Certifiable n[u|e]tcase

Cailin takes on the Sun Certified Network Admin exam.

It's been long, too long, since I started studying for my SCNA exam. I have to admit that I didn't follow my own tips (refer to my website), which led to me taking way too much time, while I could've been certified a long time ago. Which is why I've decided to make this writeup. It contains a summary of the "SCNA study guide" by Rick Bushnell (see my website for a detailed review). Of course anyone is free to use my writeup to prepare for his/her exam, but I have to warn you: in no way do I guarantee that I've covered all of the required details. I sincerely recommend that you get the book itself as well, since you'll have to read it at least twice to be properly prepared.

Chapter 1: Layered network models

OSI/ISO model was developped in 1983 and applies to any type of network.

TCP model was developped in 1962 and only applies to TCP networks, such as the Internet and ARPA net.

Examples of things that fit into the TCP network model layers:
5. HTTP, FTP, SMTP, SSH and so on.
4. TCP, UDP.
3. IP, ICMP, ARP, RARP.
2. Ethernet, Token Ring, FDDI, ATM.
1. Coax, fiber, UTP.

	OSI / ISO		TCP
7	Application	5	Application
6	Presentation		
5	Session		
4	Transport	4	Transport
3	Network	3	Internet
2	Datalink	2	Network
1	Physical	1	Physical

Names for data in a number of protocols:
Applications use messages and streams. TCP makes use of segments, while UDP uses datagrams. IP sends its data in packets or datagrams. On the data link layer data gets transmitted in frames and finally, on the physical level, things are handled as signals.

UDP checksumming can be enabled by using **`ndd /dev/udp set udp_do_checksum 1`**.

RFC numbers for applications and protocols are listed in the book on pages 18 and 19.

Chapter 2: Introduction to local area networks

The odd term "collapsed bus" applies to devices like switches and hubs.

There are three basic topologies for a network: bus, star and ring. Ethernet networks have a bus topology but are usually laid out in either a bus or a star geography. Token ring networks have a ring topology, but are usually laid out in a ring or star geography.

Applying traditional terms to modern hardware would make a hub a "multiport repeater" and a switch a "multiport bridge".

Repeaters and hubs function on layer 1 of the OSI model.
Bridges and switches function on layer 2.
IP switches and routers all function on layer 3.
Finally, gateways live at layer 4 of the model.

In an Ethernet the maximum round trip of one network segment is the same time it would take to transmit 576 bits (which would be 5120 nanoseconds, a number chosen as standard). On a 10Mbit network this makes the maximum segment length 2500 meters, whereas it's 250 meters for a 100 Mbit network.
Protocols of interest which aren't discussed in the book:

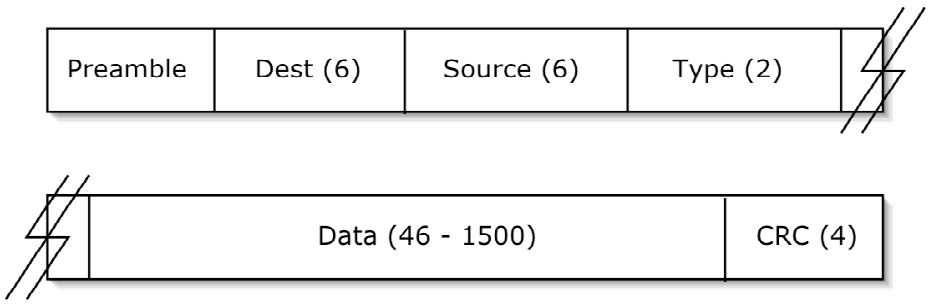
X.400	Mailing
X.500	Directory service
IPX/SPX	Novell
NetBIOS	IBM -> Windows
NetBEUI	Exends NetBIOS. Requires TCP/IP to work on Internet level.

Chapter 3: Ethernet LANs

Ethernet's main characteristics are as follows:
* Can use both copper and fiber glas interconnects.
* Functions at speeds of 10 Mbit, 100 Mbit (called Fast Ethernet) and 1000 Mbit (called Gigabit Ethernet), all of which can be combined in one segment.
* Supports multiple LAN technologies.
* Has a geography which is easy to alter.
* It's low cost in hardware.
* Has a capacity for high transmission speeds and high bandwidths.
* Is a vendor independant standard.
* Easy to install and upgrade.
* Developped at Xerox PARC (Like so many other great things. Refer among other books, to "Apple Confidential 2.0" by Owen Linzmayer), with DEC, Intel and Xerox working together on version 2 of the standard.

Nodes on an Ethernet network are addressed by their "hardware address" or "MAC address" (MAC = Media Access Control). Sun Microsystems has three MAC address ranges in use:
* Enterprise 10k systems use 00:00:be:XX:YY:ZZ
* Blade systems use 00:03:be:XX:YY:ZZ.
* All other Sun systems use 08:00:2D:XX:YY:ZZ.

The first three octets of a MAC address are called the CID (Company ID), with the other three being the VID (Vendor ID).



Above you see the layout of an Ethernet frame. Each field has both its name and its size (in bytes) printed in each label. As you can see the overhead in an Ethernet frame consists of 14 bytes in the header and an extra 4 bytes in the form of the Cyclic Redundancy Check. Hence, the minimum frame size is 64 B and the maximum size would be 1518 bytes.

These frame sizes are requirements based upon the length of the Ethernet interconnects (else the CSMA/CD part of Ethernet wouldn't function properly) and the need to be able to check for truncation.

The preamble of an Ethernet frame consists of a series of high-low transitions on the line, which is used to claim access to the Ethernet bus. The preamble allows a node to transmit on the line without causing a collision in the middle of sending a frame. Unfortunately I have forgotten the amount of high-low transitions which make up the preamble.

The type field is used to show which kind of data is encapsulated within the Ethernet frame. Possible values are:

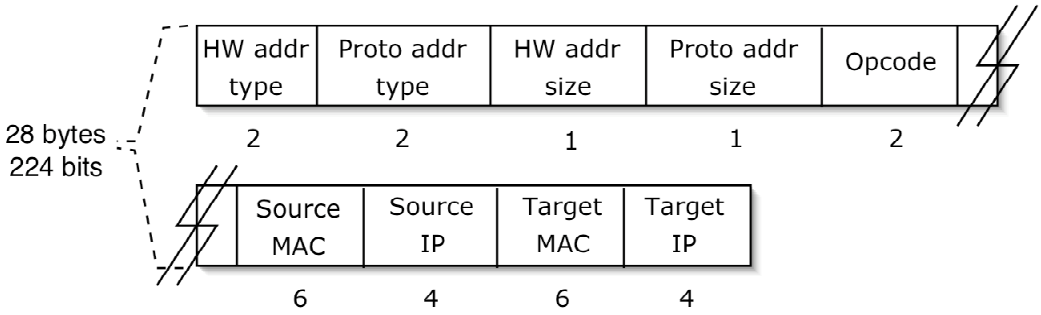
0800	IP	8035	RARP
0806	ARP		

The collision rate of a node can be calculated by dividing the number of collisions caused by the node by the number of frames sent by the node. This number should then be multiplied by 100%, giving you a percentage between 0 and 100. Anything above 5% is bad and should be cause for an investigation.

Maximum throughput on an Ethernet LAN can be gained at around 60% of the available bandwidth (meaning approximately 6 Mbit, 60 Mbit and 600 Mbit). Attempting to send more data across a segment (by more than one host) will cause an increase of the number of collisions (and thus a lower throughput).

Various **`ndd`** parameters can be found on pages 59 through 64 of the book. These pages list all tunable parameters for devices **`/dev/ip`**, **`/dev/tcp`**, **`/dev/udp`** and **`/dev/icmp`**. These lists can also be acquired in Solaris, by running **`ndd $device \?`**.

Chapter 4: ARP and RARP



There are four valid OP codes available in the ARP/RARP scheme:

1 ARP request
2 ARP reply
3 RARP request
4 RARP reply

In ARP the "target" system, is the node which the source host would like to communicate with. Hence, the ARP/RARP replies are sent by either the target node itself, or by a third party who publishes ARP entries.

In RARP a host wants to find out its own IP address. The host uses its own MAC address to fill in both the source and the target MAC addresses. A RARP reply gets sent by an **in.rarpd** server, which enters the requested IP address as the source IP.

in.rarpd -a	Start the RARP daemon
arp $host	Find the MAC address for $host
arp -a	Show all cache contents
arp -d	Delete an entry from the cache
arp -s	Add static entry to ARP cache
arp -f	Populate ARP cache from a file

Bot the -s and the -f flags can be modified with the additional options **perm** and **pub**. **pub** signifies that the host should answer any ARP requests involving this particular static entry, whereas the **perm** option signifies that the ARP cache should not remove the entry automatically (which happens normally after 20 minutes).

Chapter 5: Internet layer and IP

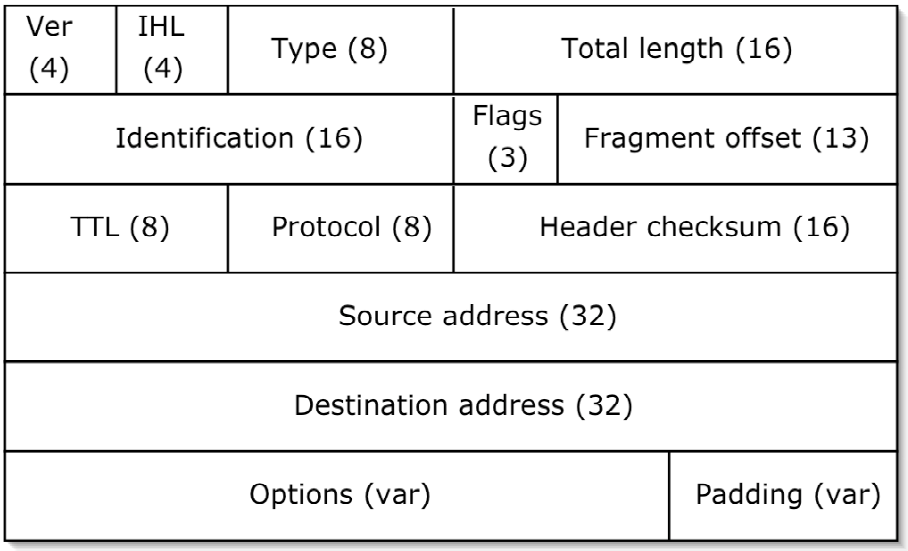
In IP version 4 there are four address classes:

Class	Starts with	Starts with	Network	Max nodes
A	0	0-127	1st octet	~16e6
B	10	128-191	1st two octets	64k
C	110	192-223	1st three octets	254
D	1110	224-239	Multicast	(2^29)-1

The netmasks for network classes A through C are obvious. The netmask for a class D network however, is 240.0.0.0.

Netmasking is byte bound, whereas subnetmasking can be. Subnetmasking can make use of single bits, eg a netmask of 255.255.255.224 -> 11111111.11111111.11111111.11100000. This subnetmask would make all subnets start at a multiples of 32 -> .0 .32 .64 .96 .128 .160 .192 .224.

The IP header usually consists of a maximum of 6 lines, each 32 bits (4 bytes) in length. Hence the headers size is 192 bits. However, the minimum length is 160 bits and the real maximum length is 480 bits.



IHL Header Length in bytes. Ranges from 20 to 60 (160 - 480 bits), depending on the options field.
Type Service level
ID Unique value for reassembly of fragments
Flags Fragmentation control
Proto Encapsulated protocol (refer to **`/etc/inet/protocols`** for all values)
Examples are 0 (or is it 2? -> ICMP), 6 (TCP) and 17 (UDP).

An IP datagram's maximum size is 64 kB, which leads to various grades of fragmentation, depending on the underlying network infrastructure (and the types of networks along the path between the source and destination nodes). Keep in mind: not only hosts, but also routers and the likes fragment incoming and outgoing data.

CIDR = Classless Internet Domain Routing. A mechanism which allows routing based on geographical locations for instance. It bases routing decissions on the N high order bits in an address. All IP addresses with high order bits X get routed to the main group of routers for their particular group or location. Further routing is performed within said group.

VLSM = Variable Length Subnet Masking. Allows the nesting of multiple subnetmasks and is not support per default by Solaris 8. VLSM requires the additional **gated** software, which is available for download freely.

In Solaris the default maximum amount of virtual interfaces per physical interace is 256. This value can be modified using the **ndd** command. Creating a virtual interface is done as follows:

```
ifconfig $dev:$num plumb
ifconfig $dev:$num $ip netmask $mask -trailer up
```

The above gives you a virtual network interface while the system remains up and running. All of this can be made permanent by putting the virtual interface's hostname in **/etc/hostname.$dev:$num.**

Startup scripts involved with getting the network up and running in a Solaris box are:

```
/etc/rcS.d/S30network
/etc/rc2.d/S69inetinit
/etc/rc2.d/S72inetsvc
```

Chapter 6: Routing with TCP/IP

Basic routing decissions from source host.
Is $dest on my subnet?
    If so -> destination IP is $dest_ip and destination MAC is $dest_mac.
    If not -> destination IP is $dest_ip and destination MAC is $router_mac.

Routing decissions on the receiving host.
Is $dest_mac me?
    If so -> Is $dest_ip me?
        If so -> Pass up stack.
        If not -> Am I a router?
            If so -> Pass on to next hop.
            If not -> Ignore.
    If not -> Ignore.

There are three types of routes: to a host, to a network and default routes.

Static routes can be set in a multitude of ways:

`/etc/defaultrouter`	Configures the default route for traffic outside this subnet. If this file is present, RDP and RIP are NOT started at boot time.
`/etc/gateways`	Configures a list of gateways to use for foreign subnets.
`route` (command)	Used to add new entries to the routing table.

Dynamic routes can be set using the following methods:

RIP -> **`in.routed`**	-q = client, -s = server
RDP -> **`in.rdisc`**	-s = client, -r = server
ICMP redirects	

RIP = Routing Information Protocol. RIP uses broadcast addresses and port 520 to send updates approximately every 30 seconds.
* Support for VLSM and subnetting in RIP version 2.
* Based on UDP.
* Suitable for internal use (>15 hops is unreachable).
* Example: If router A is separated from router B by ten network segments an update on A will take about five minutes to propagate to router B.
* The "split horizon" feature ensures that a router does not send routes to other routers who may already know about them.
* "convergency" = a state in which all routers agree which routes are good and which ones are bad / dead.

RDP = Router Discovery Protocol. Sends updates to "default routers" through the multicast address 224.0.0.0.
* Solaris 8 allows for multiple "default routers" per host to perform load balancing.
* Address 224.0.0.1 is the multicast address for all nodes on a segment.
* Address 224.0.0.2 is the multicast address for all routers on a segment.
* RDP is based on ICMP.
* RDP does not advertise routes to a whole network as RIP does, but only to default routers. Hosts will only have to add these default routers to their local table.

A number of multicast addresses are reserved by the IANA for default host groups. The IANA owns the block of addresses starting with 01:00:5e, so the range is 01:00:5e:00:00:00 through 01:00:53:7f:ff:ff. Calculating a multicast MAC address from a multicast IP address can be done as follows: take the last three octets of the IP and convert them into HEX. Add these three numbers to the MAC address starting with 01:00:5e.

Multicast IP addresses always start with 1110xxxx.

The **/etc/gateways** file uses the same syntaxt as the **route add** command:
[host|net] $address gateway $address_gw metric $value [passive|active]
Active type routes are removed from the routing table if not refreshed by RIP every 180 seconds. Net 0.0.0.0 signifies a default router.

If a Solaris system contains more than one network interface it is autmatically configured as a routing system. This can be corrected by touching **/etc/notrouter** and by running **ndd -set /dev/ip ip_forwarding 0**.

route monitor	Gives live output of all routing table changes.
route get	Shows routing table, line by line.
netstat -r	Shows full routing table.

Routing table flags:

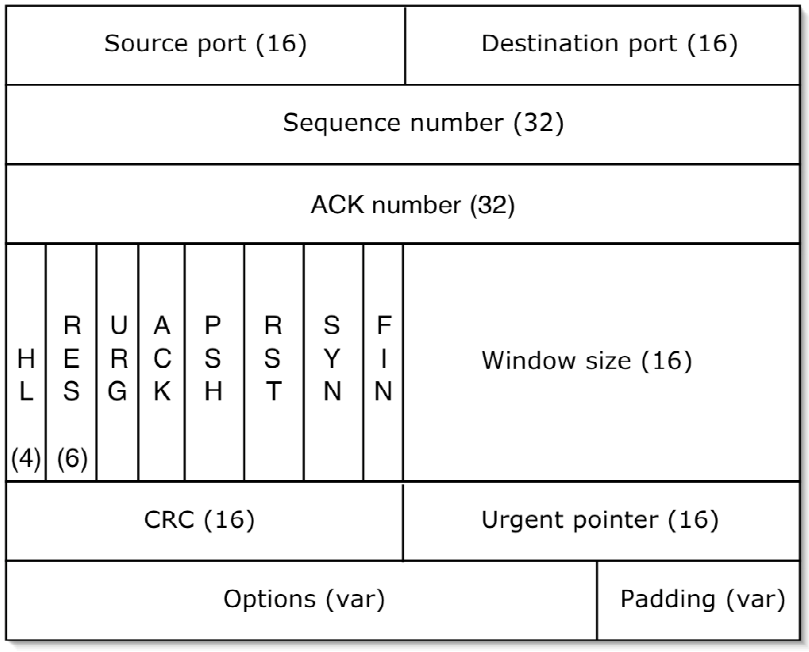
H	Host specific.
U	Up.
G	Reachable through this gateway.
D	Route added through ICMP.
M	Route modified through ICMP.

in.routed has the following options:

[-s | -q]	Server/router or Quiet/client.
-v	Verbose.
-t	Write to stdout.

Chapter 7: Transport layer protocols

TCP header and protocol.



SECQ Sequence number, or the number of the first byte in the segment, excluding the header.
ACK Number of last received byte, plus 1. Thus it is the number of the next expected byte. It is also ignored if the ACK flag is not set.
HL Header length in 32 bit words (default is 5, thus excluding the options and padding).
URG Urgent, signifying that TCP needs to transmit the data immediately. Requires the SECQ field to be set to the SECQ of the byte after the urgent data.
ACK Signifies that segment is used to ACK a previously received segment.
PSH Signifies that every character typed must be pushed to the application immediately. Rlogin and telnet use this option.
RST Reset the connection.
SYN Only used during the threeway handshake, during the setup of a connection.
FIN Signals a ready to close the connection.
WIND Number of unACKed bytes that the node is willing to accept, starting after byte $ACK.
OPT Used during threeway handshake. Contains settings like the MSS.

For the calculation of the CRC (or checksum) a pseudo-header is prepended to the real header. Then the header is padded with zeroes until it reaches a multiple of 16 bits. The CRC is then calculated. The pseudo header and the padding are not part of the data which gets transmitted. The receiving end re-calculates the checksum after rebuilding the pseudo-header on its side.

Most fields of the pseudo-header are obvious (refer to the graphic on the next page). The protocol field either gets set to 6 (TCP) or 17 (UDP) and the TCP length is the length of the entire segment excluding the pseudo header.

Source IP (32)		
Destination IP (32)		
Padding (8)	Protocol (8)	TCP length (16)

The four most important features of the TCP protocol are:
* Three way handshake.
  Agree on connection, exchange initial sequence number (ISN), agree on MSS, ACK each other's ISN's -> SYN, SynACK, ACK.

* Maximum Segment Size (MSS) negotiation.
  Based on MTU of system proposing the MSS. Aim is to never let any segments be fragemented along the way.

* Positive acknowledgement with retransmission.
  Retransmission occurs when the receiving end fails to ACK a segment within a set timeout.

* Sliding window.
  Total window size equals the amount of bytes sent but not ACKed, plus the bytes ready to send. All bytes ready to send, but outside the window need to wait for the ACK of previous bytes.

UDP header and protocol.

Source port (16)	Destination port (16)
UDP length (16)	CRC (16)

The UDP checksum is not enabled by default. You can turn this on with **`ndd -set /dev/udp udp_do_checksum 1`** This checksum uses a pseudo-header, which gets calculated in the same way as the TCP pseudo-header.

UDP is:
* High speed.
* No guarantees of delivery.
* Connection-less.
* Low overhead.

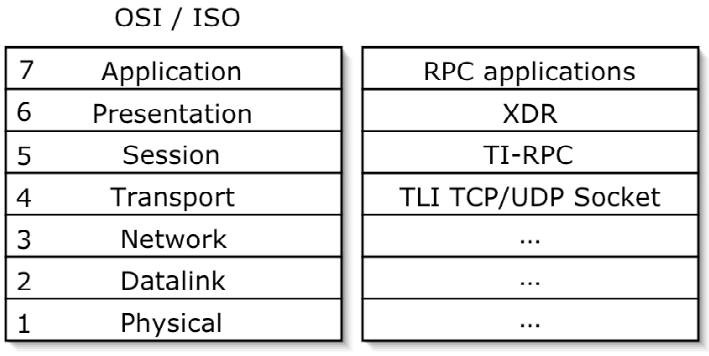
UDP does not provide any measures for reliability. If an application or protocol requires such measures, but would like to use UDP, they are expected to deliver their own.

Chapter 8: Client-server model

ONC+ Open Network Computing +

RPC Remote Procedure Call

XDR External Data Representation. Allows RPC applications to exchange data cross-platform and cross-network.



TLI Transport Layer Interface. Allows applications to use TCP or UDP without the need to have this hard coded into the application.

Sockets BSD-style method for Inter Process Communications. Implemented in Solaris according to the XNS-5 standard.

TI-RPC Transport Independant RPC.

There are two kinds of server processes:
* Stand-alone.
* Started through inet daemon.

The inet daemon requires:
/etc/inetd.conf
/etc/services
rpcbind process

Socket based services are configured in **/etc/inetd.conf** in the following way:

```
name    socket-type  proto  flags  user  path              arguments
eg: ftp stream       tcp6   nowait root /usr/sbin/in.ftpd  in.ftpd
```

Type Either stream (TCP), dgram (UDP) or tli (TLI).
Proto Either tcp, tcp6, udp or udp6. (IPv6 enabled services are ftp, telnet, shell, login, exec, tftp, finger and printer)
Flags Either wait or nowait.
Path "internal" means that the service is built into the inet daemon itself.

TI-RPC based services are configured in **/etc/inetd.conf** in the following way:

```
name/version  endpoint-type  rpc/proto  flags  user  path  arguments
eg: rexd/1    tli            rpc/tcp    wait   root  /usr/sbin/rpc.rexd
                                                        rpc.rexd
```

Port allocation on the client side is arbitrary. On the server side however ports get assigned according to the **services** file or through the **rpcbind** process. The **services** file can assign ports 0-65535, where ports 0-511 are IANA official ports and ports 0-1024 are started with root privileges.

Rpcbind runs on port 111 and is also knows as the "portmapper". It listens to both TCP and UDP traffic. Using the PMAPPROC_SET instruction the portmapper assigns a port to a service (which can be a different port each time). Requests from clients use the PMAPPROC_GETPORT instruction.

RPC program numbers are listed in **/etc/rpc**. These numbers are registered with the IANA and are what is used to identify an RPC service (instead of a port number). A client sends a request to a server, requesting the current port for the service with a certain program number. The contents of **/etc/rpc** are divided into the following fields: **service program alias1 alias2**

The rpcinfo command returns a full list of registered programs when used with the -p flag. The displayed list is divided into the following fields:
program version protocol port service

The **rpcbind** process is listed as program number 100000. De-registering a service is also performed with the **rpcinfo** command, but this time with the -d flag.

The **netstat** command has a number of interesting flags:

-a	Active sockets.
-P	Protocol [tcp | udp].
-p	Display the ARP cache.
-n	Show IP and port numbers, instead of resolved names.
-s	Protocol statistics.
-r	IP routing table.
-m	Kernel buffers.
-i	Interface parameters.

The **netstat** command can display a number of columns which are not always obvious in their meaning:

swind	Send window size.
send-Q	Real-time bytes in send queue.
rwind	Receiving window size.
recv-Q	Real-time bytes in receiving queue.
state	Either bound, closed, closing, establised, listen, close_wait, find_wait_1, find_wait_2, idle, last_ack, syn_received, syn_sent or time_wait.

In **netstat** an "unbound" port is free for use and an "idle" port is in use, but not currently servicing any requests. UDP ports never show a remote address in **netstat**, since UDP is connection-less.

Chapter 9: Dynamic addresses and DHCP

DHCP clients (can) receive almost any network information from their server. This may be an IP address, a netmask, DNS settings, NTP settings, routers, etc. In order for a DHCP client to bind to its server across subnets a DHCP/BOOTP relay is required in the client's own subnet. One turns on DHCP by using the following command:

`ifconfig $device dhcp`.

`ifconfig $device dhcp status` show the current state of your interface's DHCP state.

The idea of IP leases are what set DHCP apart from the BOOTP protocol. A server leases a binding to a client for a specified period of time, which may be renewed to allow the binding to exist for a longer period of time.

Benefits of using DHCP are:
+ Automatic (de)allocation of IP addresses.
+ Easy reuse of IP's.
+ Provides other network settings as well.
+ IP sharing by nodes who connect to the network randomly (laptops).
+ Node relocation is made very easy.
+ Possible across multiple subnets.

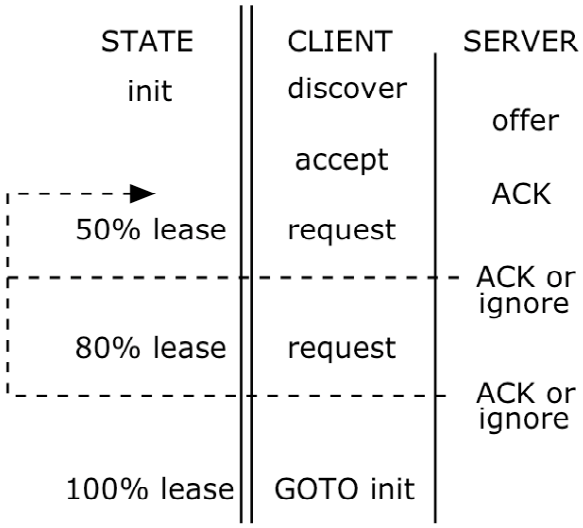
Disadvantages of using DHCP are:
- Intruders get easy access to your network information.
- Requires additional training for your system administrators.
- Requires additional monitoring.

In DHCP the following terms are defined as follows:

Permanent IP	Given for an infinite amount of time and is not subject to lease.
Leased IP	Given for a finite amount of time, measured in seconds since Epoch.
Specific IP	Predetermined IP which is reserved for one client, aka"static" or "fixed".
Manual allocation	Permanent and specific.
Automatic alloc.	Permanent and non-specific.
Dynamic alloc.	Leased and non-specific.

Reasons to use specified addresses:
* Monitoring and logging.
* Restricting the use of services.
* Access and NATing through firewalls.

DHCP states are traversed as follows:



DHCP configuration files

DHCP uses the following configuration files, which will be discussed in order:
/etc/dhcp/inittab
/var/dhcp/$network_ip
/var/dhcp/dhcptab
/etc/dhcp.$interface
/etc/default/dhcp
/etc/default/dhcpagent

/etc/dhcp/inittab
Exists on both the client and the server. Has five option categories, each with its own scope: Standard (IANA), Site, Vendor, Field and Internal. Most options describe information passed on from the server to the client system. Most lines even have the same fields, regardless of the category. This file should not be edited, except to add specific Site or Vendor lines.

Field 1 Identifier, a user friendly mnemonic for the option number (field 3).
Field 2 Type or category.
- STANDARD — Defined by IANA. Applies to all DHCP clients and should not be modified.
- SITE — Empty by default. Allows customization per location or site. New entries are always "sdmi".
- VENDOR — Up to 254 items that are specific to one vendor. "smi" is Sun Microsystems Inc.
- FIELD — Allows aliasing of DHCP packet fixed fields to mnemonics (see the **dhcpinfo** command).
- INTERNAL — Should not be modified. Part of Sun's specific implementation of DHCP.

Field 3 Option number. Possible values are 1-127 for STANDARD, 128-254 for SITE and 1-254 for VENDOR.
Field 4 Data format, specifies what kind of data is passed. A full list of formats is: Ascii, octet, Unumber8, Snumber8, Unumber16, Snumber16, Unumber32, Snumber32, Unumber64, Snumber64, IP.

<continued on next page>

Field 5 Granularity, indicates how many items of type $Field-4 make up the whole value.
Field 6 Count allowed, indicates how many units of type $Field-4 may be passed from Client to Server in one packet. Setting this value to zero means there is no maximum.
Field 7 Visibility. Which programs may use this information. Values are: smdi, smi, id, dm (see the **dhcpinfo** command).

/var/dhcp/$network-ip
Created by the **dhcpconfig** command and could, for example, be called **/var/dhcp/192_168_0_0**. The contents of this file may also be stored in NIS+.

This file defines a pool of IP addresses, the server that hands them out and the macro from **/var/dhcp/dhcptab** that defines the applicable binding.

Field 1 Client identifier. Usual string is "01$MAC", where $MAC is the MAC address of the DHCP client system, without delimiters.
Field 2 Flags. Possible values are:
- 00 Dynamic. Negotiation requires the **LeaseNeg** macro to be set in **dhcptab**.
- 01 Permanent.
- 02 Manual.
- 04 Unusable (already in use).
- 08 BOOTP (reserved for BOOTP clients).

Values may be combined in such a way that 01+02=03.
Field 3 Client IP address **$node-ip**.
Field 4 IP address of DHCP server who 'owns' the client IP -> **$server-ip**.
Field 5 Lease period in seconds.
Field 6 Applicable **dhcptab** macro.

All entries for this file should be created using the **pntadm** command. This ensures that all entries adhere to the proper format and syntax. My guess is that if you leave the Client_ID field blank the **pntadm** command will ensure that the IP address and other data will be assigned dynamically to whichever client comes along.

/var/dhcp/dhcptab
This file defines DHCP macros and the symbols used in them. These macros are then used in the file **/var/dhcp/$network-ip** to configure DHCP clients. A macros is a grouping of options and the values that you wish to assign for these options to a certain client system or group. A number of basic/default macros get created by the **dhcpconfig** command when adding a new network to the DHCP server.

Field 1 Name.
Field 2 Type, either "symbol" or "macro".
Field 3 Colon delimited option-value pairs.

Macros consist of a number of option-value pairs, divided by colons. Lease negotiation uses two symbols in the relevant macros: LEASETIM (lease period in seconds) and LEASENEG (boolean, defines if negotiation is allowed).

If DHCP fails while booting the client will use **/etc/inet/hosts** to get its IP address. If that file does not exist the client will keep on trying to use DHCP. Examples are provided on the next pages.

Macros with certain names will be processed automatically for each DHCP client. These names are:
* $Client_class, for instance SUNW,Ultra-1.
* $Network_addr, for instance 194.168.85.0.
* $Client_ID, for instance 08002011DF32.

Macros with names other than those in the list above will need to be mapped to a specific client (or group thereof) through the $network_ip file, or by inclusion in one of the automatic macros.

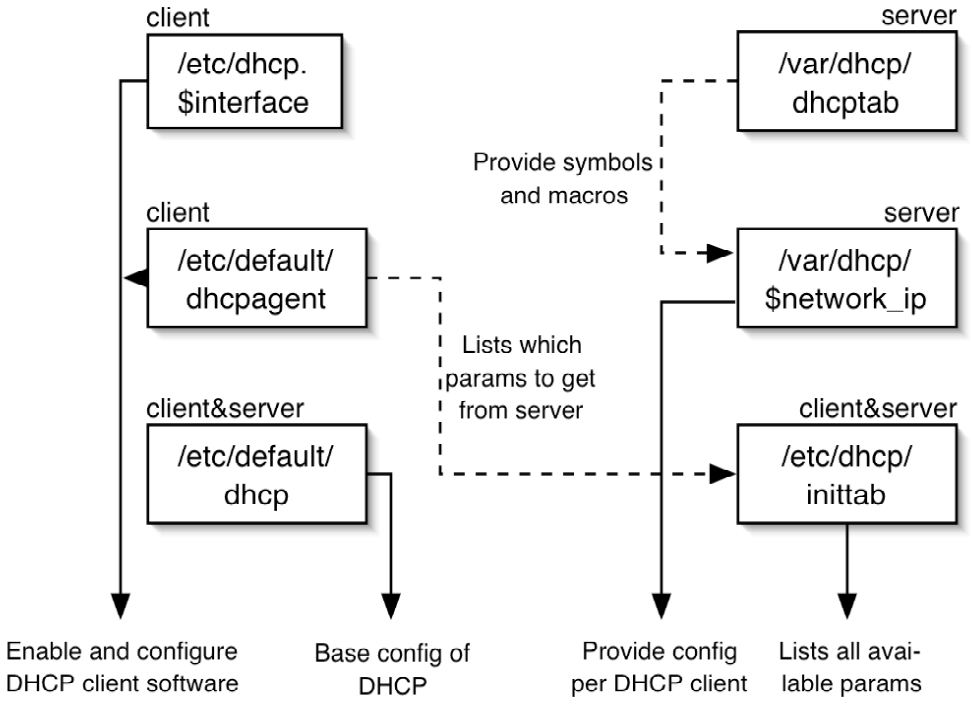
One DHCP rule says that option-value pairs from a more specific macro will overrule a pair from a more generic macro. For instance, an option-value pair from a macro named $Client_ID will overrule the value for that same option from a macro mapped to an IP address or from a macro named $Network_addr.

All entries for this file should definately be created using the **`dhtadm`** command.

`/etc/dhcp.$interface`
Causes the system to bypass **`/etc/hostname.$interface`**.

`/etc/default/dhcp`
Contains settings regarding the DHCP host type and various resources. Example entries are:
```
RUN_MODE=server
RESOURCE=file
PATH=/var/dhcp
```

`/etc/default/dhcpagent`
Handles tunable parameters for DHCP and ifconfig, for example:
`PARAM_REQUEST_LIST=1,3,12,43` (resp. netmask, router, hostname and vendor options), or **`ARP_WAIT=1000`** (the amount of miliseconds to wait while checking if an IP is already in use).



Examples of the interconnection between DHCP configuration files.

/etc/default/dhcp indicates that this host is a server and that its configuration is stored in files in **/var/dhcp**.

/var/dhcp/194_168_85_0 indicates the following configuration for two of the hosts in this network:

```
010800207b33a4 00 194.168.85.180 194.168.85.51 1009397309 gvonnet1
0100008627a032 03 194.168.85.181 194.168.85.51 -1         gvonnet1
```

The first host would be a dynamic DHCP client with a finites lease period. The second host is of the type Permanent-Manual and has no lease period. Both hosts can be further defined using the macro **gvonnet1** from **/var/dhcp/dhcptab**.
/var/dhcp/dhcptab in this case lists the macro **gvonnet1** as follows:

```
gvonnet1 m :GvonstatRt=10.1.1.0 194.168.85.62:Include=Locale: :Timeserv=
194.168.85.51:Leasetim=259200:DNSdmain=gvon.com:DNSserv=194.168.85.1
194.168.85.2:LeaseNeg:
```

In this macro definition **GvonstatRt** is a symbol, which is also defined in **/var/dhcp/dhcptab**, as follows: **GvonstatRt s SITE,130,IP,2,0**. This definition translates into the following:

Context	GvonstatRt
Type	Symbol
Value	Scope=SITE, Code=130, Type=IP, Granularity=2, Maximum=0.

Now, say for instance that the **dhcpagent** file on a client system tells it:
PARAM_REQUEST_LIST=1,3,12,43.

This would mean that the client will attempt to get the listed parameters from the DHCP server, which defines these params in the **/etc/dhcp/inittab** file. For example:

```
Subnet   STANDARD,      1,      IP,     1,      1,      sdmi
UTCoffstSTANDARD,      2,      SNUMBER32, 1,   1,      sdmi
Router   STANDARD,      3,      IP,     1,      1,      sdmi
```

DHCP commands

dhcpconfig
Is a command line based wizard which performs an automatic setup of your DHCP server. It asks a number of questions regarding options and which IP ranges to serve. This command is best used when you either want to A create a new DHCP server or B) want to add new networks to your DHCP server. In the case of B the **dhcpconfig** will create the necessary table, as well as all of the default macros.

pntadm
Manages the **/var/dhcp/$network-ip** files, and their macros and symbols. This command has a number of options, which cannot be combined.

-C Create main table file. Param = $network-ip.
-B Batch mode. Read multiple configurations from a file.
-A Add entries. Params -> see below.
-M Modify entries. Params -> see below.

-D Delete entries. Params = $node-ip and $network-ip.
-P Display contents of file. Param = $network-ip.
-L List all networks that have a table file. Takes no params.
-R Remove main table file. Param = $network-ip.

Two parameters are common to all options: **-r $resource** and **-p $table-file**. These indicate the used resource (either "SUNWfiles" or "SUNWnisplus") and the path within said resource.

Other parameters of note are:

-f	Flags.	-c	Comment.
-e	Expiry / Absolute lease.	-h	Hostname entry for **hosts** file or DNS.
-s	DHCP server.	-i	Client ID (in ASCII if used with -a).

dhtadm

Manages the **/var/dhcp/dhcptab** file and the contained macros and symbols. The -A and -M options take the following parameters:

[-s $symbol-name | -m $macro-name] -d [$symbol-characteristics | $macro-definition] -r $resouce -p $directory

The -M option also accepts the -n parameter, which defines a new name for an existing object.

dhcpmgr

An X11 application to manage DHCP settings. It is not covered by the book.

in.dhcp daemon

Stop **/etc/init.d/dhcp stop**
Start **/etc/init.d/dhcp start**
Normal **/usr/lib/inet/in.dhcpd**
Debug **/usr/lib/inet/in.dhcpd -d -v** (output to stdout)

Chapter 10: Network management with SNMP

Network management as defined by the ISO can be divided into the following subjects:
* Configuration management
* Fault management
* Performance management
* Accounting management
* Security management

SMI = Structure of Management Information. RFC's 1155 and 1156 define the SMI for objects contained in an SNMP Management Information Base (MIB).

OID = Object Identifier. A string of numbers defining the location of an object in the tree. The root itself is unlabeled. For example, 1.3.6.1.4.1.42.3.1.2 is the path to **`iso.org.dod.internet.private.enterprise.sun.sunMib.SunSystem.hostid`**.

MIB = Management Information Base. RFC 1156 describes MIB objects using Abstract Syntax Notation One (ASN-1). These objects include:
* System
* Interface
* Address translation
* UDP
* IP
* ICMP
* TCP
* EGP

Properties of an object:
* Name
* Syntax
* Definition
* Access
* Status

SNMP functions at layer 7 of the OSI model and uses UDP to traverse the network.

Chapter 11: Domain name service

BIND = Berkeley Internet Domain Naming, which exists in three current versions: 4(.1.9), 8(.1.12) and 9(.2). Version 9 is rare, while versions 4 (Solaris 6) and 8 (Solaris 8) are encountered in the wild. The difference between these versions lies mostly in performance and security.

The main configuration file for BIND4 is **`/etc/named.boot`**.
The main configuration file for BIND8 is **`/etc/namded.conf`**.
The BIND daemon is **`/usr/sbin/in.named`**.

FQDN = Fully Qualified Domain Name, eg "lab.sun.com". Max length = 255 characters.

RDN = Relative Domain Name, eg "lab". Max length = 63 characters.

TLD = Top Level Domain. TLD's are managed by ICANN (Internet Corporation for Assigning of Names and Numbers), which delegates management to local authorities.

Forming a new Zone Of Authority requires your own SOA (Start of Authority) and NS records, as well as a master server.

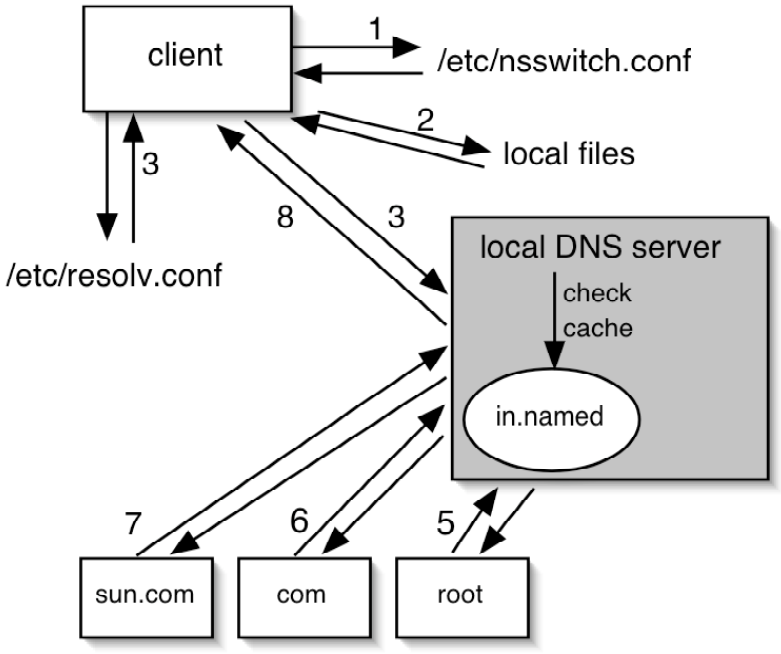
Root Level DNS servers are managed by ICANN (and Internic and IANA and so on), since they are in charge of the TLD's. Currently there are thirteen separate Root Level DNS servers which are accessible to anyone. A full list of servers can be downloaded from **`ftp.rs.interic.net://domain`** . These servers are named **`[a-m].root-servers.net`**. The details for these servers should be entered into your local DNS servers in the file indicated in the **`zone "."`** in **`/etc/named.conf`** . See page 21.

Master servers, which are also known as "primary servers" are in charge of their own Zone of Authority. A master server may replicate its database to a number of slaves and may delegate management of subdomains to lower level master servers. Your main master server must be registered at the same time you register your Internet domain. It should also be reachable for DNS traffic from the Internet.

Slave servers, also know as "secondary servers" are used for load balancing and redundancy within your local domain. The synchronisation process between the master and slave servers is known as a "zone transfer". Slaves are usually also registered with your Internet domain and hence should also be reachable from the outside. One slave server is considered to be the absolute minimum for a domain.

Caching-only servers are not authorative and should only be accessible to the inside of your domain. They are used for load balancing purposes only and they contain very little local DNS information. They get most of their data from the thirteen root server. DNS entries are kept in memory only and will disappear after a reboot.

DNS Forwarding servers are servers (either master, slave or caching) which forward all of their requests to a specific and predetermined external DNS server. These kind of DNS servers are used for security purposes. The difference with a normal DNS server is the fact that normal servers will send out queries to other DNS servers according to the domain names they are looking up.



Traffic between the client and its local DNS server is "recursive", insofar that the client will keep on waiting for an answer. In the meanwhile the DNS server may query other DNS servers for the real answer. In technical terms the DNS client is also called the "resolver".

Traffic between the local DNS server and the external DNS's is "cyclic" in nature and thus "iterative". When a local DNS server does not have the answer to a query it was posed it will start with the root level DNS server and from there drill down through the domain structure until it has found what it was looking for.

Every DNS server that learns new information will store it in its cache for the duration of the TTL (Time To Live).

/etc/named.conf
Lists the zones which are supported by this DNS server and points to the location of the relevant database(s). MX Records (Mail EXchanger) point to the main e-mail server for each zone.

Every line ends with a semi-colon and multi-line sections are delimited by accolades. Six main sections are supported:

* Options	* ACL
* Server	* Include
* Zone	* Key (not used)

The **options** section sets default parameters for all zones in the file. These settings may be changed individualy per zone. Only one options section exists within this file and it should be placed at the head of the file. A number of the many options are:

directory	Location of resource files.
allow-transfer	List of slave servers, enclosed in accolades with a semi-colon after each IP address.
allow-update	Shows who is allowed to perform dynamic updates.
forwarders	List of servers to query if you don't know the answer yourself. If these servers don't know the normal iterative process takes place.
forward only	Strictly limit your queries to the list of servers in forwarders.

The **server** section is completely optional and may be used to disable the querying of certain DNS servers using the **bogus yes** directive.

Each supported zone requires its own **zone** section in this file. There are four types of entries in this section:

hint Points to a local file with thirteen root level servers. These entries will be added to the cache immediately.

```
zone "." in {
        type hint;
        file "named.root";
};
```

master Forward zone -> Allows a DNS server to map a name to an IP address.

```
zone "gvon.com" in {
        type master;
        file "g/db.gvon.com";
};
```

master Reverse zone -> Mapping of an IP address to a hostname.

```
zone "85.168.194.in-addr.arpa" in {
        type master;
        file "networks/db.194.168.85";
};
```

slave Shows which master server to look at and where to store the information.

```
zone "domainbank.co.uk" in {
        type slave;
        masters { 194.168.85.105; };
        file "secondary/d/db.domainbank.co.uk.";
};
```

stub Delegates authority for a zone away to another master.

```
zone "support.gvon.com" in {
        type stub;
        masters { 194.168.85.2; };
        file "stubs/db.support.gvon.com";
};
```

The **ACL** section defines a range of IP addresses or an IP address which gets bound to an alias. This alias may be used in the options section for things like **allow-transer**. For instance: **acl "internal" { 94.168.85/24; };**.

The **include** section can be compared to C and C++ includes :) It allows you to read in other files into **named.conf**.

DNS database and resource records

As described earlier you are free in your choice of file names for the zone forward and reverse lookup databases. However, it would be wise to let the filename contain the zone's forward or reverse name, so it can be easily identified.

In a database the @-symbol is used as an alias for the zone name quoted in named.conf. When optional fields are omitted BIND assumes the previous value of these parameters for the current value.

The syntax of a resource record is as follows:

```
name    ttl     class   type    data
```

name Optional. Contains either a host or a zone name. If the name field is not qualified the current origin (@) is appended by BIND. Examples:

```
gvon.com.       IN      NS      auriga.gvon.com
                IN      NS      centauri.gvon.com
```

ttl Optional. Defines how long a slave caches this particular entry. If omitted the default value from the SOA record is used instead.

class Optional. Always "IN", which is short for Internet, which is also the default value.

type Required. Can have multiple values:

- SOA Start of Authority. Lists the master server, hostmaster e-mail address, slave synchronisation information and default TTL.
- A Address record, an IP address.
- AAAA Address record, an IPv6 address.
- NS Name Server (DNS) record, a hostname.
- MX Mail Exchange name.
- PTR Pointer Record. Required for reverse lookups.
- CNAME Host alias.
- HINFO Host information. Things like operating system, CPU and so on.
- TXT Text field, which is not used often.

data Required. Contents vary with each entry type.

SOA

```
$name IN SOA $master $hostmaster {
$serial ;           Version number for synchronisation with slaves.
$refresh ;          Check every X seconds for an update on the
                    master.
$retry ;            Retry every X seconds after a failed refresh.
$expire ;           Give up refresh after X seconds.
$default-ttl } ;    Time To Live, in seconds.
```

A

```
$hostname $ttl   IN      A       $host-ip
auriga           IN      A       194.168.85.1
centauri         IN      A       194.168.85.2
```

NS

```
$zone   IN      NS      $hostname
```

MX

```
name    class   type    preference      data
        IN      MX      5               $mailhost1

        IN      MX      10              $mailhost2
```

PTR

```
$host-ip $ttl   IN      PTR     $hostname
1               IN      PTR     auriga.gvon.com.
2               IN      PTR     centauri.gvon.com.
50              IN      PTR     alpha.gvon.com.
```

CNAME

```
$alias $ttl     IN      CNAME   $hostname
www     IN      CNAME   voyager.gvon.com.
smtp    IN      CNAME   ophelia.gvon.com.
```

Delegation of zones can be done with two entries:

```
$zone   IN      NS      $master
$master IN      A       $master-ip
```

Setting up DNS systems

Creating a new DNS server:
1. Create **/etc/named.conf**.
2. Create the directories which are referenced by **named.conf**.
3. Create the zone files referenced by **named.conf**.
4. Modify **/etc/nsswitch.conf**.
5. Start the **in.named** daemon.
6. Create a proper **/etc/resolv.conf**.

Creating a new DNS client:
1. Modify **/etc/nsswitch.conf**.
2. Create a proper **/etc/resolv.conf**.

Sending signals to **in.named**:

INT	Dumps cache into /var/named/named_dump.
HUP	Reload configuration.
USR1	Enable real time debugging.
USR2	Disable debugging.
TERM	Kill process.
KILL	Kill process.

Chapter 12: Network time protocol

"Strata" are the various levels recognised within the NTP hierarchy. Stratum-1 servers have their own atomic clocks and are thus the most accurate servers. Each stratum synchronises the servers in the stratum beneath it. The public should only synchronise against strata 2 through 5.

"Jitter" is defined as the difference in differences encountered when measuring time repeatedly. The most accurate server has the least jitter. "Accuracy" defines how closely one follows a reference clock. "Wander" are frequency variations.

A "drift file" contains the frequency offset from the local clock oscillator. This drift file is usually stored in **/var/ntp/ntp.drift**. It allows Solaris to adjust its clock at boot time.

/etc/inet/ntp.conf is either a copy of **ntp.client** or **ntp.server**. As you can guess this is the main configuration file which allows you to either configure an NTP server or a client. This configuration file is referenced by **/etc/init.d/xntpd**, the start script for the NTP process. Any debugging output gets sent to the **messages** file.

NTP consists of two packages: **SUNWntpr** and **SUNWntpu**.

Setting up an NTP server

1. Choose either to synchronise against a stratum server or to be undisciplined.
2. Copy **ntp.server** to **ntp.conf**.
3. Modify the server line in **ntp.conf**, for example:

```
server 127.127.Xtype.0 prefer
server ntp2.mcc.ac.uk
```

   If necessary, add a fudge line to show that a server is not very reliable:

```
fudge 127.127.1.0 stratum 12
```

   Add a line for the drift file:

```
driftfile /var/ntp/ntp.drift.
```

4. Start the NTP server using the init script.
5. Check your NTP hierarchy using the command **ntpq**.

Xtypes can be found in **/etc/inet/ntp.server**. Type 1 is a local clock, while types 2-27 are various GPS receivers.

Setting up an NTP client

1. Copy **ntp.client** to **ntp.conf**.
3. Modify the **ntp.conf** file, for example:

```
server ntp2.mcc.ac.uk
```

   or,

```
multicastclient 224.0.1.1
```

   or,

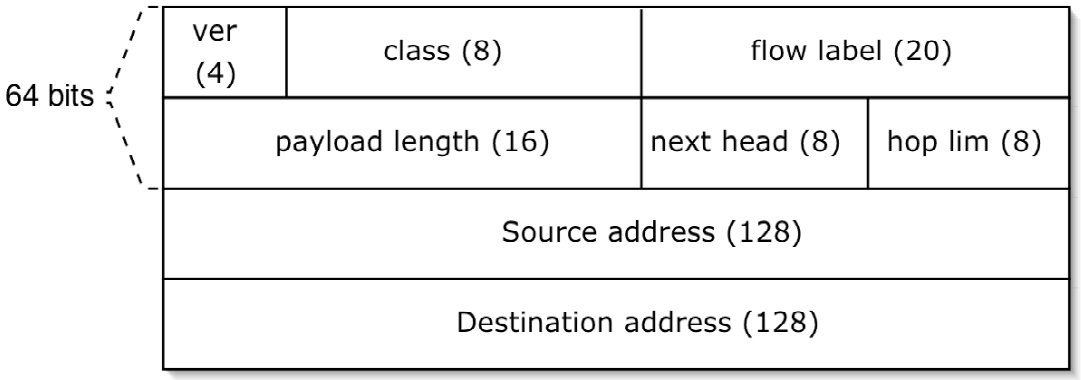
```
broadcastclient 194.168.85.255
driftfile /var/ntp/ntp.drift
```

The **ntptrace** command shows the full synchronisation path used to set the time for your client. The **ntpdate** command can be used to immediately synchronise the time and date on your client.

Chapter 13: IP version 6

IP version 6 was introduced pretty recently, now that IPv4 addresses are nearly depleted. IPv4's biggest failing lies in the assignment of addresses and ranges thereof. If an entity gets assigned an A class address range it has the possibility of assigning 16 million addresses! Nobody needs that many!

The IPv6 header is quite a bit simpler than the IPv4 one. The minimum header size is 40 bytes (or 320 bits) with the source and destination addresses each taking up 16 bytes (or 128 bits).



Version	Currently "6".
Traffic class	Undefined -> all zeroes.
Flow label	Label to identify a stream of data between source and destination. Also used for QoS on routers.
Next header	Compare "protocol" field in IPv4. Defines which protocol in encapsulated in this datagram. As usual 6 is TCP, 17 is UDP and 2 is ICMP. It is also a possibility that the payload consists of another header which serves as the options field. This would be called the "extension header".
Hop limit	Compare to the TTL in IPv4.

One datagram header may contain multiple "extension headers", however the last extension header always designates the encapsulated protocol type. There are six types of extension headers, which provide additional control information:

* Routing
* Authentication
* Hop-by-hop options
* Fragmenting
* Destination options
* Encrypted security payload

IPv6 no longer performs a CRC since both TCP, UDP, ICMP and other protocols already do so. Also, underlying networks like Ethernet and ATM usually have some form of CRC as well.
In IPv6 routing no fragmentation takes place. Through path-MTU discovery a node discovers the lowest MTU along the path to its destination. The sending node may still fragment the data by itself of course, if TCP/UDP/whatever demands it.

While the IPv4 address is made up out of six octets, divided by dots, the IPv6 address consists of eight groups of 16 bits each, delimited by colons. IPv6 addresses are printed in hexadecimal instead of in decimal, to keep things a bit more simple.

Converting a binary IPv6 address into hexadecimal is easy! Divide each group of sixteen bits into four nibbles of four bits each. Then convert each nibble into hexadecimal code. This is actually quite the same as BCD encoding.
You can either choose to print an address in Long Format, which means in full. This way you'll have to print a lot of zeroes. Which is something that you'll avoid by printing addresses in Short Format. In short format addresses you compress the address by omitting blocks of zeroes. You may also omit leading zeroes within each 16-bit block. Finally you may also reduce multiple colons to a single pair.

1080:0000:0000:0000:0008:0800:200C:417A
1080::::8:800:200C:417A
1080::8:800:200C:417A

Please note though that you may remove zero-blocks in only one location in the address. Else nobody will be able to rebuild the original address!

It is also possible to translate IPv4 addresses so they can be used in an IPv6 environment. Leave the octets in decimal format and prefix a double colon. Thus the following conversion takes place. **192.168.0.10 -> ::192.168.0.10**. See page 24.

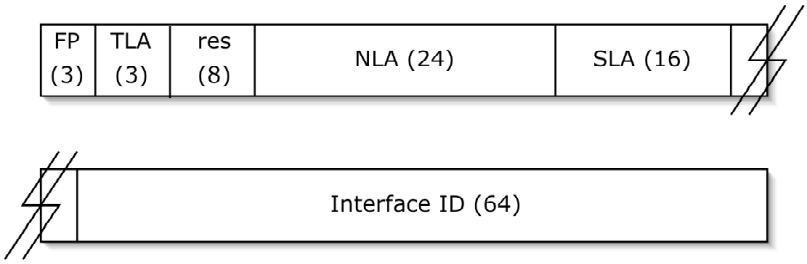
IPv6 does not make use of netmasks like IPv4 does. Instead it uses prefixes in a similar fashion. If for example, you have address **12AB:0000:0000:CD30:1234:4567:98AB:CDEF/60** then the first sixty bits are marked as the subnet prefix. Which in this case is **12AB:0000:0000:CD3(0)**.

There are three types of IPv6 addresses. Each type has a discrete range (LAN, site or global) and can be divided into unicast, multicast and anycast as well. Uni- and multicast addresses are comparable in use to those kinds of addresses in IPv4. Anycast addresses however are kind of like multicast addresses, but with a twist: in this case the datagrams only get delivered to one node -> the one that is perceived as being the closest to the source node. Broadcast addresses are not a part of IPv6.

Instead of using classes to put hierarchy in addresses like IPv4, IPv6 breaks the addresses into separate components. Each type of unicast address has a different internal structure.
* Aggretable global unicast addresses are for use on the Internet.
* Site local unicast addresses are not allowed to go outside the boundaries of the local site. They get blocked by routers.
* Link local unicast addresses are not allowed to go through routers at all and are thus limited to the current network segment.

Aggretable Global Unicast

Aggretable Global Unicast addresses were designed in such a way that one single routing table entry can identify a vast number of networks. Addresses are built in such a way that allows global routing.



Format prefix	Always "001".
Top Level Aggregator.	Ranges assigned by IANA to large providers like RIPE and ARIN who can divide ranges between country-wide organisations (like AT&T and BT).
Reserved	Extra bits for the TLA should they ever be needed.
Next Level Aggregator.	Describes level below the TLA, for instance a second tier provider.
Site Local Aggregator	Allocated to a link within a site. If a company switches provider both the TLA and NLA will change, but the SLA will remain the same. IPv6's auto-configuration ensures that you will NOT have to manually reconfigure all of your nodes' TLA and NLA settings.
Interface ID	A unique value for each network interface.

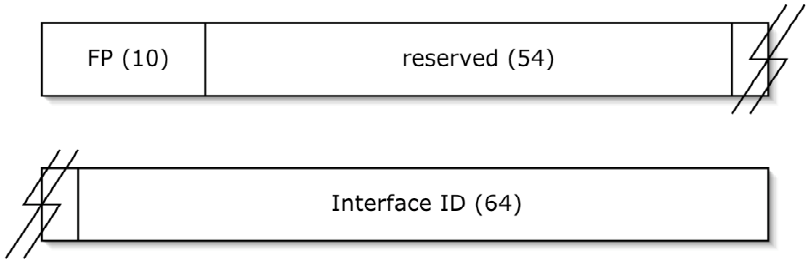
Site Local Unicast

Intended for site-local, non-internet usage as uniqueness can only be guaranteed within your site.

Format prefix	Always "1111111011", or "FEC" in hex.
Subnet ID	Assigned by the organisation at its own discression, for example to identify different networks within its organisation.

Link Local Unicast

These addresses are limited to a single network and cannot be routed. To be used by hosts on the same link.



Format prefix Always "1111111010", or "FE8" in hex.
Reserved Unused.

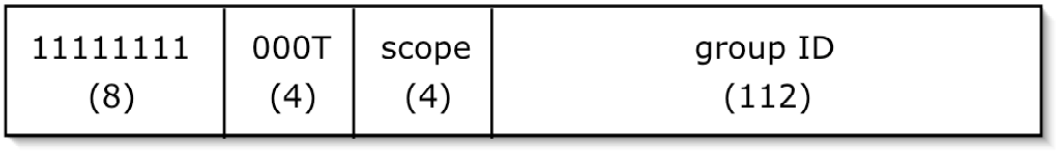
Aside from the address types mentioned above, IPv6 also has three special types of unicast addresses:
* An unspecified address consists of all zeroes -> ::.
* A loopback address consists of all zeroes, but ends in a 1 -> ::1.
* IPv6 address with embedded IPv4 address. These special cases we'll discuss below.

IPv4 unicast in IPv6 environment

An IPv4 compatible address is used to guide IPv6 traffic across IPv4 routers.

An IPv4 mapped address is used for IPv4 hosts in an IPv6 network.

IPv6 multicast address



Format prefix Always "11111111", or "FF" in hex.
Flags "000"+transient bit, which is either "0" for a well known, permanent address (assigned by IANA) or "1" for a non-permanent address, used on a private network.
Scope Current assigned values are:
0001 = 1 = node local 1000 = 8 = organisation local
0010 = 2 = link local 1110 = E = global
0101 = 5 = site local
Group ID Definition of multicast group. IANA has already reserved a large number of these for special use on the Internet, eg the NTP multicast group is 43. Also, GID 1 is "all nodes" and GID 2 is "all routers".

IPv6 anycast addresses get routed from their source host to the nearest host with the specific anycast address. These address cannot be distinguished from normal unicast addresses. Instead, a unicast address automatically becomes an anycast address once assigned to multiple nodes. Mind you: anycast addresses may not be used as the source address for a datagram.
Solaris 8 cannot be configured as an IPv6-only host. Currently, IPv6 is implemented as an optional second IP stack. There are two forms of auto-configuration:
* Stateful, using DHCP.
* Stateless, based on the local host's MAC address.

Stateless auto-configuration uses the 48 bit IPv4 MAC address to build a 64 bit IPv6 address, according to IEEE EUI-64 standards. The process of calculating said address is as follows:
1. Obtain a MAC address.
2. Convert the first two characters into a 4 bit nibble.
3. Invert the high order seventh bit.
4. Insert "FFFE" between the CID and VID of the MAC address.
4. Convert back to hexadecimal.
5. Add a prefix of FE80 and pad with three blocks of zeroes.
6. The **`ifconfig`** command will add "/10" to indicate the prefix (which shows that the address is link-local).

Hosts who undergo stateless auto-configuration will automatically calculate their own link local unicast address according to the 'formula' above. Then the host will issue an ICMPv6 "router sollicitation" which will make sure that the default router forwards information to the host which allows it to build its global aggretable unicast address. Alternatively a router may also advertise itself and its settings for global IP addresses by issuing an ICMPv6 "router advertisement". All of this ensures that an IPv6 interface is reachable using multiple IPv6 addresses at once.

The command to use to turn on automatic configuration would be: **`ifconfig $device inet6`**.

Manual configuration of an IPv6 interfaces would look like:
```
ifconfig $device:$num inet6 plumb
ifconfig $device:$num inet6 $address/$suffix up
```

To ensure that your IPv6 interfaces is brought up at boot time, create the following two files:
* **`/etc/$hostname6.$interface`**, eg **`/etc/kilala6.en0`**.
* **`/etc/inet/ipnodes`** (comparable to the hosts file).

Revision history

1.0	11-15 sept 2004	Initial creation.
1.1	15 sept 2004	Corrections.
2.0	15 sept 2004	Rebuild document layout.
2.1	15 sept 2004	Resolve unclarities in text.
2.2	19-20 sept 2004	Corrections and resolving more unclarities.
	1 oct 2004	More corrections.