea5:e9f5

This shows a special thing - a number of consecutive zeros can be abbreviated by a single “::” once in the IPv6 address. The above address is thus equivalent to fe80:0:00:000:2a0:d2ff:fea5:e9f5 - leading zeros within groups can be omitted, and only one “::” can be used in an IPv6 address.

To make addresses manageable, they are split in two parts, which are the bits identifying the network a machine is on, and the bits that identify a machine on a (sub)network. The bits are known as netbits and hostbits, and in both IPv4 and IPv6, the netbits are the “left”, most significant bits of an IP address, and the host bits are the “right”, least significant bits, as shown in Figure 23.4, “IPv6-addresses are divided into more significant network- and less significant hostbits, too”.

Figure 23.4. IPv6-addresses are divided into more significant network- and less significant hostbits, too

In IPv4, the border is drawn with the aid of the netmask, which can be used to mask all net/host bits. Typical examples are 255.255.0.0 that uses 16 bit for addressing the network, and 16 bit for the machine, or 255.255.255.0 which takes another 8 bit to allow addressing 256 subnets on e.g. a class B net.

When addressing switched from classful addressing to CIDR routing, the borders between net and host bits stopped being on 8 bit boundaries, and as a result the netmasks started looking ugly and not really manageable. As a replacement, the number of network bits is used for a given address, to denote the border, e.g.

10.0.0.0/24

is the same as a netmask of 255.255.255.0 (24 1-bits). The same scheme is used in IPv6:

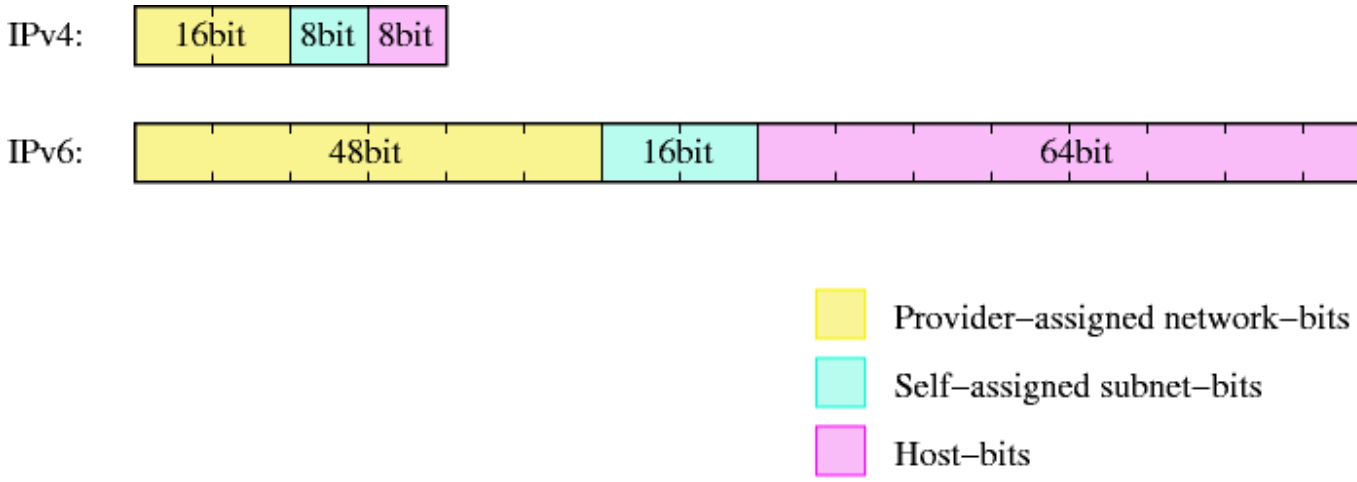
2001:638:a01:2::/64

tells us that the address used here has the first (leftmost) 64 bits used as the network address, and the last (rightmost) 64 bits are used to identify the machine on the network. The network bits are commonly referred to as (network) “prefix”, and the “prefixlen” here would be 64 bits.

Common addressing schemes found in IPv4 are the (old) class B and class C nets. With a class C network (/24), you get 24 bits assigned by your provider, and it leaves 8 bits to be assigned by you. If you want to add any subnetting to that, you end up with “uneven” netmasks that are a bit nifty to deal with. Easier for such cases are class B networks (/16), which only have 16 bits assigned by the provider, and that allow subnetting, i.e. splitting of the rightmost bits into two parts. One to address the on-site subnet, and one to address the hosts on that subnet. Usually, this is done on byte (8 bit) boundaries. Using a netmask of 255.255.255.0 (or a /24 prefix) allows flexible management even of bigger networks here. Of course there is the upper limit of 254 machines per subnet, and 256 subnets.

With 128 bits available for addressing in IPv6, the scheme commonly used is the same, only the fields are wider. Providers usually assign /48 networks, which leaves 16 bits for a subnetting and 64 hostbits.

Figure 23.5. IPv6-addresses have a similar structure to class B addresses



Now while the space for network and subnets here is pretty much ok, using 64 bits for addressing hosts seems like a waste. It's unlikely that you will want to have several billion hosts on a single subnet, so what is the idea behind this?

The idea behind fixed width 64 bit wide host identifiers is that they aren't assigned manually as it's usually done for IPv4 nowadays. Instead, IPv6 host addresses are recommended (not mandatory!) to be built from so-called EUI64 addresses. EUI64 addresses are - as the name says - 64 bit wide, and derived from MAC addresses of the underlying network interface. E.g. for ethernet, the 6 byte (48 bit) MAC address is usually filled with the hex bits “fffe” in the middle and a bit is set to mark the address as unique (which is true for Ethernet), e.g. the MAC address

01:23:45:67:89:ab

results in the EUI64 address

03:23:45:ff:fe:67:89:ab

which again gives the host bits for the IPv6 address as

::0323:45ff:fe67:89ab

These host bits can now be used to automatically assign IPv6 addresses to hosts, which supports autoconfiguration of IPv6 hosts - all that's needed to get a complete IPv6 address is the first (net/subnet) bits, and IPv6 also offers a solution to assign them automatically.

When on a network of machines speaking IP, there's usually one router which acts as the gateway to outside networks. In IPv6 land, this router will send “router advertisement” information, which clients are expected to either receive during operation or to solicit upon system startup. The router advertisement information includes data on the router's address, and which address prefix it routes. With this information and the host-generated EUI64 address, an IPv6-host can calculate its IP address, and there is no need for manual address assignment. Of course routers still need some configuration.

The router advertisement information they create are part of the Neighbor Discovery Protocol (NDP, see [RFC2461]), which is the successor to IPv4's ARP protocol. In contrast to ARP, NDP does not only do lookup of IPv6 addresses for MAC addresses (the neighbor solicitation/advertisement part), but also does a similar service for routers and the prefixes they serve, which is used for autoconfiguration of IPv6 hosts as described in the previous paragraph.

23.7.2.2. Multiple Addresses

In IPv4, a host usually has one IP address per network interface or even per machine if the IP stack supports it. Only very rare applications like web servers result in machines having more than one IP address. In IPv6, this is different. For each interface, there is not only a globally unique IP address, but there are two other addresses that are of interest: The link local address, and the site local address. The link local address has a prefix of fe80::/64, and the host bits are built from the interface's EUI64 address. The link local address is used for contacting hosts and routers on the same network only, the addresses are not visible or reachable from different subnets. If wanted, there's the choice of either using global addresses (as assigned by a provider), or using site local addresses. Site local addresses are assigned the network address fec0::/10, and subnets and hosts can be addressed just as for provider-assigned networks. The only difference is, that the addresses will not be visible to outside machines, as these are on a different network, and their “site local” addresses are in a different physical net (if assigned at all). As with the 10/8 network in IPv4, site local addresses can be used, but don't have to. For IPv6 it's most common to have hosts assigned a link-local and a global IP address. Site local addresses are rather uncommon today, and are no substitute for globally unique addresses if global connectivity is required.

23.7.2.3. Multicasting

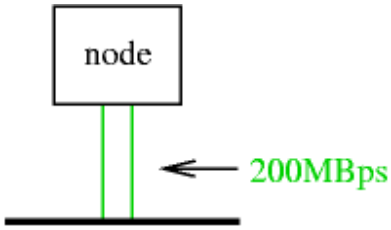
In IP land, there are three ways to talk to a host: unicast, broadcast and multicast. The most common one is by talking to it directly, using its unicast address. In IPv4, the unicast address is the “normal” IP address assigned to a single host, with all address bits assigned. The broadcast address used to address all hosts in the same IP subnet has the network bits set to the network address, and all host bits set to “1” (which can be easily done using the netmask and some bit operations). Multicast addresses are used to reach a number of hosts in the same multicast group, which can be machines spread over the whole internet. Machines must join multicast groups explicitly to participate, and there are special IPv4 addresses used for multicast addresses, allocated from the 224/8 subnet. Multicast isn't used very much in IPv4, and only few applications like the MBone audio and video broadcast utilities use it.

In IPv6, unicast addresses are used the same as in IPv4, no surprise there - all the network and host bits are assigned to identify the target network and machine. Broadcasts are no longer available in IPv6 in the way they were in IPv4, this is where multicasting comes into play. Addresses in the ff::/8 network are reserved for multicast applications, and there are two special multicast addresses that supersede the broadcast addresses from IPv4. One is the “all routers” multicast address, the others is for “all hosts”. The addresses are specific to the subnet, i.e. a router connected to two different subnets can address all hosts/routers on any of the subnets it's connected to. Addresses here are:

- ff0*X*::1 for all hosts and
- ff0*X*::2 for all routers,

where “*X*” is the scope ID of the link here, identifying the network. Usually this starts from “1” for the “node local” scope, “2” for the first link, etc. Note that it's perfectly ok for two network interfaces to be attached to one link, thus resulting in double bandwidth:

Figure 23.6. Several interfaces attached to a link result in only one scope ID for the link



One use of the “all hosts” multicast is in the neighbor solicitation code of NDP, where any machine that wants to communicate with another machine sends out a request to the “all hosts” group, and the machine in question is expected to respond.

23.7.2.4. Name Resolving in IPv6

After talking a lot about addressing in IPv6, anyone still here will hope that there's a proper way to abstract all these long & ugly IPv6 addresses with some nice hostnames as one can do in IPv4, and of course there is.

Hostname to IP address resolving in IPv4 is usually done in one of three ways: using a simple table in `/etc/hosts`, by using the Network Information Service (NIS, formerly YP) or via the Domain Name System (DNS).

As of this writing, NIS/NIS+ over IPv6 is currently only available on Solaris 8, for both database contents and transport, using a RPC extension.

Having a simple address<->name map like `/etc/hosts` is supported in all IPv6 stacks. With the KAME implementation used in NetBSD, `/etc/hosts` contains IPv6 addresses as well as IPv4 addresses. A simple example is the “localhost” entry in the default NetBSD installation:

```
127.0.0.1               localhost
::1                     localhost
```

For DNS, there are no fundamentally new concepts. IPv6 name resolving is done with AAAA records that - as the name implies - point to an entity that's four times the size of an A record. The AAAA record takes a hostname on the left side, just as A does, and on the right side there's an IPv6 address, e.g.

```
noon            IN      AAAA
3ffe:400:430:2:240:95ff:fe40:4385
```

For reverse resolving, IPv4 uses the in-addr.arpa zone, and below that it writes the bytes (in decimal) in reversed order, i.e. more significant bytes are more right. For IPv6 this is similar, only that hex digits representing 4 bits are used instead of decimal numbers, and the resource records are also under a different domain, ip6.int.

So to have the reverse resolving for the above host, you would put into your `/etc/named.conf` something like:

```
zone "0.3.4.0.0.0.4.0.e.f.f.3.IP6.INT" {
      type master;
      file "db.reverse";
};
```

and in the zone file db.reverse you put (besides the usual records like SOA and NS):

```
5.8.3.4.0.4.e.f.f.f.5.9.0.4.2.0.2.0.0.0 IN   PTR
noon.ipv6.example.com.
```

The address is reversed here, and written down one hex digit after the other, starting with the least significant (rightmost) one, separating the hex digits with dots, as usual in zone files.

One thing to note when setting up DNS for IPv6 is to take care of the DNS software version in use. BIND 8.x does understand AAAA records, but it does not offer name resolving via IPv6. You need BIND 9.x for that. Beyond that, BIND 9.x supports a number of resource records that are currently being discussed but not officially introduced yet. The most noticeable one here is the A6 record which allows easier provider/prefix changing.

To sum up, this section talked about the technical differences between IPv4 and IPv6 for addressing and name resolving. Some details like IP header options, QoS and flows were deliberately left out to not make this document more complex than necessary.

Prev | Up | Next

Part IV. Networking and related issues | Home | Chapter 24. Setting up TCP/IP on NetBSD in practice