UFEEHX-20-2

Computer Networks & Operating Systems

Ian Johnson
(Networks – term 2)
Nigel Gunton
(*MODULE LEADER* O.S. – term 1)

Where can you find us?

Room: 3Q77
Telephone: extension 83167
Email:
Ian.Johnson@uwe.ac.uk
Nigel.Gunton@uwe.ac.uk

http://www.cems.uwe.ac.uk/~irjohnso/uqc109s2.html
http://www.cems.uwe.ac.uk/~ngunton/networks.html

Resources

1. The WWW

- http://www.cems.uwe.ac.uk/~ngunton
- http://www.cems.uwe.ac.uk/~irjohnso

We both maintain material for a variety of modules, have a dip around!

Website for the supplemental networks text

- http://www.netbook.cs.purdue.edu

Search engines, learn to use them effectively!

Linux Documentation Project

- http://www.tldp.org

Many mirrors, maintains free documentation, examples, tutorials, etc. – Very useful and multilingual.

RFCs

- http://www.rfc-editor.org

This is the definitive source for RFCs. Many mirrors, particularly of commonly referenced ones.

2. Dead Trees

- On networking
- On unix & unix-like operating systems
- On C Programming

The library!

Course Text:

"*Introduction to Operating Systems*", John English, Palgrave

Supplemental Course Texts:

"*Operating Systems with Linux*", J. O'Gorman, Palgrave, 2001.

"*Computer Networks & Internets*", D.E. Comer, Prentice Hall, 4nd Ed. 2004

There are numerous other valuable books you could refer to.

Anything by W Richard Stevens
Anything by Andrew S Tannenbaum

Linux Socket Programming by example – Que
Computer Networking – W Stallings, Pearson.

3. Worksheets

These are intended to help and assist! Do them!

What is this module about?

Networks & Operating Systems ☺

Two inter-related topics, e.g.

- **Network administration**
- **Security**

Does *ifconfig* belong under Networks or OS?

Lab Sessions and lectures will not synchronise!

Networking Topics (ch10 English)

- Introduction
    - A little history
    - Why network?
    - Basic tools – probing the internet (ch2 Comer)

- An intro to network programming using a simplified API (ch3 Comer)

- IP Networking
    - TCP, IP, UDP, ICMP
    - Routing, naming, address & name resoloution.ARP,
    - RARP & DNS. IPv4 address space
    - Network layers & stacks (mainly part IV Comer)

- Higher level protocols (HTTP, HTCPC)

- Clients & Servers, socket programming & RPC
(chap 28,29 30 & 38 Comer)

- Network equipment: routers, bridges, hubs switches

- Network Security (chap 40 Comer)

NOTE THIS LIST IS NOT IN ORDER!

We'll only get to Security if we have time (usually not!)

Introduction

A Brief History

Computer networking as we understand it began in the 1960's. As computer systems became more important and supported time-sharing, but were also extremely expensive and rare, ways of allowing remote users access became important.

The first published work on packet-switching was:
Kleinrock, L, "*Information Flow in Large Communication Networks*", RLE Quarterly Progress Report, July 1961

This was swiftly followed by work from Baran at Rand, and Davies at NPL. (ACM SOSP (1967) "*A Digital Communications Network For Computers*"

The earliest packet switching nodes were known as IMPs and were built by BBN.

Figure 1: Leonard Kleinrock & the first IMP (1969) (from Kleinrock's Homepage)

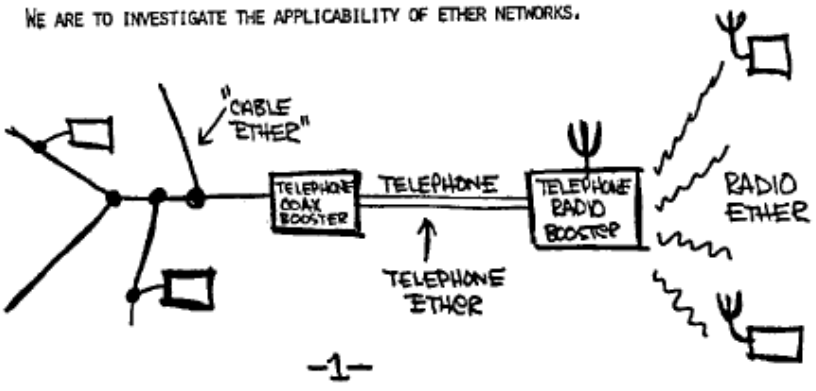
The first IMP was installed at UCLA under Kleinrock in 1969, with the next 3 at SRI (Stanford), UC Santa Barbara, and the University of Utah.

In 1969, ARPAnet the precursor to the internet had 4 nodes.

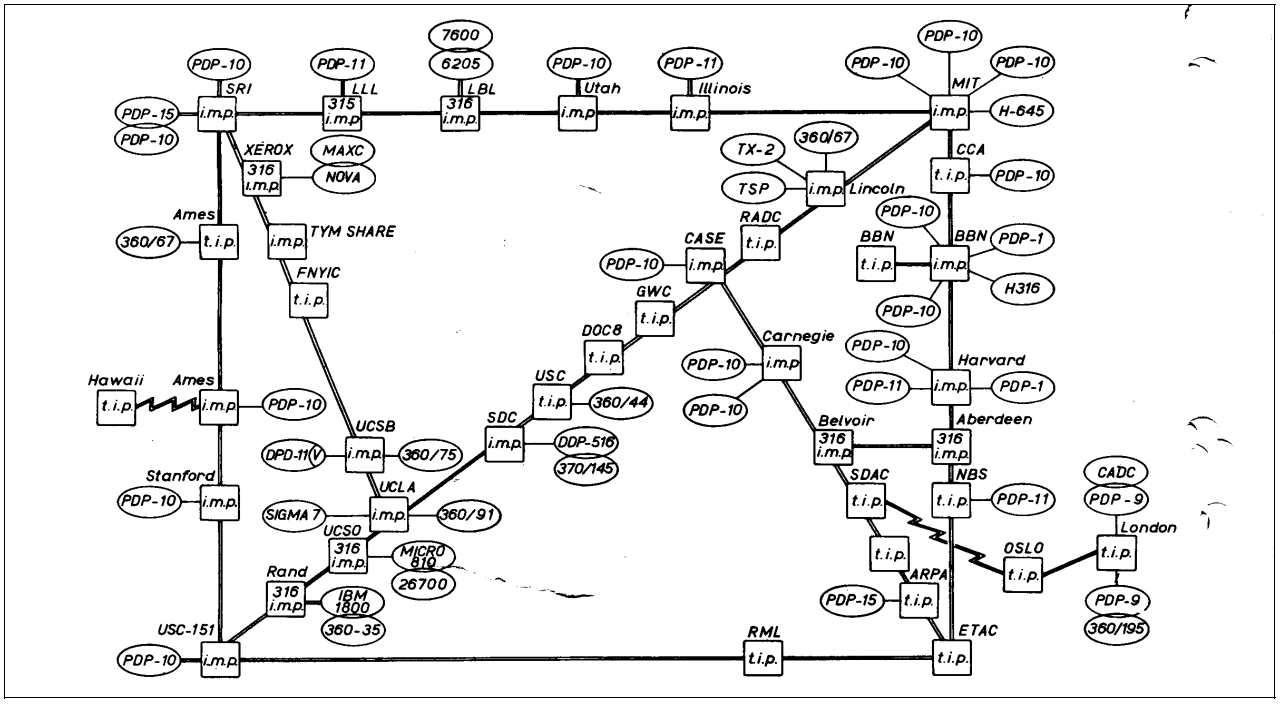
In 1972, 15 nodes and the first protocol and RFC (RFC001 – The Network Control Protocol (NCP).

Other competing packet switch networks also came into being at this time, e.g. IBM SNA.

1973 Robert Metcalfe invented Ethernet (from his original XeroX memo)



1973 ARPAnet:



1974 Cerf & Kahn get a DARPA grant to investigate “Internetting” and design IP.

1976 Metcalfe R.M. & Boggs D.R., “*Ethernet: Distributed Packet Switching for Local Computer Networks*”, CACM vol.19, no.7, 1976.

1979 200 nodes on ARPAnet, Metcalfe founds 3Com

1983 January 1st 1983 – NCP obsoleted and replaced by TCP/IP (RFC 801)

1984 JANET (UK) – 9.6 Kbps 50 sites – Based on X25/Coloured Book software

1986 NSFNET – 56 Kbps backbone interconnecting universities (US).

1991 **Explosion!**

- Berners-Lee releases the first WWW server

- Nicola Pellow (Placement student from Leicester Polytechnic UK.) releases text-mode portable browser.

- JANET supports TCP/IP

- NSFNET allows commercial access

1992 Demon Internet dial-up born (£10 a month + call charges).

1995 The end of NSFNET, Internet traffic is now carried by commercial providers. This is the net we know.

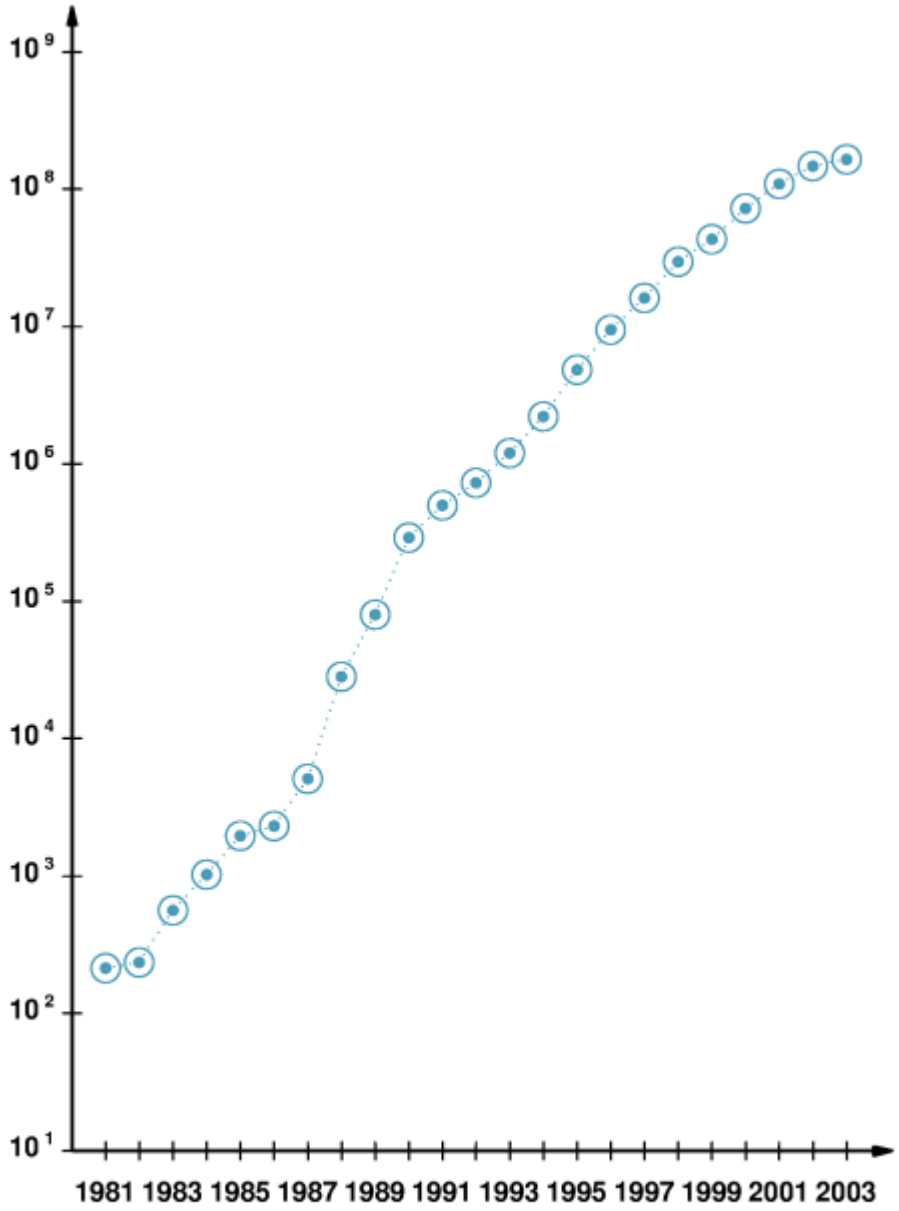


Figure 2.2 Internet growth from 1981 through 2003 plotted on a log scale illustrates the exponential growth.

For more history:

Naughton, J "***A Brief History of the Future***", Pheonix (Orion), 2000.
Hafner, K & Lyon, M "***Where Wizards Stay Up Late: the origins of the Internet***", Simon & Schauster, 1996

and Wikipedia entries for most of the personalities!

Why Network?

To share resources:

Originally computers …
… Then printers, disks etc.

Now information.

1981 IT ⟶ 2003 ICT

Probing the Internet (ch. 2 Comer)

For playing with ping & traceroute see also the network tools worksheet.

Ping

Possibly the simplest tool for probing the internet. We'll talk more about ping later.

NOTE: Many modern firewalls block ping packets!

```
irj@akira:~$ ping www.google.com
PING www.google.akadns.net (216.239.41.99): 56 octets data
64 octets from 216.239.41.99: icmp_seq=0 ttl=57 time=154.0 ms
64 octets from 216.239.41.99: icmp_seq=1 ttl=57 time=152.0 ms
64 octets from 216.239.41.99: icmp_seq=2 ttl=57 time=150.7 ms
64 octets from 216.239.41.99: icmp_seq=3 ttl=57 time=147.7 ms
64 octets from 216.239.41.99: icmp_seq=4 ttl=57 time=139.8 ms
--- www.google.akadns.net ping statistics ---
6 packets transmitted, 5 packets received, 16% packet loss
round-trip min/avg/max = 139.8/148.8/154.0 ms
```

What does this tell us?

- www.google.com is an ***alias*** for www.google.akadns.net
- It has the IP address 216.239.41.99
- we lost a packet, but why & which one?
- Timing information.
- A clue to the number of routers involved.
  - Ttl=60, and decrement as per spec.
  - Ttl=255, BSD et. al.
  - Leave unchanged, (½) or reset.

```
irj@akira:~$ ping mit.edu
PING mit.edu (18.7.21.70): 56 octets data
```

```
64 octets from 18.7.21.70: icmp_seq=0 ttl=243 time=182.5 ms
64 octets from 18.7.21.70: icmp_seq=1 ttl=243 time=198.6 ms
64 octets from 18.7.21.70: icmp_seq=2 ttl=243 time=180.8 ms
64 octets from 18.7.21.70: icmp_seq=3 ttl=243 time=181.4 ms
64 octets from 18.7.21.70: icmp_seq=4 ttl=243 time=181.8 ms
64 octets from 18.7.21.70: icmp_seq=5 ttl=243 time=180.9 ms
64 octets from 18.7.21.70: icmp_seq=6 ttl=243 time=180.8 ms
--- mit.edu ping statistics ---
7 packets transmitted, 7 packets received, 0% packet loss
round-trip min/avg/max = 180.8/183.8/198.6 ms
```

East Coast U.S. is “further” than google, (cf. Comer 48 ms)

```
irj@akira:~$ ping berkeley.edu
PING berkeley.edu (169.229.131.109): 56 octets data
64 octets from 169.229.131.109: icmp_seq=0 ttl=239 time=146.6 ms
64 octets from 169.229.131.109: icmp_seq=1 ttl=239 time=146.2 ms
64 octets from 169.229.131.109: icmp_seq=2 ttl=239 time=146.5 ms
64 octets from 169.229.131.109: icmp_seq=3 ttl=239 time=146.9 ms
64 octets from 169.229.131.109: icmp_seq=4 ttl=239 time=146.2 ms
64 octets from 169.229.131.109: icmp_seq=5 ttl=239 time=147.0 ms
```

Berkeley appears to be running BSD, now there is a surprise!
It is closer to us, (cf. Comer 31 ms)
More routers in the route to Berkeley than MIT.

Traceroute

Traceroute gives us information on a route packets ***may*** take, lost packets, and timings.

```
irj@akira:~$ traceroute berkeley.edu
traceroute to berkeley.edu (169.229.131.109), 30 hops max, 38 byte
packets
1 router.densitron.net (217.148.39.1) 77.377 ms 251.242 ms 3.516 ms
2 hs4-0.br2.ldn0.as8785.net (212.82.83.21) 2.878 ms 2.782 ms 2.611 ms
3 fxp0.cr4.ldn0.as8785.net (212.82.64.21) 4.630 ms 4.320 ms 3.792 ms
4 ge-7-0-0-1009.lon12.ip.tiscali.net (22 ms
5 so-1-0-0.was21.ip.tiscali.net (213.200.81.154) 179.758 ms 176.414 ms
175.432 ms
6 * interconnect-eng.Washington1.Level3.net (209.0.227.125) 75.466 ms
74.694 ms
7 so-5-0-0.gar2.Washington1.Level3.net (209.244.11.13) 75.462 ms
74.541 ms 74.802 ms
8 so-0-1-0.bbr2.LosAngeles1.level3.net (64.159.1.126) 137.283 ms
136.353 ms 136.281 ms
9 pos9-0.core1.LosAngeles1.Level3.net (209.247.10.202) 136.406 ms
136.556 ms 136.974 ms
10 6-1.ipcolo1.LosAngeles1.Level3.net (209.244.10.170) 137.223 ms
136.896 ms 136.963 ms
11 unknown.Level3.net (64.156.191.10) 137.705 ms 136.994 ms 136.393 ms
12 inet-ucb--lax-isp.cenic.net (137.164.24.142) 145.953 ms 146.193 ms
145.775 ms
13 vlan195.inr-201-eva.Berkeley.EDU (128.32.0.250) 146.599 ms 146.540
ms 146.567 ms
14 vlan209.inr-203-eva.Berkeley.EDU (128.32.255.2) 148.368 ms 147.017
ms 146.803 ms
15 arachne.Berkeley.EDU (169.229.131.109) 147.063 ms 146.874 ms
146.753 ms
```

A Gentle Network Programming Taster (Comer ch.3)

"A Programmer can create Internet application software without understanding the underlying network technology or communication protocols"

Comer, op. cit. p19

You can program sockets in any programming language you choose, each have their idiosyncrasies.

Comer provides a simplified API for socket programming in C in his book, and on the website.

Client-server computing is the basis for most internet services, (peer-to-peer systems are generally far more complex).

The basic idea:

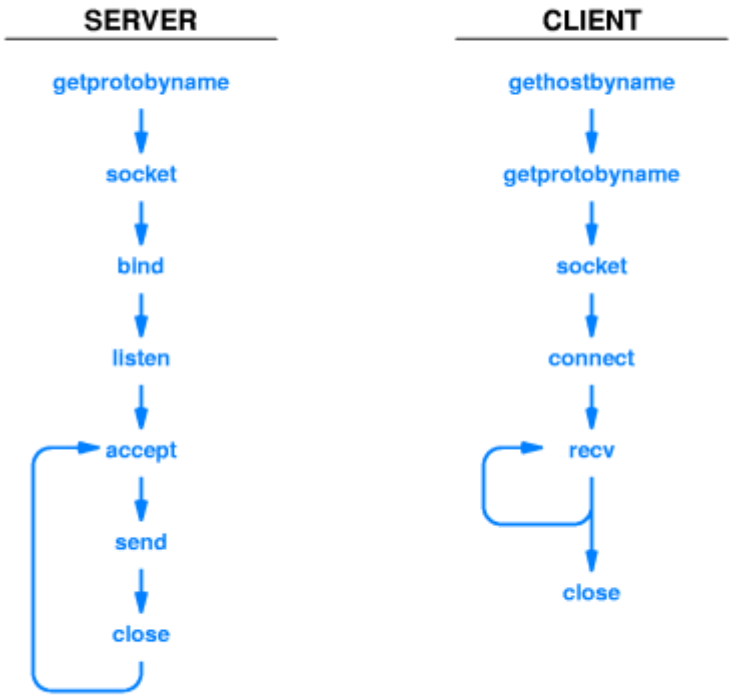
- A program (server) is started, and waits for another program (client) to contact it.
- The address of a server is a combination of machine & port.
- Have a look at /etc/services.
- Comer reduces this to computer/application.

Comer API (*Comer op. cit.*) :

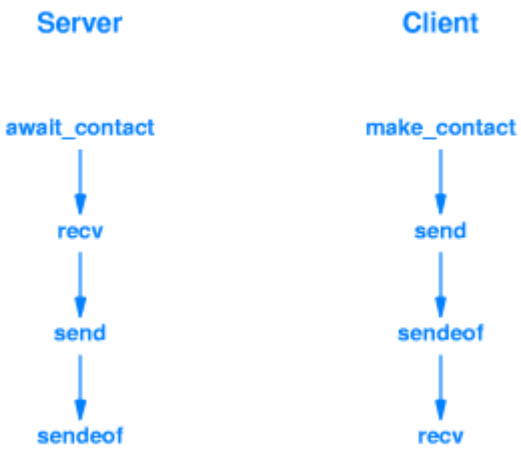
Operation	Meaning
await_contact	used by a server to wait for contact from a client
make_contact	used by a client to contact a server
cname_to_comp	used to translate a computer name to an equivalent internal binary value
appname_to_appnum	used to translate a program name to an equivalent internal binary value
send	used by either client or server to send data
recv	used by either client or server to receive data
send_eof	used by both client and server after they have finished sending data

Figure 3.1 An example API consisting of seven operations. These seven functions are sufficient for most network applications†.

Traditional Socket programming API:
*From Douglas E Comer, "**Computer Networks & Internets**", Prentice Hall, 1997*



Simplified API (Comer op. cit.)



We also have three types provided by the API:

- appnum
- computer
- connection

Lets have a look at the server (op. cit. Comer).

```
* echoserver.c */
#include <stdlib.h>
#include <stdio.h>
#include <cnaiapi.h>
#define BUFFSIZE 256
/*-------------------------------------------------------
*
* Program: echoserver
* Purpose: wait for a connection from an echoclient and
echo data
* Usage: echoserver <appnum>
*
*-------------------------------------------------------
*/
int main(int argc, char *argv[])
{
connection conn;
int len;
char buff[BUFFSIZE];
if (argc != 2) {
(void) fprintf(stderr, "usage: %s <appnum>\n",
argv[0]);
exit(1);
}
/* wait for a connection from an echo client */
conn = await_contact((appnum) atoi(argv[1]));
if (conn < 0)
exit(1);
/* iterate, echoing all data received
until end of file */
while((len = recv(conn, buff, BUFFSIZE, 0)) > 0)
(void) send(conn, buff, len, 0);
send_eof(conn);
return 0;
}
```

The Client

```
/* echoclient.c */
#include <stdlib.h>
#include <stdio.h>
#include <cnaiapi.h>
#define BUFFSIZE 256
#define INPUT_PROMPT "Input > "
#define RECEIVED_PROMPT "Received> "
int readln(char *, int);
/*-----------------------------------------------------------------
-
*
* Program: echoclient
* Purpose: contact echoserver, send user input and print server
response
* Usage: echoclient <compname> [appnum]
* Note: Appnum is optional. If not specified the
* standard echo appnum (7) is used.
*
*----------------------------------------------------------------
*/
int
main(int argc, char *argv[])
{
computer comp;
appnum app;
connection conn;
char buff[BUFFSIZE];
int expect, received, len;
if (argc < 2 || argc > 3) {
(void) fprintf(stderr, "usage: %s <compname>
[appnum]\n", argv[0]);
exit(1);
}
/* convert the arguments to binary format comp and appnum */
comp = cname_to_comp(argv[1]);
if (comp == -1)
exit(1);
if (argc == 3)
      app = (appnum) atoi(argv[2]);
else
if ((app = appname_to_appnum("echo")) == -1)
     exit(1);
/* form a connection with the echoserver */
conn = make_contact(comp, app);
if (conn < 0)
exit(1);
(void) printf(INPUT_PROMPT);
(void) fflush(stdout);
/* iterate: read input from the user, send to the server, */
/* receive reply from the server, and display for user */
while((len = readln(buff, BUFFSIZE)) > 0) {
/* send the input to the echoserver */
(void) send(conn, buff, len, 0);
(void) printf(RECEIVED_PROMPT);
(void) fflush(stdout);
/* read and print same no. of bytes from echo server */
expect = len;
for (received = 0; received < expect;) {
len = recv(conn, buff, (expect - received) < BUFFSIZE
? (expect - received) : BUFFSIZE, 0);
if (len < 0) {
send_eof(conn);
return 1;
```

```
}
(void) write(STDOUT_FILENO, buff, len);
received += len;
}
(void) printf("\n");
(void) printf(INPUT_PROMPT);
(void) fflush(stdout);
}
/* iteration ends when EOF found on stdin */
(void) send_eof(conn);
(void) printf("\n");
return 0;
}
```

And in java, this is trivial!

English p312 (Server on p313)

```
import java.net.*;
import java.io.*;
public class Client {
   public static void main (String args[]) {
      try {
         Socket s = new Socket(args[0],2000);     // (1)
         InputStream  in = s.getInputStream();    // (2)
         OutputStream out= s.getOutputStream();
         String r = args[1] + "\n";               // (3)
         out.write(r.getBytes());
         int c;
         while ((c = in.read()) != -1) {          // (4)
            System.out.print((char)c);
         }
         s.close();                               // (5)
      }
      catch (Exception e) {                       // (6)
         System.out.println("Exception: " + e.getMessage());
      }
   }
}
```

Summary

At a simple level, all higher level protocols (e.g. http, smtp, telnet) involve opening and read/write operations between a client and a server

Protocols, Stacks & Layers (*Comer, ch16*)

Protocols

A protocol is simply an agreed method of communication.

In general computer protocols define formats and meanings

An example SMTP (Comer)

```
Server: 220 foobar.com Simple Mail Transfer Service Ready
Client: HELO example.com
Server: 250 OK
Client: MAIL FROM: <Mary_Smith@foobar.com>
Server: 250 OK
Client: RCPT TO: <John_Q_Public@example.com>
Server: 250 OK
Client: DATA
Server: 354 Start mail input; end with <CR><LF>.<CR><LF>
Client: ... body of mail message, which continues
Client: ... across as many lines as needed...
Client: <cr><LF>.<CR><LF>
Server: 250 OK
Client: QUIT
Server: 221 foobar.com Service closing communication channel
```

Protocols hide complexity by providing a higher level interface to something, similar in many ways to an API.

A “networking” protocol would have to define everything from the communication medium up. This is not really a good idea.

Instead we design protocol suites – sets of related protocols

However, we want to avoid:

- re-inventing the wheel
- duplication of functionality

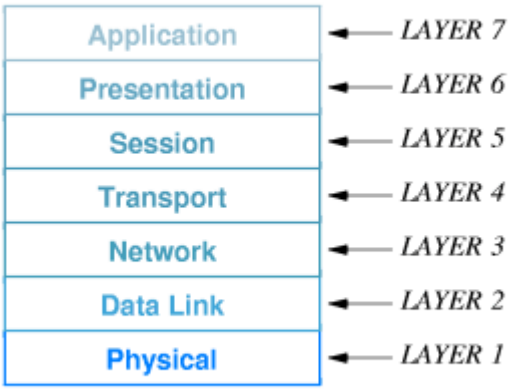
Layered Models

One way of avoiding duplication is to adopt a layered model.

Each layer “talks” to the layer below & above it via an API (or protocol)

The classic “7 layer model” is however obsolete. It is however useful to consider as a model:

From Comer: The ISO OSI 7 Layer model



Layer 1: Physical

Hardware & Wiring etc, for Ethernet the specification of voltages, cables e.g. cat5, coax etc.

Layer 2: Data Link

How to organise data into frames for specific network formats, (e.g. Ethernet); CRCs, checksums, byte-stuffing etc. Basically how to get bits from A to B, dealing with duplicate or damaged frames, for broadcast networks would include collision resolution, e.g.CSMA/CD for Ethernet.

Layer 3: Network

How packets are forwarded between networks, fragmentation address resolution & assignment. Routing.

Layer 4: Transport

Reliable & Unreliable Transfer

Layer 5: Session

Handling duplexing / turn taking. How to perform remote logins etc.

Layer 6: Presentation

How are real numbers represented? Byte ordering etc. e.g. XDR.

Layer 7: Application

The program a user interfaces talks its own protocol e.g. ftp, http etc. are here.

How does TCP/IP fit into this model?

It doesn't! ☺

It is useful to compare the two approaches:

- TCP/IP has an application layer, it provides smtp, ftp, http amongst others.

- The application layer via protocols such as MIME and XDR provides the ISO level 6 (presentation) functionality.

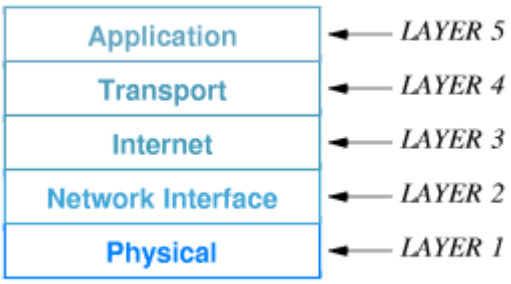
- The session layer (5) does not have an equivalent. Some of the functions expected here are provided by TCP.

- Layer 4 (Transport) is provided by TCP & UDP. TCP offering a reliable service, UDP an unreliable one.

- Layer 3 (Network e.g. X25) is replaced by Internet – provided by IP.

- Layer 2 (Data Link) and Layer 1 (Physical) roughly equal the subnet layer of TCP/IP. The subnet layer of TCP/IP is however not specified, it is simply assumed to exist! Some authors (e.g Comer – see below) separate these:

Due to Comer: 5 layer TCP/IP model



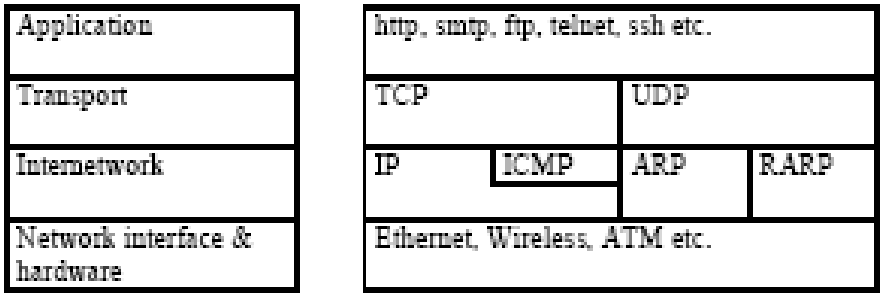
Stacks

A “stack” is simply a “pile” of protocols forming a protocol suite.

Each layer in the protocol suite has a corresponding layer in the stack.

Each layer “wraps” higher levels adding its own data (e.g. headers, checksums etc.)

Lets take a closer look at TCP/IP



.

Application Layer *Exemplar – http*

- http is the hypertext transport protocol
- http is the application protocol for WWW services
- http1.1 is defined by rfc2616
- http is ASCII based (not strictly true)

/etc/services shows:

```
http 80/tcp www www-http # WorldWideWeb HTTP
http 80/udp www www-http # HyperText Transfer Protocol
```

So http can use tcp or udp although generally it uses tcp.

How http works:

(for comprehensive details see the RFC)

1. Client is given a string:

   (URI formerly URN, formerly URL – although technically these are different!)

   Protocol_Scheme://Host:Port/Resource

   Protocol scheme specifies the protocol to talk e.g. http or ftp (or more archaically gopher)

   Host & Port specify where to find the service

   Resource is what is requested.

2. Client parses string, and resolves host.

3. Client connects to host on port Port and sends a plain text message (request).

4. Client waits for response.

Typically,
http://www.google.com/ will result in the client contacting the host www.google.com on port 80, and sending the string:

GET / HTTP……

Summary

Http protocol conversations are simple socket connections communicating text.

This is an Application Protocol and is fully defined by the RFC

TCP vs. UDP

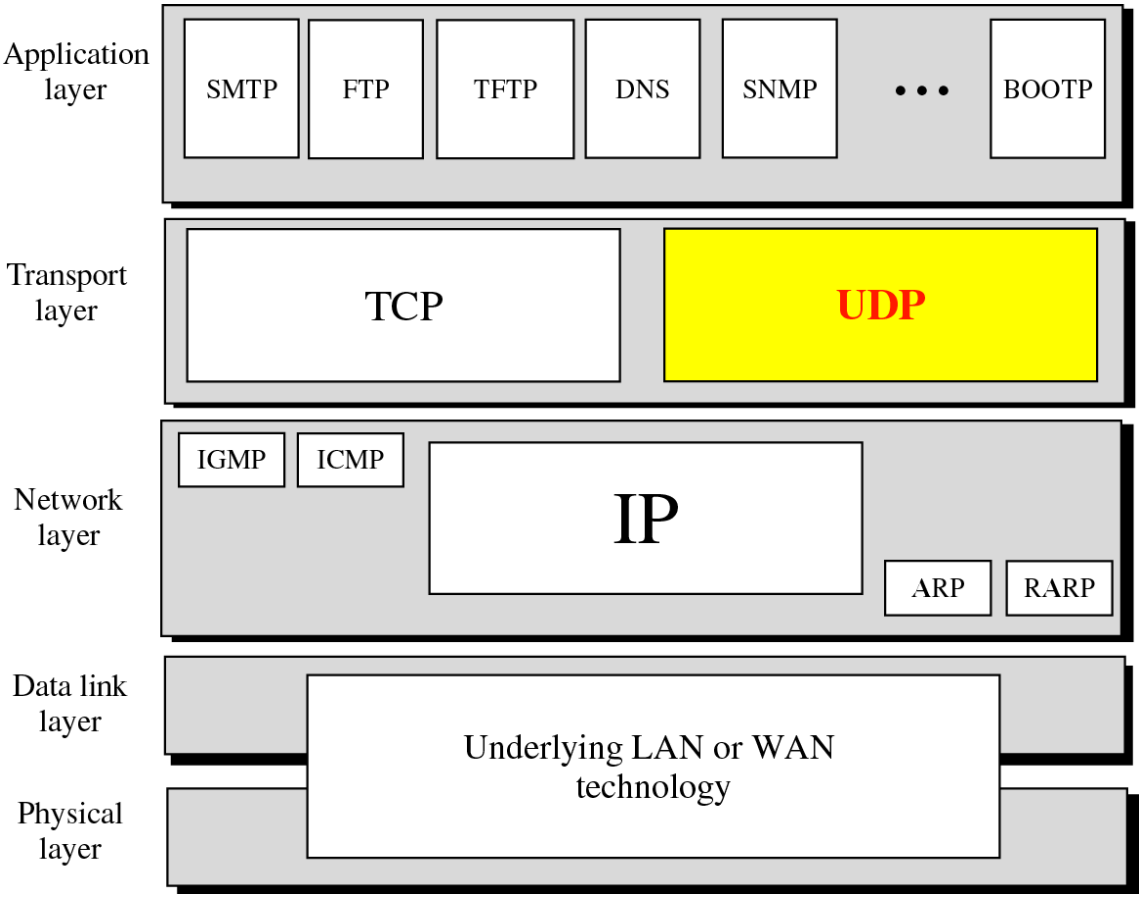
These are our transport level protocols. They are responsible for "transporting" data between processes.

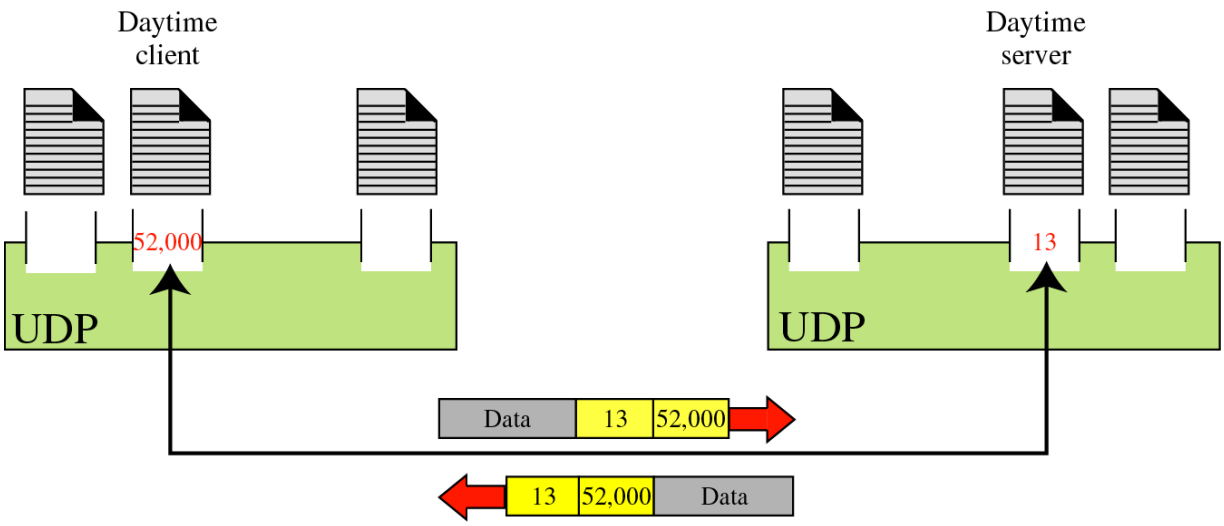
TCP = Transmission Control Protocol
UDP = User Datagram Protocol

UDP is connectionless and unreliable. It adds only port addresses, check sums and length.

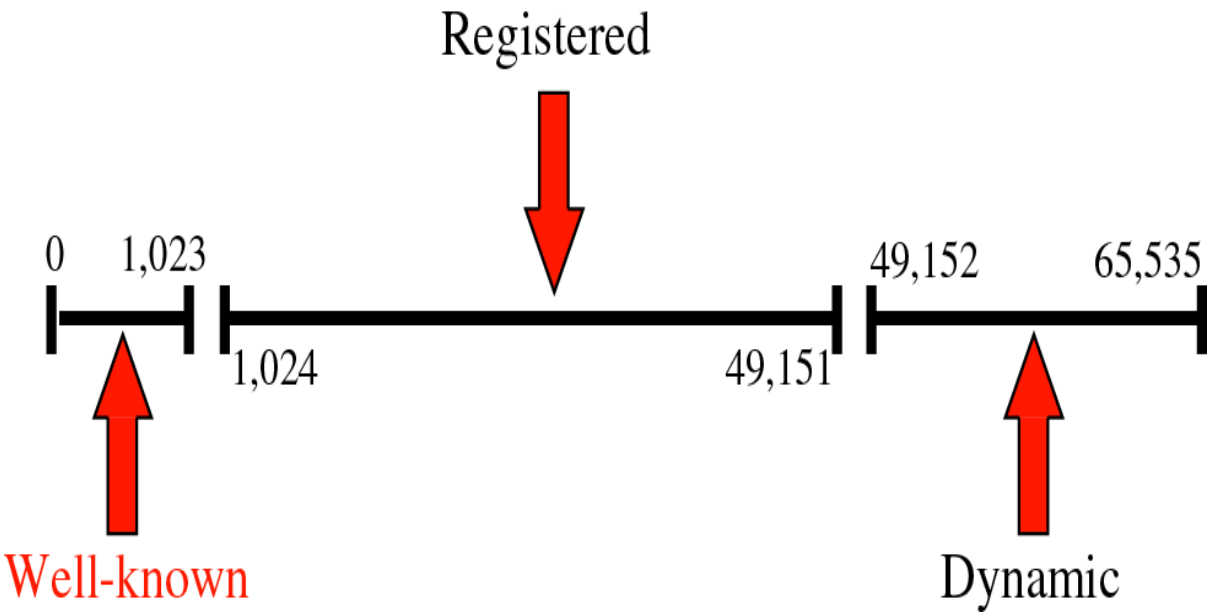
TCP is a reliable connection orientated stream service. This is obviously more complex to implement.

From Forouzan, "TCP/IP Protocol Suite"



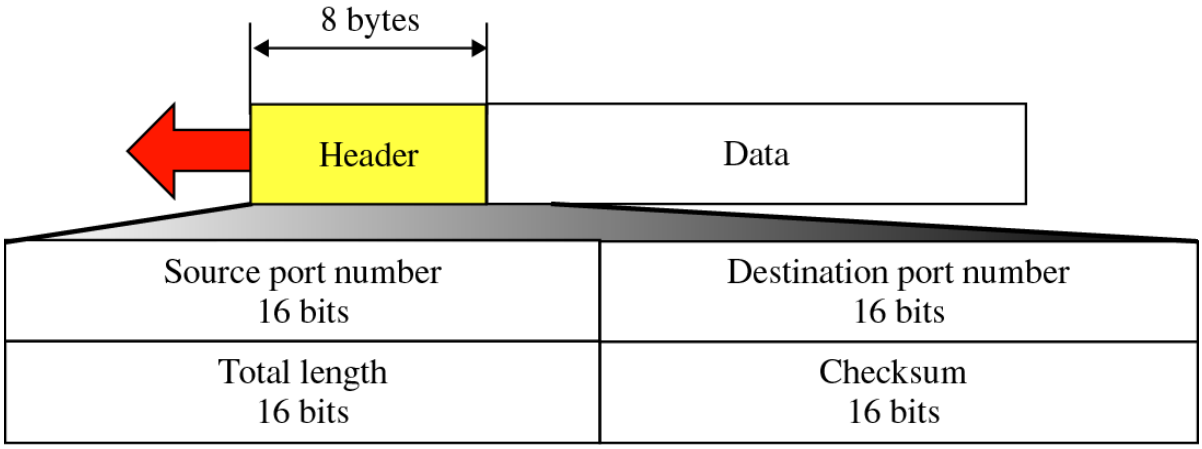


Client has a randomly allocated dynamic unprivileged port. Server is a Well Known Service



Some Operating Systems refer to these differently, for example BSD derived systems would call these reserved, non-privileged and ephemeral respectively.

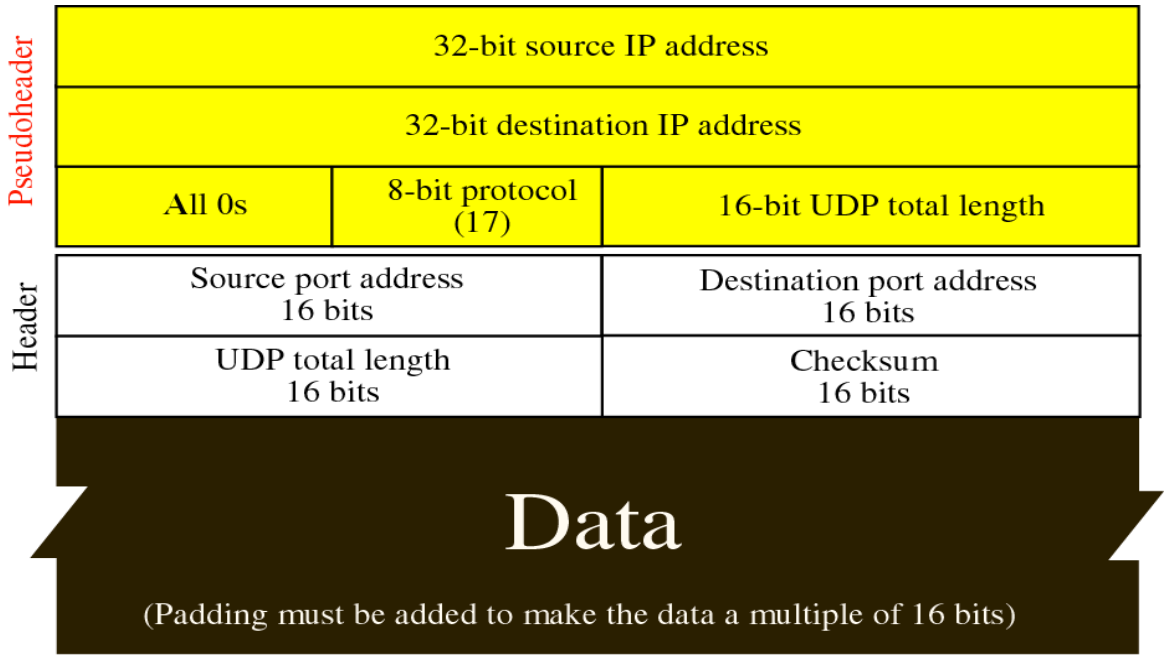
What does a User Datagram look like?



"Total length" is a 16 bit field therefore the maximum datagram size is 2^{16}-1 or 65535 bytes, **BUT** the IP header (more later) will consume 20 bytes, the UDP header (shown above) 8 bytes so the actual maximum length is 65507 bytes.

How is a UDP checksum calculated?

First add a pseudo-header:



Two problems are addressed by adding the pseudo-header pre-checksum:

1. How do we know it is a UDP packet?
2. How do we know destination address is uncorrupted?

We can now calculate the checksum:

153.18.8.105			
171.2.14.10			
All 0s	17	15	
1087		13	
15		All 0s	
T	E	S	T
I	N	G	All 0s

```
10011001 00010010 ──→ 153.18
00001000 01101001 ──→ 8.105
10101011 00000010 ──→ 171.2
00001110 00001010 ──→ 14.10
00000000 00010001 ──→ 0 and 17
00000000 00001111 ──→ 15
00000100 00111111 ──→ 1087
00000000 00001101 ──→ 13
00000000 00001111 ──→ 15
00000000 00000000 ──→ 0 (checksum)
01010100 01000101 ──→ T and E
01010011 01010100 ──→ S and T
01001001 01001110 ──→ I and N
01000111 00000000 ──→ G and 0 (padding)
-------------------
10010110 11101011 ──→ Sum
01101001 00010100 ──→ Checksum
```

1. Add pseudo header
2. Set checksum to zero & pad to a whole multiple of 16 bit words with zeroes.
3. Add all 16 bit words using 1's complement arithmetic
4. Complement (invert) the result
5. Remove pseudo heading & padding and pass to IP layer

The receiver adds pseudo header, pads & splits adds using 1's complement – result nonzero = checksum error

But…

Checksums are optional in UDP, the checksum field being zero if not used!

UDP summary (Comer ch24)

- Very simple
- Connectionless
- Not stream orientated
    - Application layer must chop data into UDP sized chunks, in other words this is the application developers problem.
- Unreliable
    - Receiver may over-flow
    - No flow or error control (except for the checksum – but a checksum error just results in the packet being dropped)
- Many to one relationship with applications
    - Many applications may wish to use the one UDP layer to send datagrams at the same time. UDP therefore multiplexes.
    - UDP (via port numbers) “knows” which application a datagram is intended for.
- Used by:
    - RIP
    - SNMP
    - TFTP
- Useful for broadcast & multicast services (e.g. SSDP in UPNP)

TCP (Comer ch25)

TCP or the Transmission Control Protocol is the more useful and complicated alternative to UDP.

TCP provides a connection orientated, reliable, buffered, stream orientated service, but to do so adds considerable complexity.

Connection Orientated

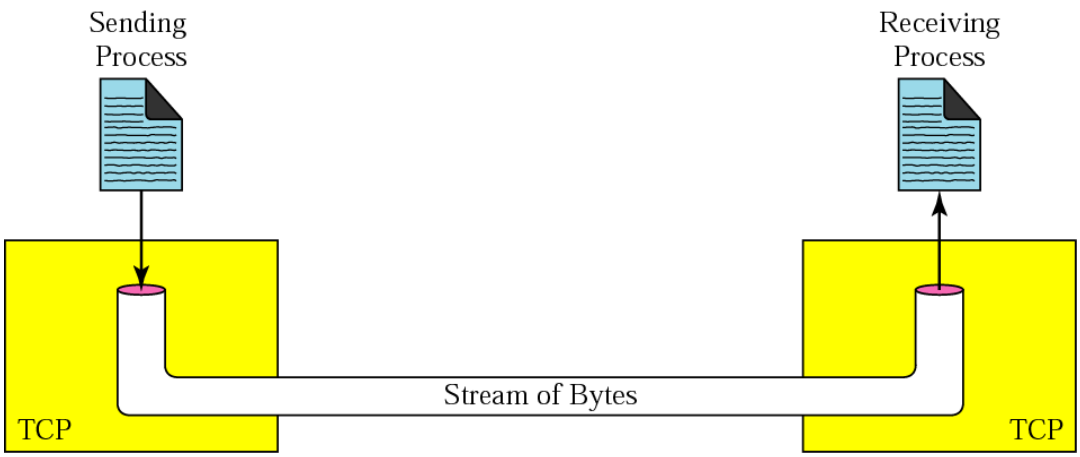
An application requests TCP to negotiate and establish a connection with a remote service (an “end-to-end” protocol). It then sends and receives data (full duplex) until it asks TCP to tear down (or terminate) the connection.

Reliable

TCP guarantees data will be reassembled in the correct order with no duplication or errors at the remote end.

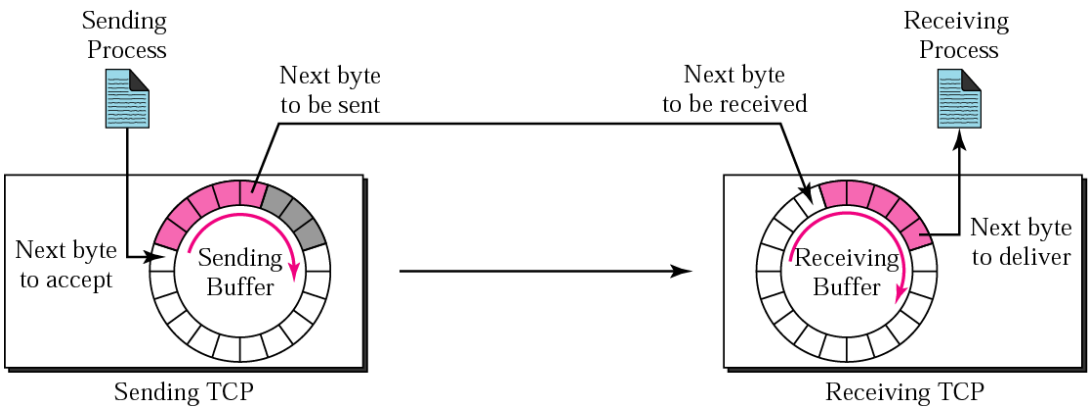
Stream Orientated

Data appears (to the application) to flow continuously. TCP has no concept of records and will send packets of a size TCP deems appropriate.



How does TCP achieve all of this?

Firstly, TCP buffers and segments data before passing it to the IP layer. Segments may be different sizes, maybe even kilobytes.



from Forouzan

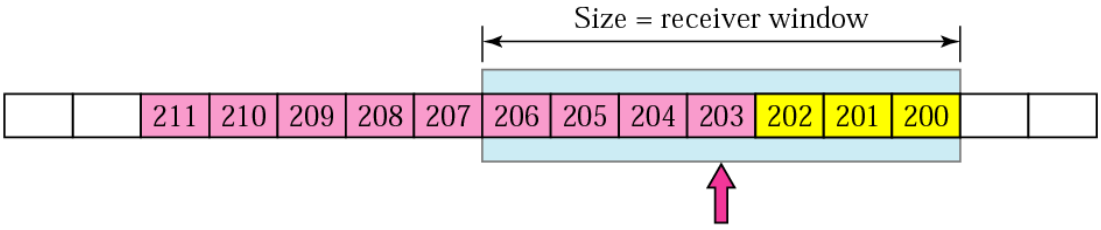
Secondly, it numbers every byte and adds a sequence number to every segment.

The sequence number for a segment is the number of the first byte it contains.

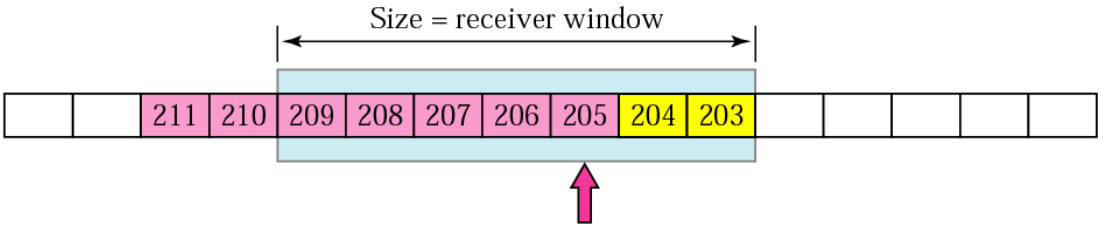
Sequence numbers are used by the receiving TCP layer to generate acknowledgement numbers. The acknowledgement number is the number of the next byte expected.

Next, TCP implements flow control via "sliding windows"

- Receiver advertises window size to sender
- Sender sets its window size to less than or equal to that value.
- Sender only sends until the window is full.



a. Before



b. After

Octets 200-202 have been acknowledged, so the window is slid along.

If the receiver is eating data faster than the sender is sending it, it can advertise this to the sender. The sender controls the size of the window and can grow or shrink it during the communication.

Let’s have a look at an example (from Comer):

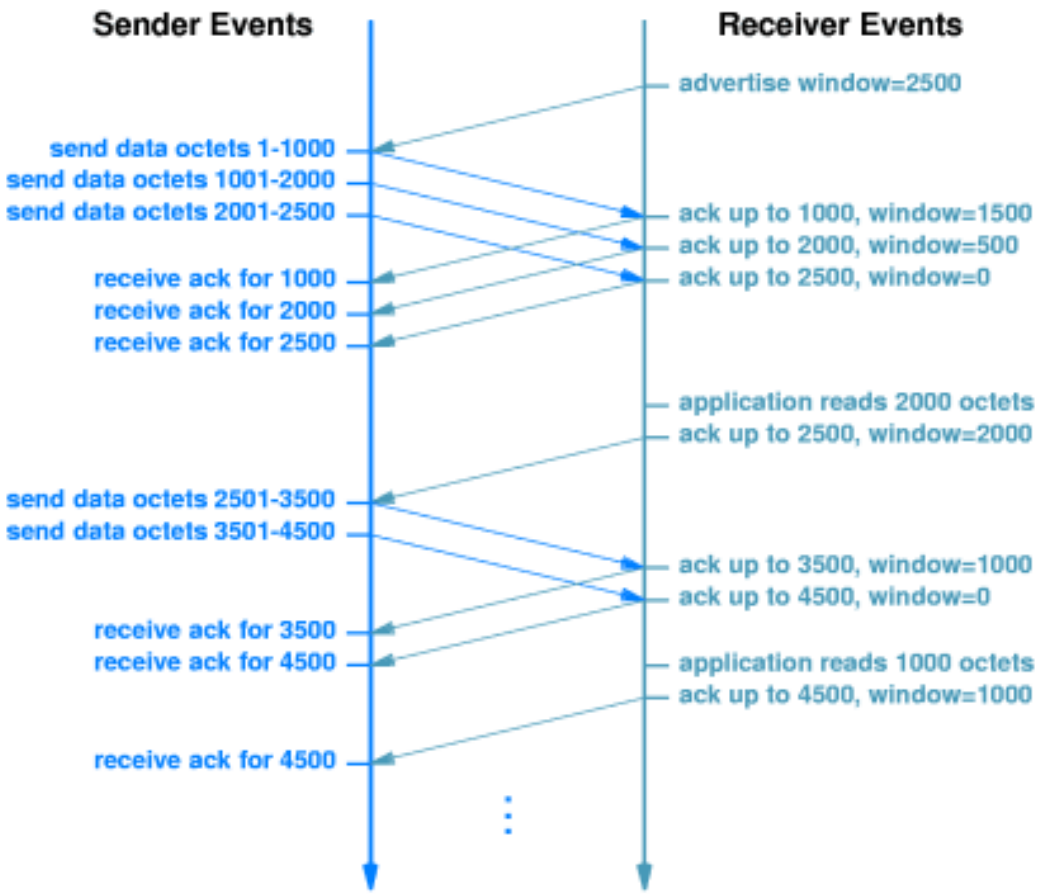


Figure 25.4 A sequence of messages that illustrates TCP flow control when the maximum segment size is *1000* octets. A sender can transmit enough data to fill the currently advertised window.

Silly Window Syndrome & Nagle's Algorithm

What happens if our producer (sender) or consumer (receiver) processes are seriously unbalanced, one being very quick, the other being very slow?

Theoretically, the receiver could shrink the window to a single octet or byte.

In this case we could end up with a 41 byte segment (40 bytes header + 1 byte payload).

totally inefficient! (1/41 = about 2.4% efficient)

Solution?

If caused by the sender,

Nagle's algorithm

- Send first "chunk" of data received from the application.
- Wait unit a maximum size segment has been built, or an ACK received before sending again.
    - Keep doing this!

If caused by the receiver,

Clark's solution

- Close the window – window size becomes zero, no data will be sent until it is reopened.
- Do not open the window until at least MSS can be received.

In reality most TCP/IP implementations contain heuristics to improve the performance beyond this measures!

Error Correction

TCP Doesn't really correct errors, it resolves them.

Each segment has an associated timer. If the timer expires before receiving an ACK, the segment is resent.

If receiver decides a segment is corrupt, (validates checksum) it does not ACK it.

If a receiver is happy with the segment – it ACK's it!

If the receiver never receives the segment ….

…. An example from Comer, note sliding windows complicate this in practice.

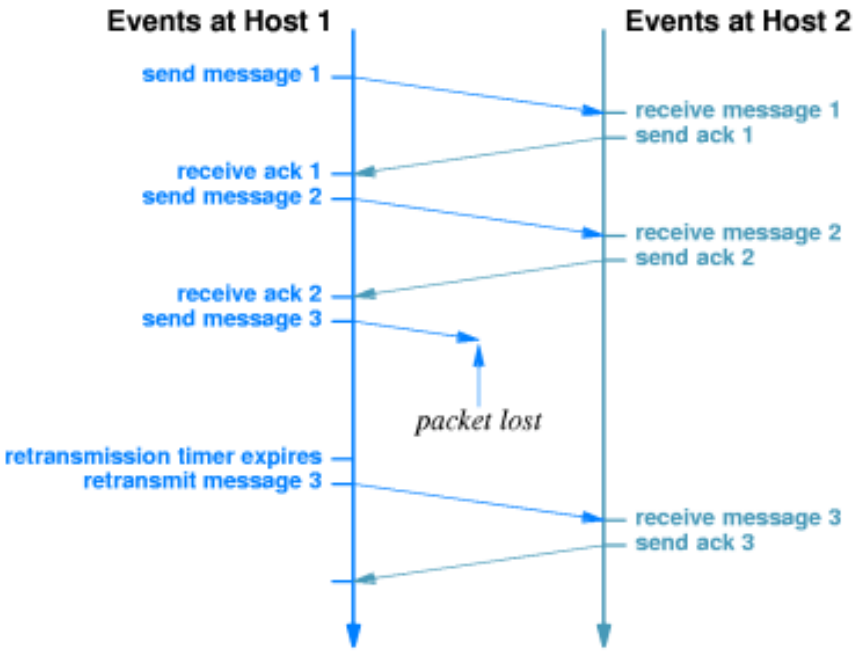


Figure 25.2 Example of retransmission. Items on the left correspond to events in a computer sending data, items on the right correspond to events in a computer receiving data, and time goes down the figure. The sender retransmits lost data.

How do we decide how long to wait?

TCP uses adaptive time-outs based on average round trip time.

Set up and Tear Down (Connection and Termination)

We need a reliable way to establish a connection.

Consider the “Two Armies” problem:

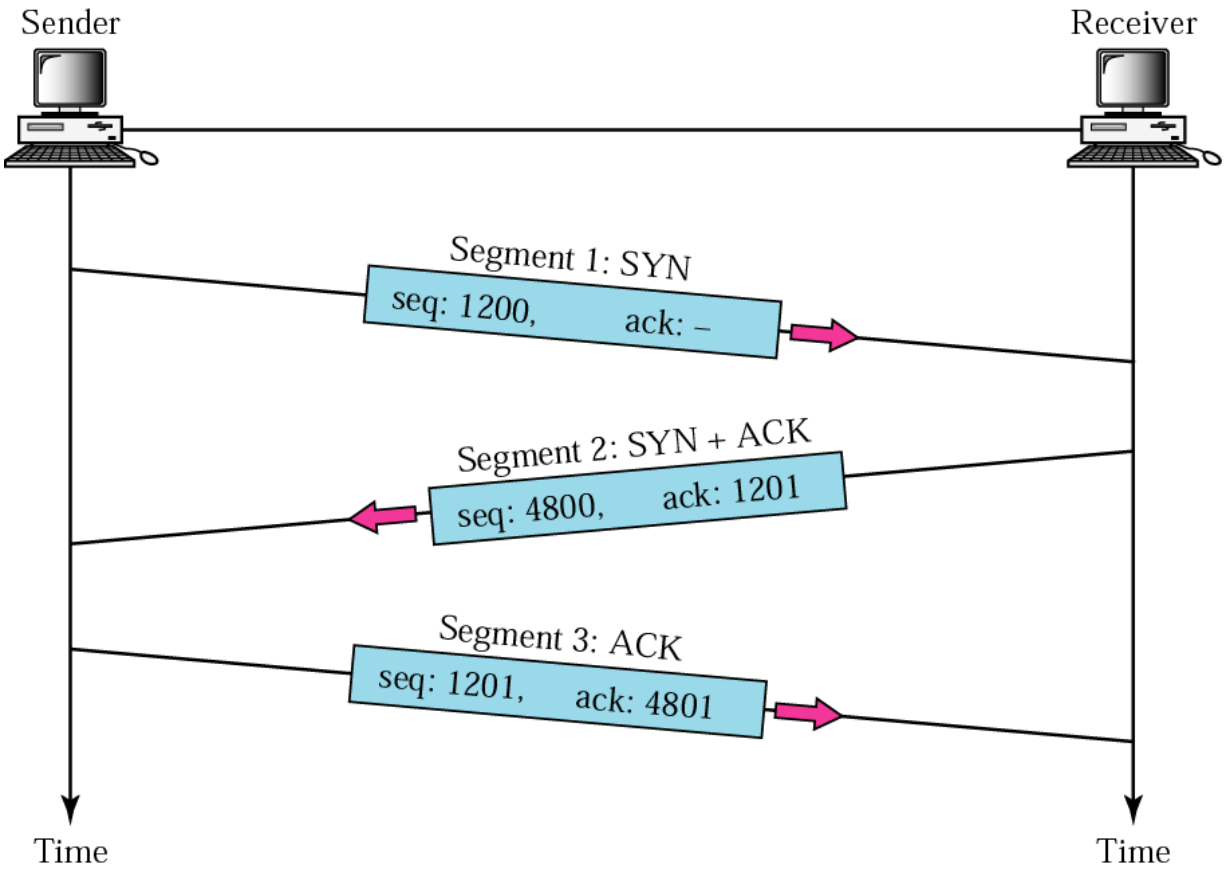
> Two blue armies are each poised on opposite hills, preparing to attack a single red army in the valley between them. The red army can defeat either of the blue armies separately, but will fail to defeat them if the attack simultaneously. The blue armies communicate via an unreliable communication mechanism (sending a foot soldier through the occupied valley). The blue commander with one of the blue armies would like to attack at noon. His problem is this: If he sends a message ordering the attack, he cannot be sure it will get through. He could ask for an acknowledgement, but that might not get through. What protocol could the blue armies adopt to solve this problem and avoid defeat?

Hint: This is the same problem that TCP faces in setting up or **SYN**chronising a socket connection.

Solution: Three way handshake

It can be proved that this is both a necessary and sufficient solution.

Let's look at Forouzan's start-up and Comer's tear-down diagrams:



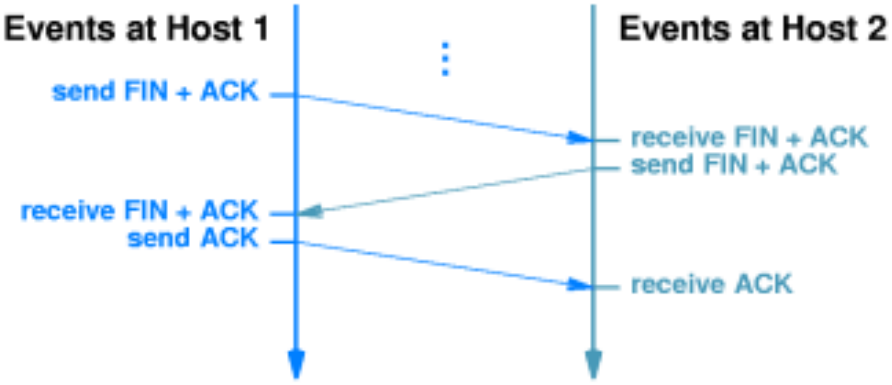
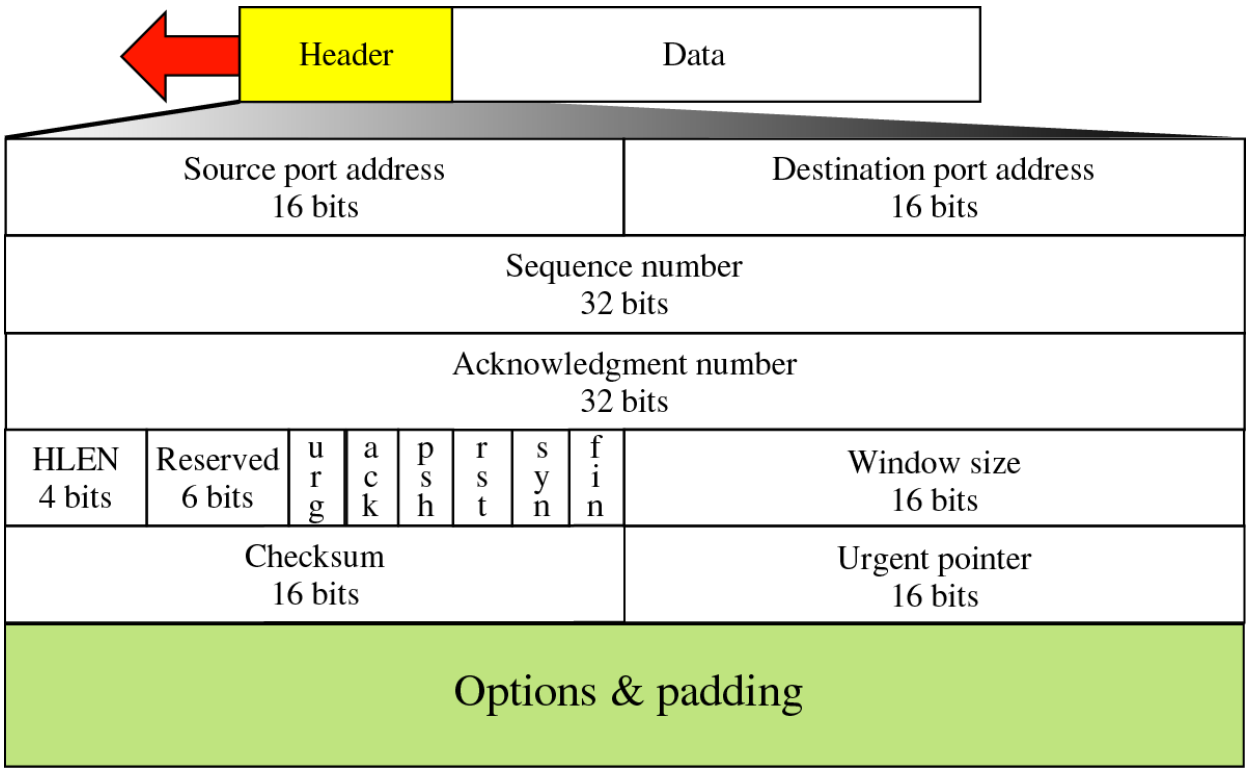


Figure 25.5 The 3-way handshake used to close a connection. Acknowledgements sent in each direction are used to guarantee that all data has arrived before the connection is terminated.

So, what does a TCP segment look like?



HLEN = Header length = Number of 32 bit words in header, where:

5 <= HLEN <= 15

(Header may be 20 to 60 bytes)

Reserved is unused. (although see RFC3514 – Bellovin's evil bit ☺)

Control Flags:

URG: Urgent pointer is valid	RST: Reset the connection
ACK: Acknowledgment is valid	SYN: Synchronize sequence numbers
PSH: Request for push	FIN: Terminate the connection

URG	ACK	PSH	RST	SYN	FIN

URGent – Data sent out of order

Pu**SH**ed – Buffer was flushed (*cf. fflush()*)

Re**S**e**T** – Resynchronise the connection

SYN - Synchronise

FIN – Finish

ACK - Acknowledge

It is worth noting that winow size (for sliding windows) could be changed with every TCP segment.

…In other words, it can piggy-back on data

TCP Options

TCP headers can have up to 40 bytes of additional data in them.

This is a (very) **partial** list!

MSS (Maximum Segment Size)

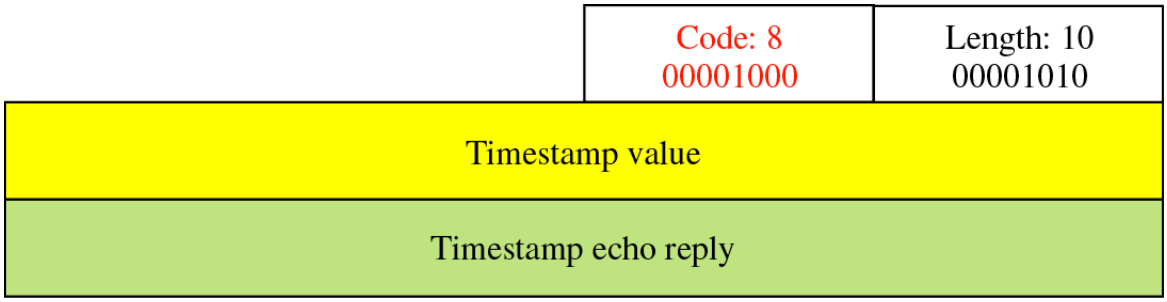
- This is actually the maximum data payload per segment!
- 16 bit value, default 536, maximum 65535 (2^{16}-1)

Window Scale Factor

- Power of 2 to be multiplied by window size.
- Real window = window_size * $2^{(\text{window scale factor})}$
- Scale factor can only be changed during initialisation (SYN)
- Maximum value supported by TCP is 16
    - In which case a window of 1, means 64k!
- Also known as the "***shift count***"
- How large a window do we need?
    - Depends on distance and speed of connection (Really latency & capacity)
    - Window size determines how much data can be sent before an ACK.
    - E.g. (Forouzan p323) OC-24 (1.244 Tbps) over 6000 miles. ACK received in a minimum 64 ms. Roughly 10Mb of data could be sent in this time!

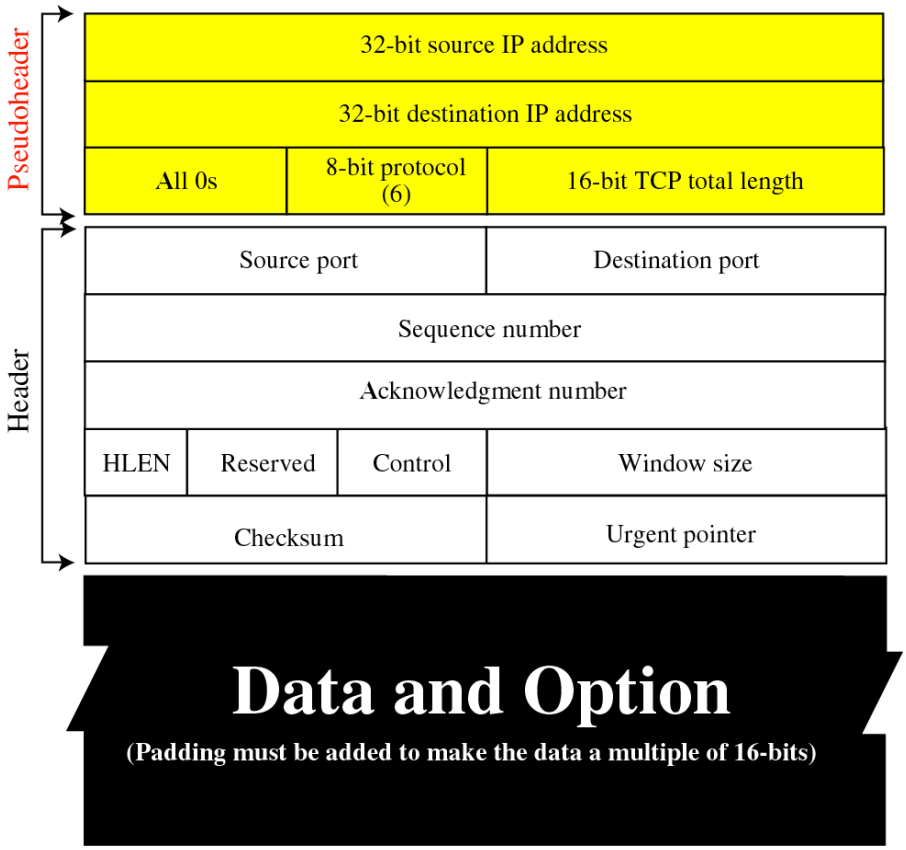
Timestamp

- 10 byte timestamp option (2 bytes for ID, type & length), with 2 times 4 byte timestamps.



TCP Checksum

Exactly the same procedure as for UDP, but whilst checksums are optional for UDP, they are mandatory for TCP.



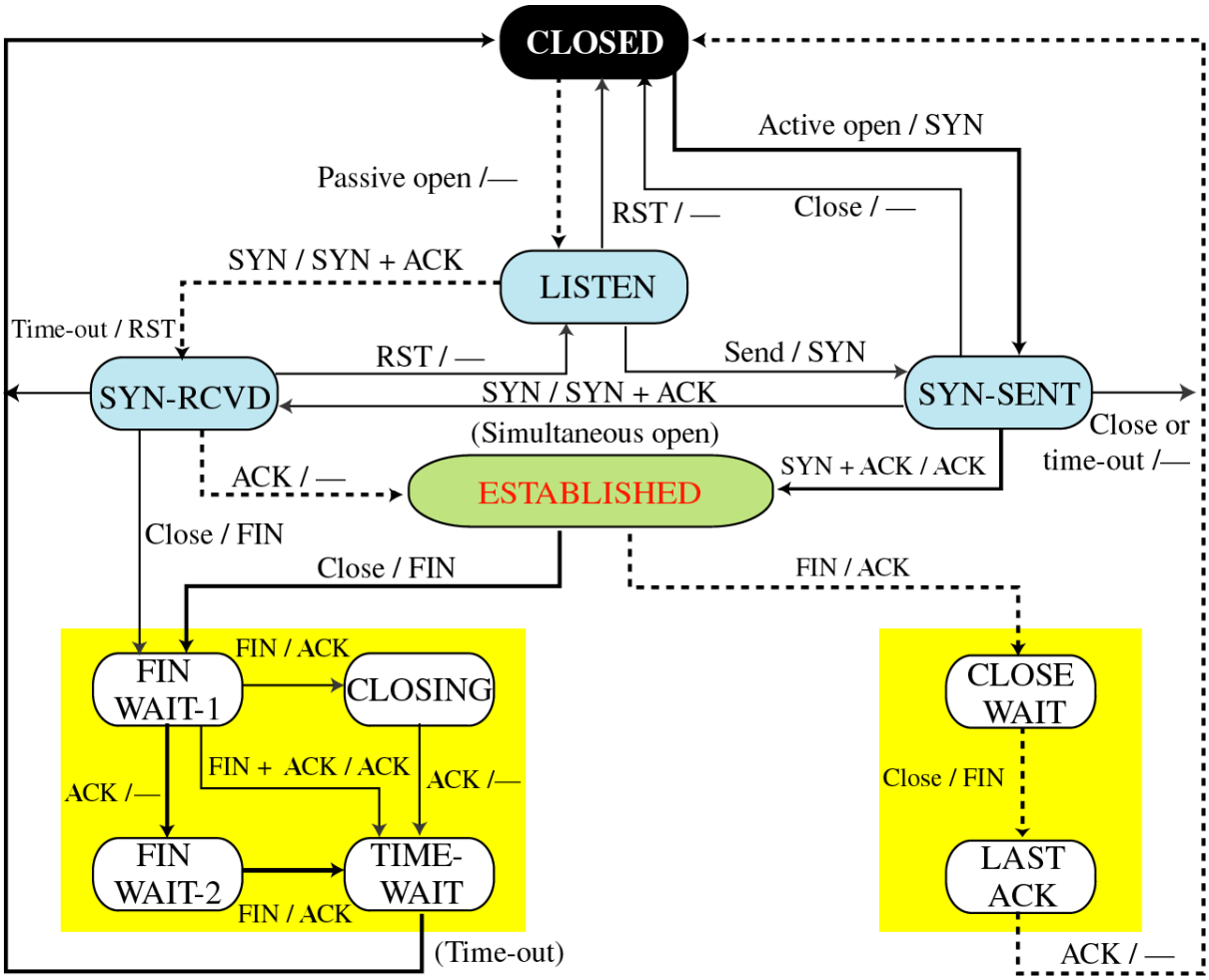
Putting it all together!

TCP is defined as a finite state machine.

Each transition has an input (or event) and associated output as well as the state transition.

Solid lines show client starts, the Dotted lines server. Both client & server start in the CLOSED state.

The following diagram illustrates this (due to Forouzan).



<u>This Diagram will not be needed for any Exam question!!</u>

TCP summary

- Reliable
  - Damaged or lost packets are retransmitted, Ordering maintained.
- Connection Orientated
  - Three way handshake for SYN & FIN
  - Remote end is confirmed as present and willing to talk
- Error Corrected
  - Not really, packets that fail checksum are retransmitted, giving the impression (and same result) as error correction.
- Sequenced, stream orientated
  - Every byte numbered, data usually processed FIFO, OS and TCP layer conspire together to manage the application
  - TCP sockets can be treated as a file.
- Adaptive congestion control
  - Massively glossed over (ignored) in these notes. Round Trip Time, ACK's (or lack of ACK's) and window size are all used to manage data transfer.
- Adaptive flow control
  - Linked to the above, managed by sliding windows.

The Internet Protocol (IP) (Comer ch18)

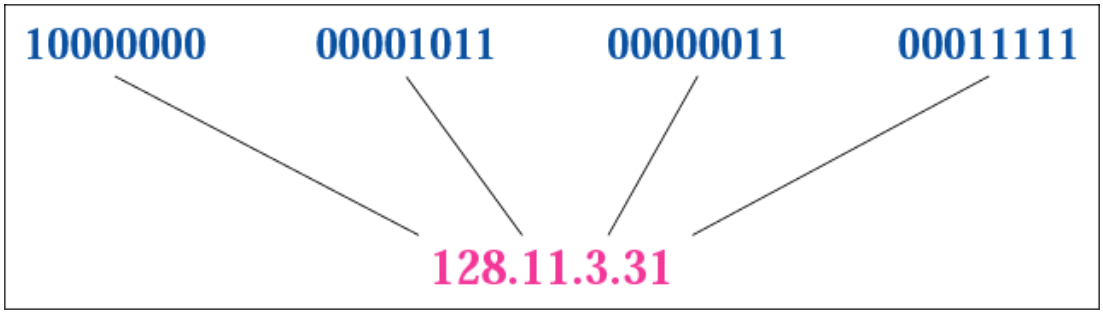
- IP provides our network layer. This is what TCP and UDP build on.
- IP provides a “best effort” datagram delivery service
- IP is connectionless, and PSS orientated. Datagrams can travel different routes and arrive in any order
- IP is concerned with delivery from machine to machine

IP Addressing

We’re going to discuss IPv4 here!

Every year is the year of IPv6, but thanks to CIDR and NAT not yet!

IPv4 addresses are 32 bit values, normally written in dotted quad decimal notation.



IP Addresses are unique

IP has an address space of 4,294,967,296 (4 Giga-addresses = 2^{32})

IP addresses can also be written in hexadecimal or binary.

Classful addressing

Should be growing obsolete, but very slowly.

Is replaced by CIDR in most installations today

BUT: Still useful when using private addresses behind a NAT router as in many home/SOHO environments.

Originally IP space was divided into 3 primary classes (A, B, C) together with class D (multicast) and class E (reserved).

Classes could be identified by the first 4 bits of the address.

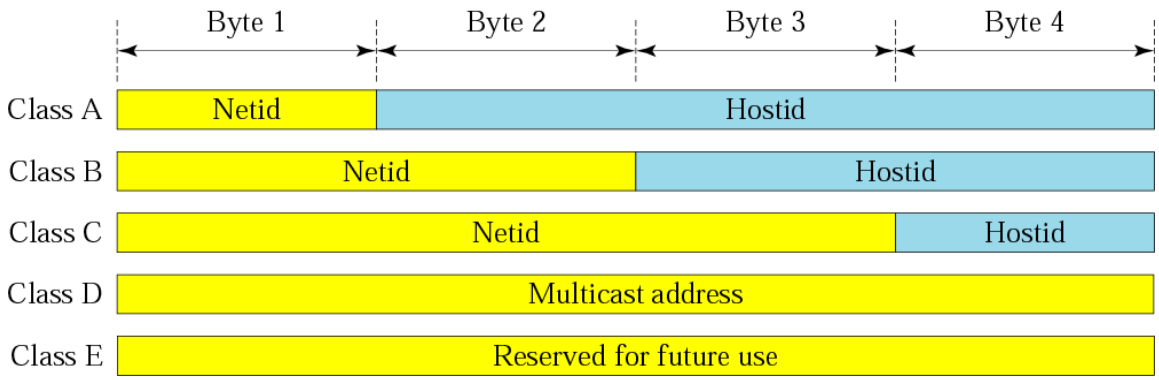
	First byte	Second byte	Third byte	Fourth byte
Class A	**0**			
Class B	**10**			
Class C	**110**			
Class D	**1110**			
Class E	**1111**			

Therefore valid first byte values for each class are:

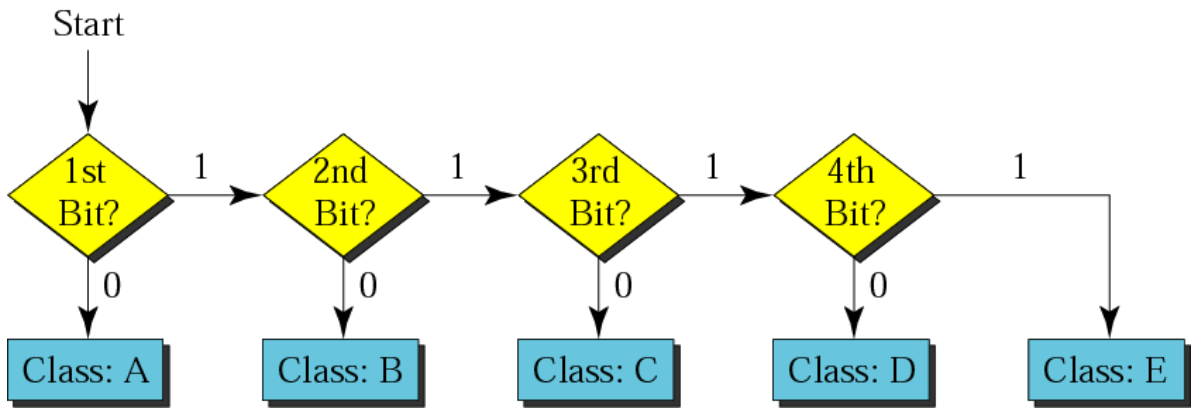
	First byte	Second byte	Third byte	Fourth byte
Class A	**0 to 127**			
Class B	**128 to 191**			
Class C	**192 to 223**			
Class D	**224 to 239**			
Class E	**240 to 255**			

For all classes an address will identify a network (netid) and host (hosted).

The amount of space used for each of these varies with the class:



How does a TCP/IP stack identify the class?

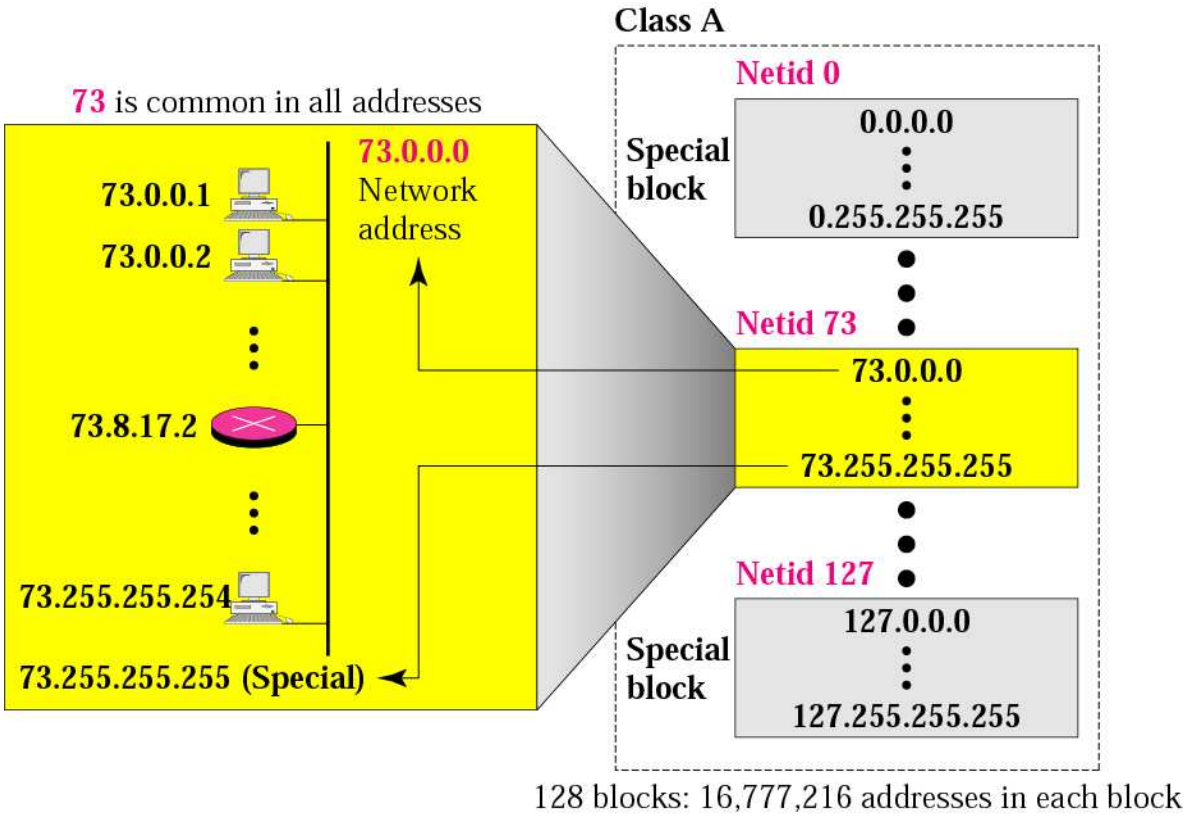


So:

Address space is divided into classes

Each class into blocks:

- Block size varies with class
- First and last blocks of class A are reserved for special purposes, one (10) is for private use leaving 125 allocatable



- B gives 16,384 blocks, 16 reserved for private use leaving 16368 blocks of 65,536 addresses

- Class C gives us 256 reserved for private use blocks, leaving 2,096,902 that can be assigned.

Netmask

In order to route (more soon) we need to be able to figure out the network address and the host address separately.

If we know the address class, we can do this manually –

But beware of sub-netting!

More generally we use a netmask

The netmask is bitwise ANDed with the IP address to give the network address.

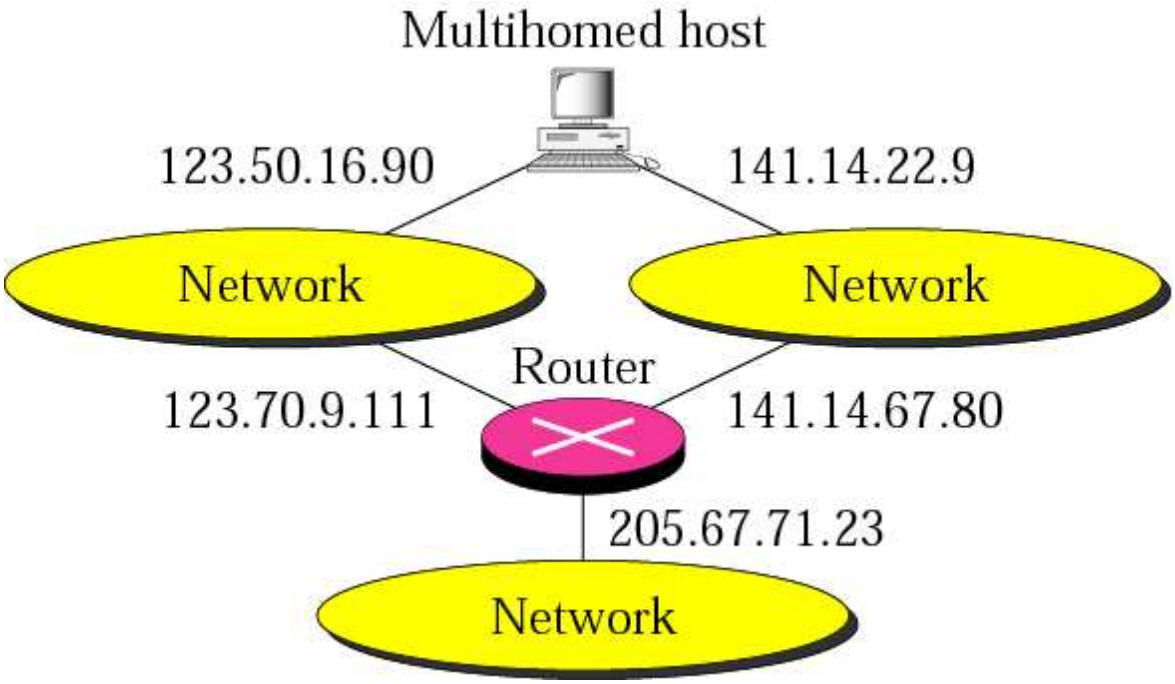
E.g.

Netmask = 255.255.0.0 = 11111111 11111111 00000000 00000000

1's correspond with network, 0's with host.

Multihomed Devices

Routers are by definition, always multi-homed



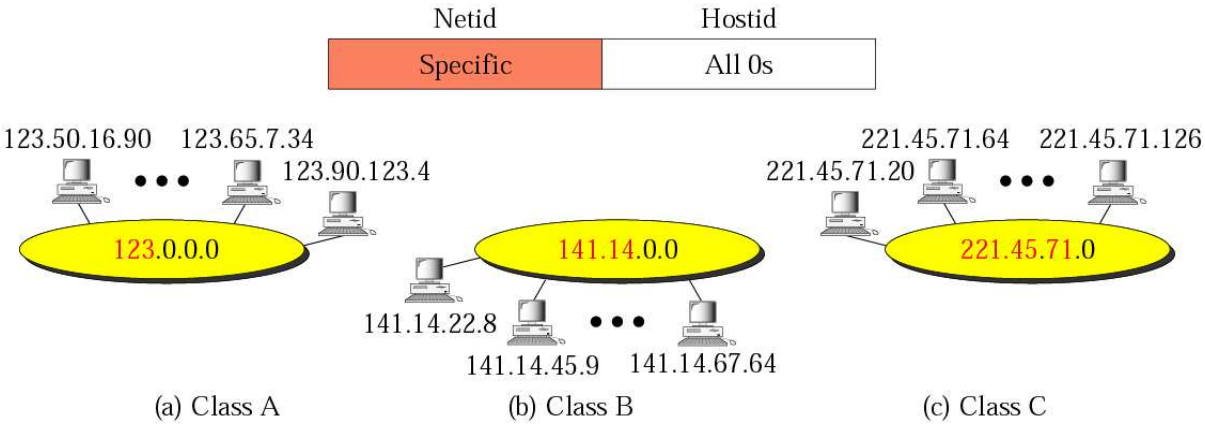
We need to remember that interfaces (e.g. eth0) have IP addresses, NOT computers!

Special Addresses

A number of addresses are reserved for special purposes

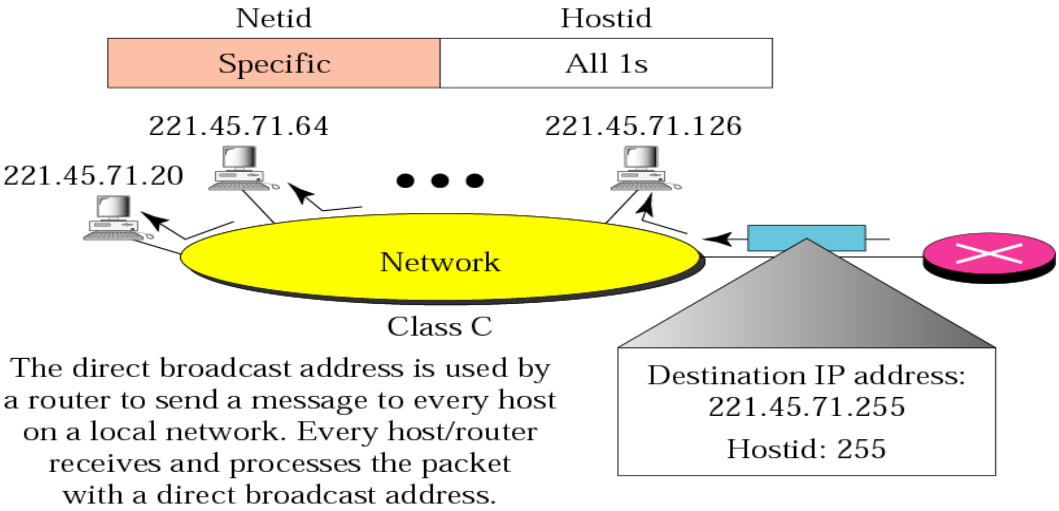
These can be identified by either a special value for the NetID, the HostID, or both.

Network Address



So, the address of a network is for example 164.11.7.0, The NetID being 164.11.7, the HostID being all zeroes.

Direct Broadcast Address

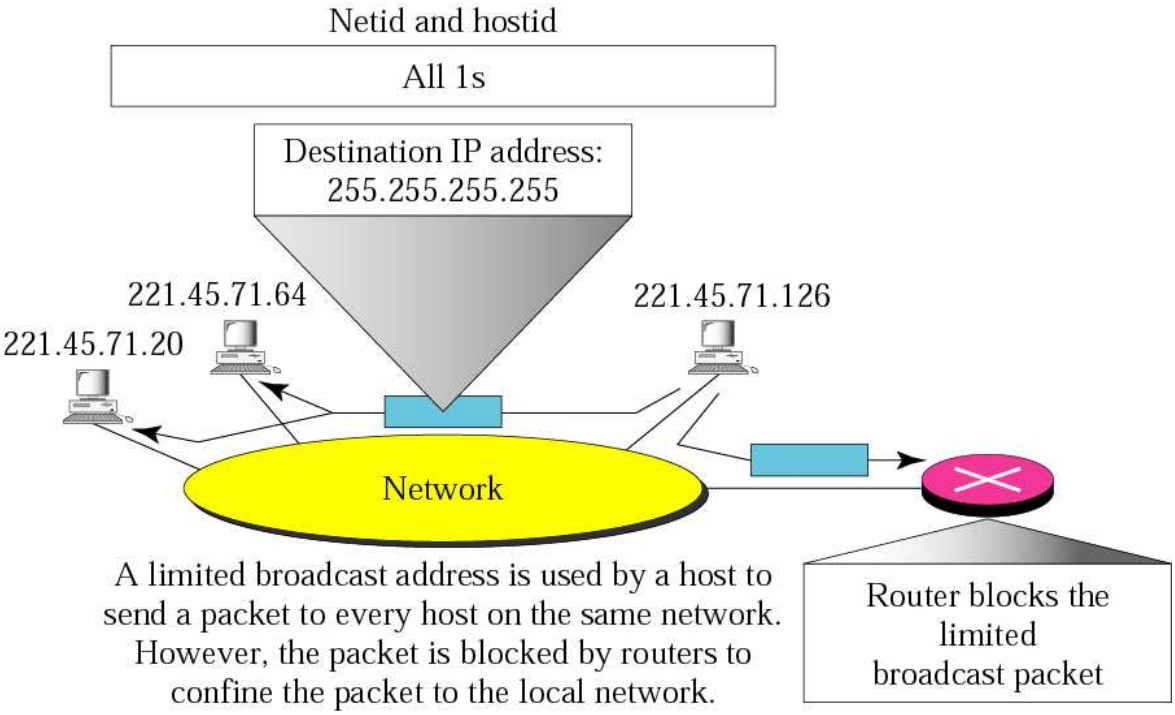


So 164.11.7.255 is the broadcast address for that network 164.11.7.0

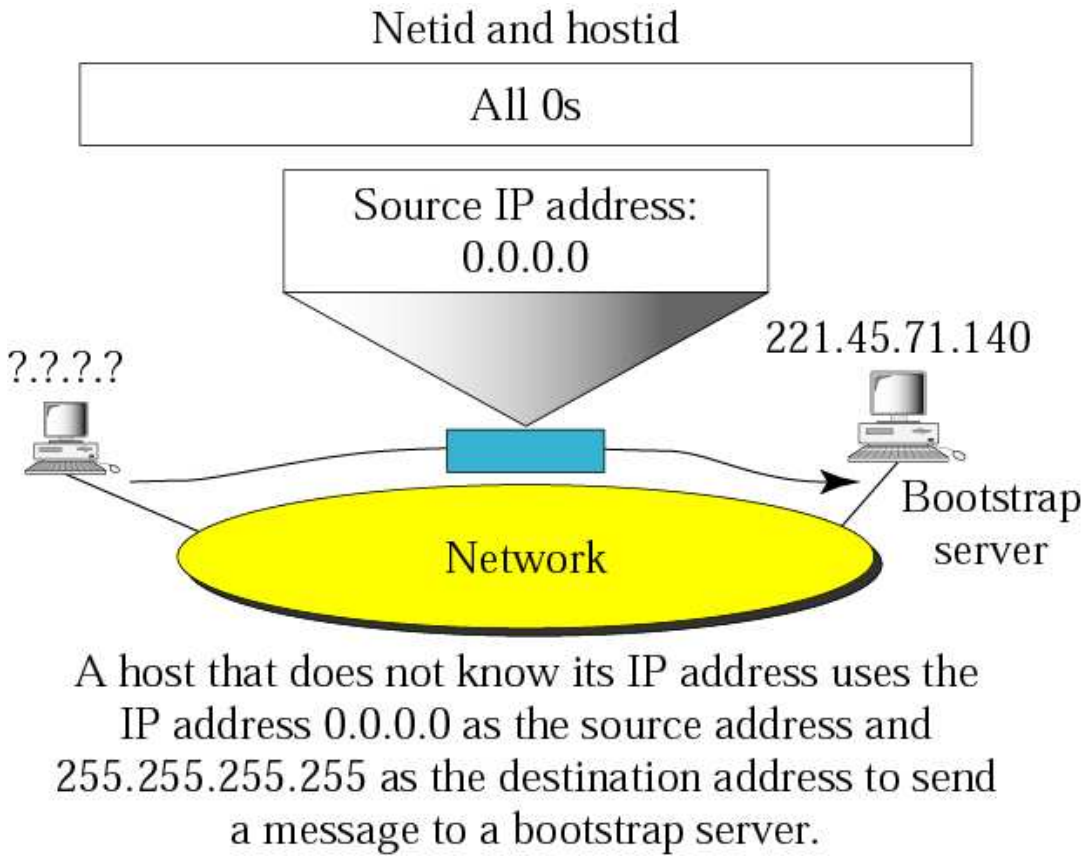
NOTE: Many TCP/IP implementations respond to broadcast ping for example on both the network and direct broadcast address.

Limited Broadcast Address

NOTE: Rarely used, class E address



This Host on this network



NOTE: Class A address

Think about why we need this.

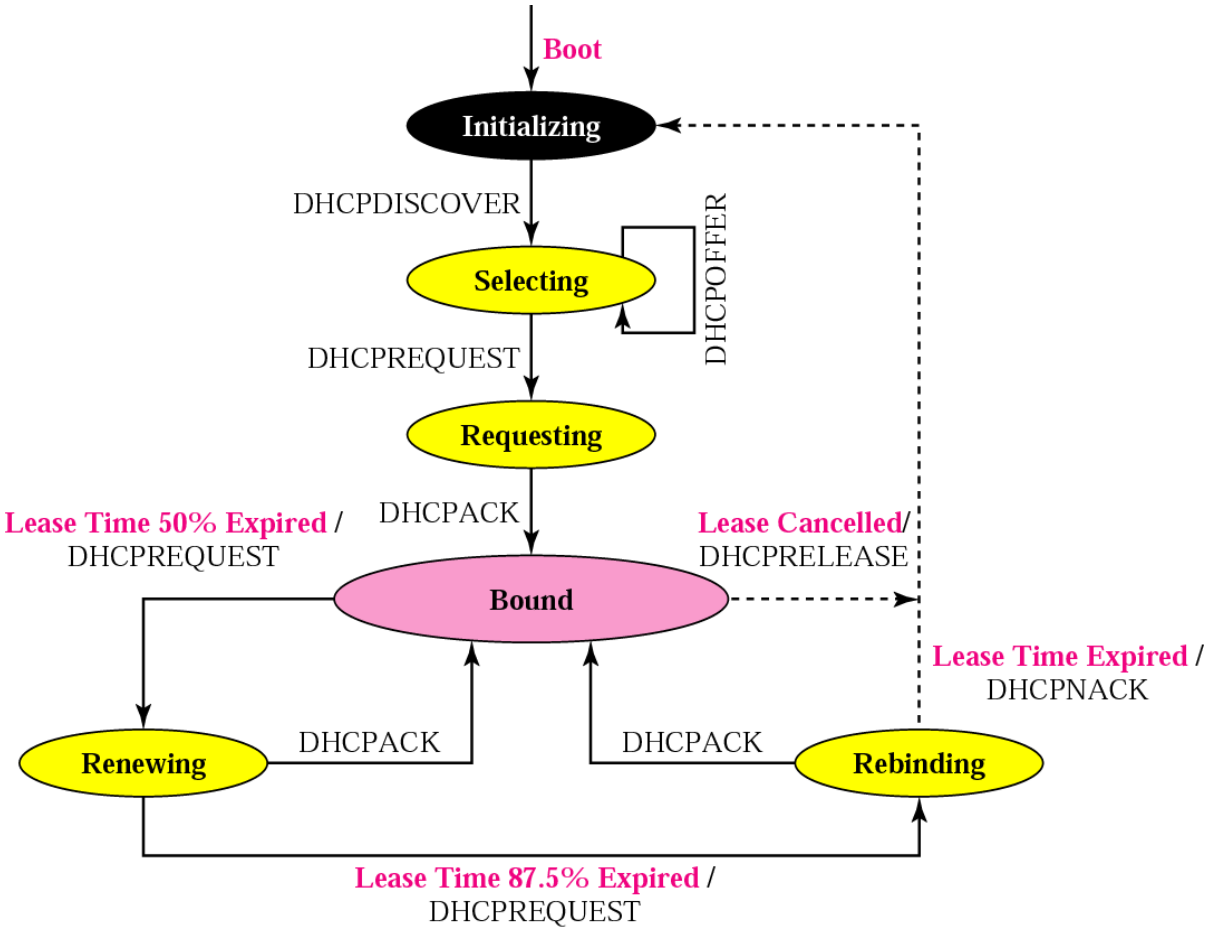
In order to configure the network interface for a host we need to know:

- Its IP address
- Its subnet mask
- IP address of a default router
- IP address of a name server

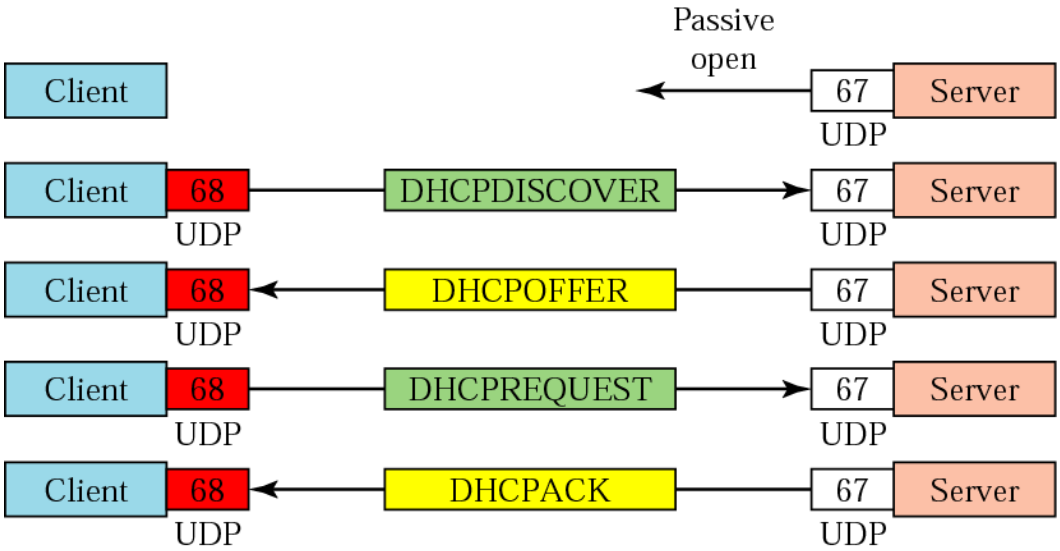
These were originally addressed via a protocol called BOOTP which although still used is now less common than DHCP.

An Introduction to DHCP

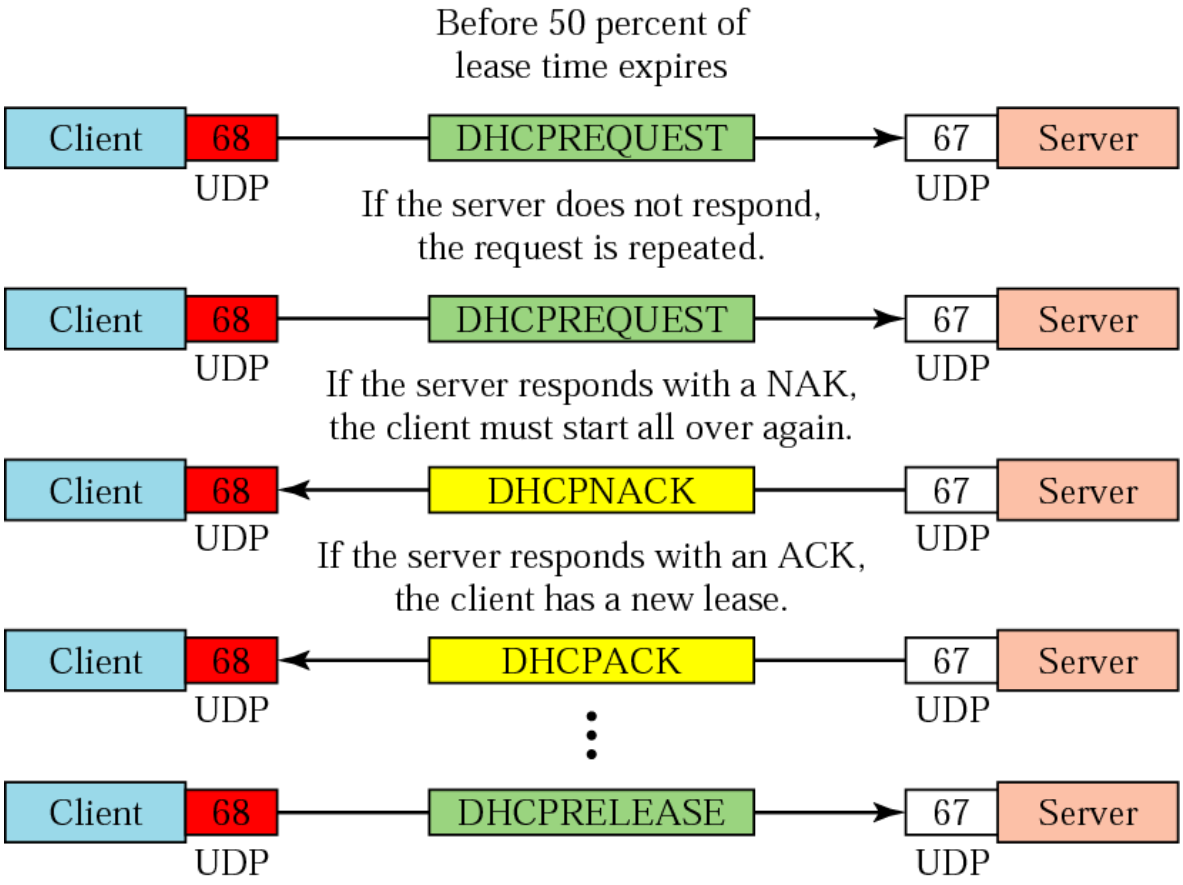
The Process:



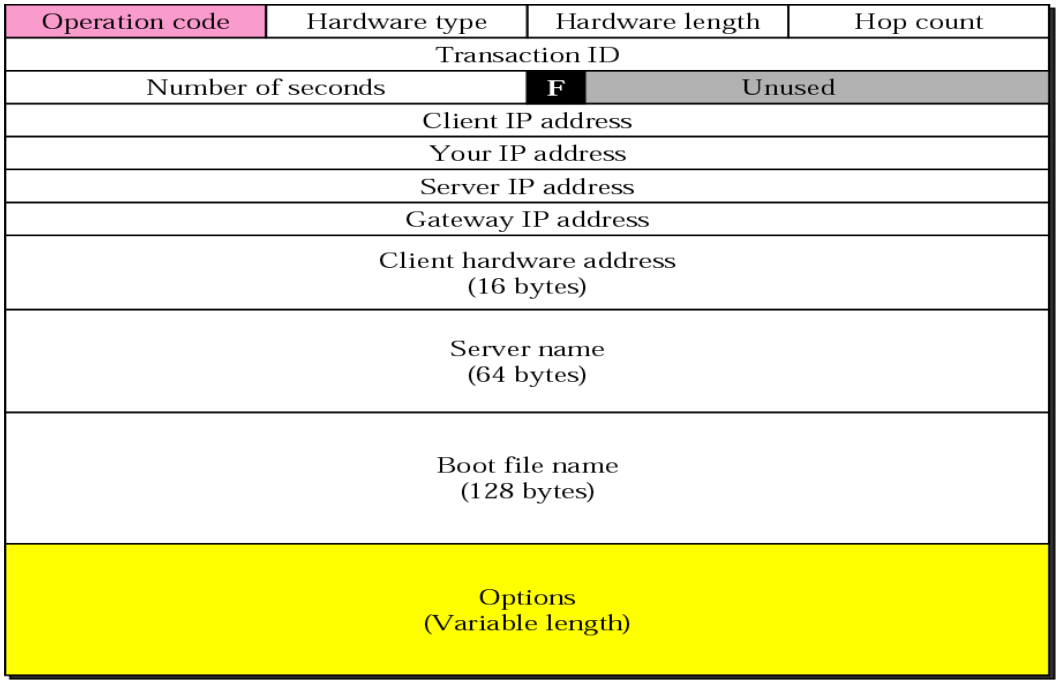
Initial Message Exchange:



Renewal:



Packet Format:



Op code

Request = 1, Reply = 2

Hardware type

For network Ethernet = 1

Hardware length

How big is the physical address, Ethernet = 6

Hop count

Same as TTL for IP

Transaction ID

Set by client, used to match responses to requests

Number of Seconds

When did the client start to boot?

F

Flag set by client to force broadcast rather than unicast by the server

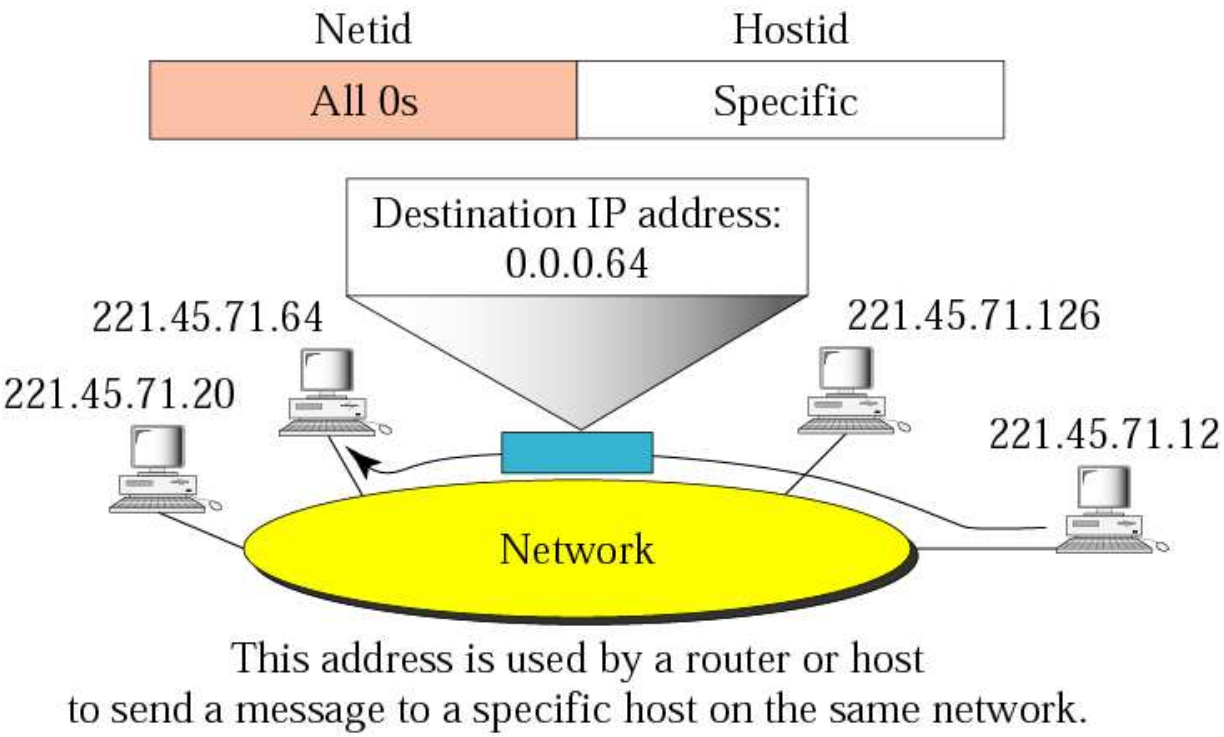
Options

Up to 312 bytes, 1 byte tag, 1 byte length, variable length contents

These include (Tags in brackets – this is a **partial** list)

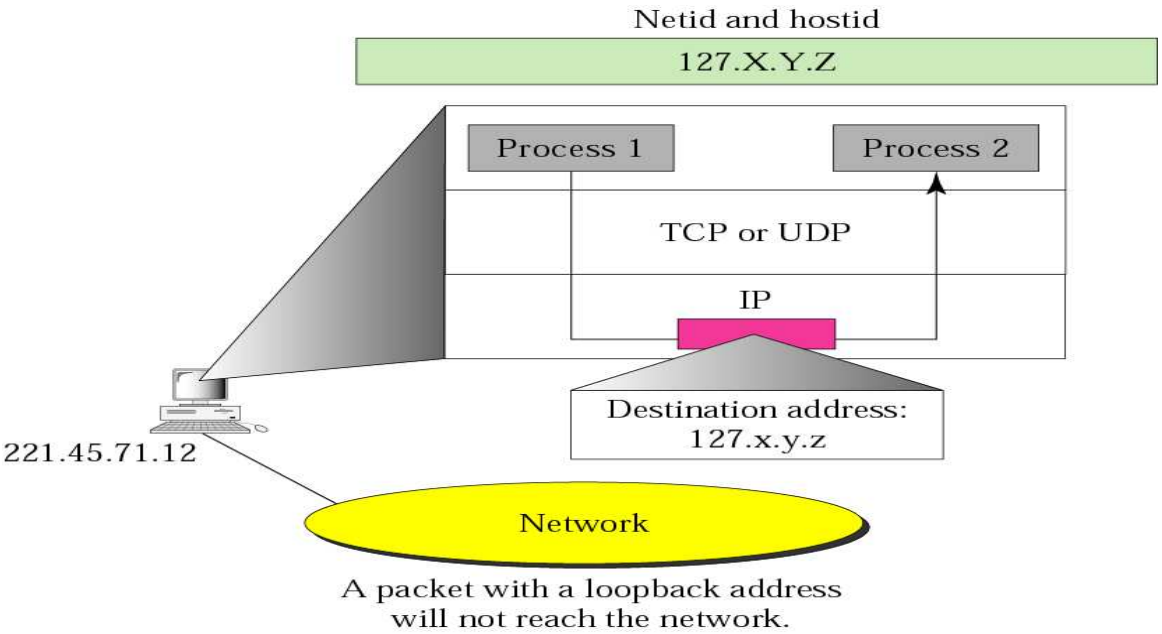
- (1) subnet mask
- (3) default routers
- (4) time servers
- (6) dns servers
- (9) print servers
- (12) hostname
- (53) DHCP interaction, e.g. DISCOVER, ACK, NACK etc.

Specific Local Host



Loopback

Using loopback uses the TCP/IP protocol stack but not any network hardware



Private addresses

Private addresses reduce the need for IP addresses

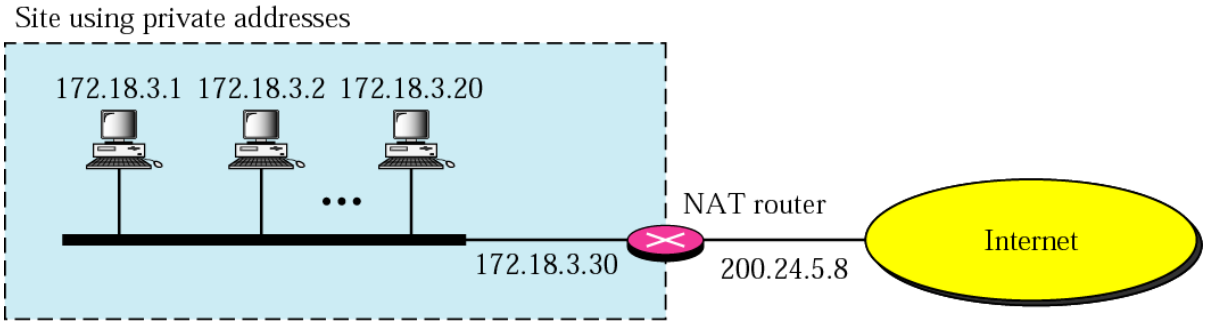
Class A : 10

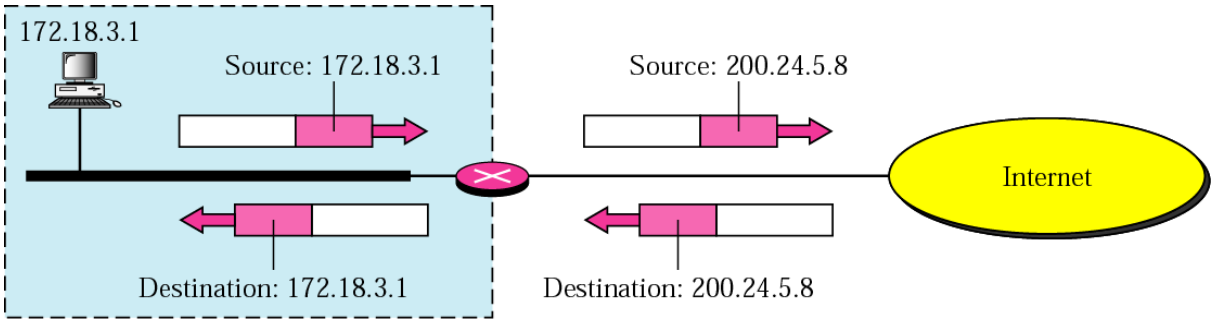
Class B : 172.16 through 172.31

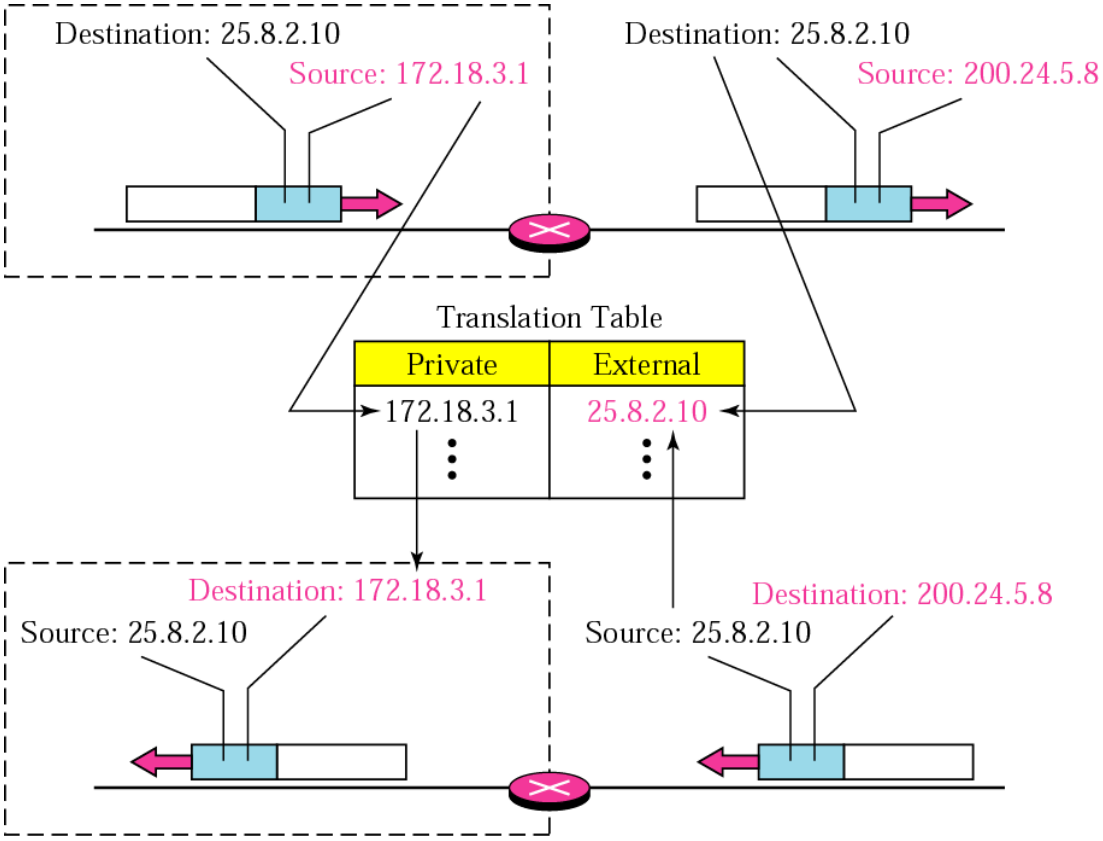
Class C : 192.168.0 through 192.168.255

These are not recognised globally, SHOULD NOT appear on public networks, and may be dropped or blocked by routers.

They do however lead to networks with a single visible (public) IP address using Network address translation (NAT)/ IP Masquarading







This is very much a gross simplification!

Using this system, no external host could initiate contact with any host in the private network.

Most NAT systems are more complex (significantly so for Linux IP-masquerading

Think:

- How can we contact an interior host?
- What happens if two web browsers on the same computer want to contact *google*?
- What happens to client programs that are also servers? (e.g. most peer to peer systems such as bittorrent)

Generally,

- Private IP
- Private Port
- Destination IP
- Destinition Port
- Protocol

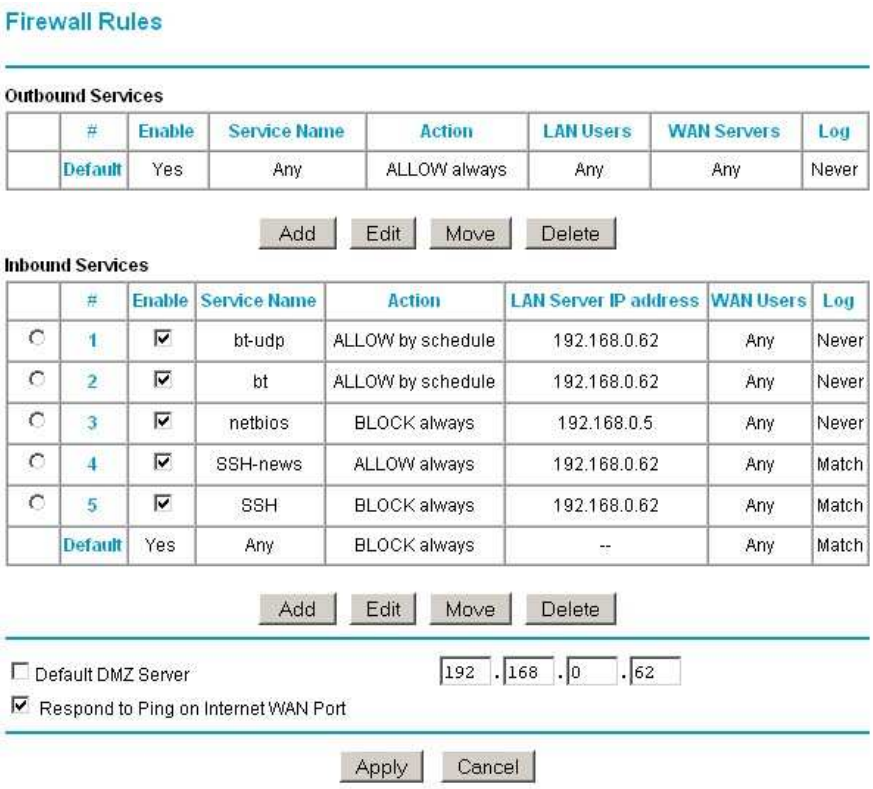
are maintained.

Often the private port will be rewritten, creating a unique ID for the connection which can also be used as an index.

Often protocols such as UPNP are used to reconfigure the router by internal hosts.

Often routers support allowing incoming connections to specific ports to be redirected to a specific internal host

Often a router will support a DMZ (not strictly correctly named) – a default host for incoming connections.



Firewall Rules

Outbound Services

	#	Enable	Service Name	Action	LAN Users	WAN Servers	Log
	Default	Yes	Any	ALLOW always	Any	Any	Never

Add Edit Move Delete

Inbound Services

	#	Enable	Service Name	Action	LAN Server IP address	WAN Users	Log
○	1	☑	bt-udp	ALLOW by schedule	192.168.0.62	Any	Never
○	2	☑	bt	ALLOW by schedule	192.168.0.62	Any	Never
○	3	☑	netbios	BLOCK always	192.168.0.5	Any	Never
○	4	☑	SSH-news	ALLOW always	192.168.0.62	Any	Match
○	5	☑	SSH	BLOCK always	192.168.0.62	Any	Match
	Default	Yes	Any	BLOCK always	--	Any	Match

Add Edit Move Delete

☐ Default DMZ Server 192 . 168 . 0 . 62

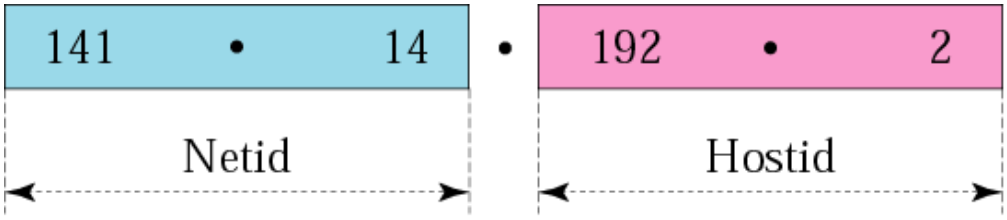
☑ Respond to Ping on Internet WAN Port

Apply Cancel

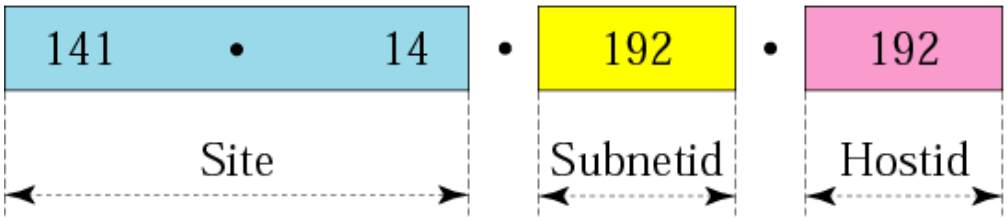
Subnetting

Often the class of address, (or in modern systems the CIDR block), we have been allocated may not give us the flexibility we require.

We can gain more flexibility by using non-standard subnetting.



a. Without subnetting



b. With subnetting

E.g.

UWE has been allocated the 164.11.0.0 network

What class is this? How do you know?

Jaz machines in 3P27 are on 164.11.7.0

Non-Jaz machines are on 164.11.222.0

What is being done here is we are dividing the address into 3 components:

- The site (network id)
- The subnet
- The host

The network address is the site + the subnet. We can find this by ANDing our subnet mask with the IP address.

In short subnetting adds more 1's to the netmask.

If we wish to divide a class B address into class C subnets, deriving the subnet mask is easy.

Others work exactly the same, but because we generally use decimal values are more complex.

Imagine I wish to subnet 192.168.0.0 into 8 subnets.

My subnet mask is:

11111111 11111111 11111111 11100000

In decimal?

255.255.255.224

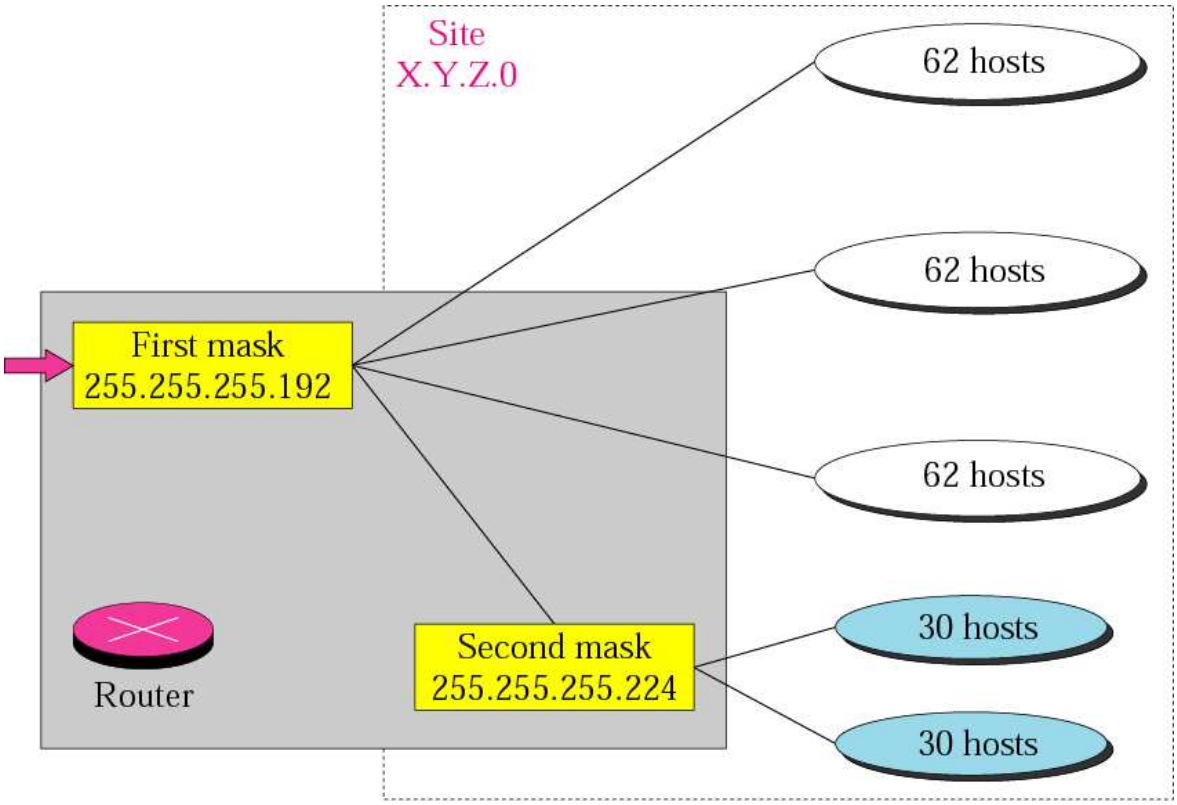
How many machines per network?

2^5-2 = 30.

First subnet	192.168.0.0 to 192.168.0.31
Second	192.168.0.32 to 192.168.0.63

and so on…

Variable Length subnetting:



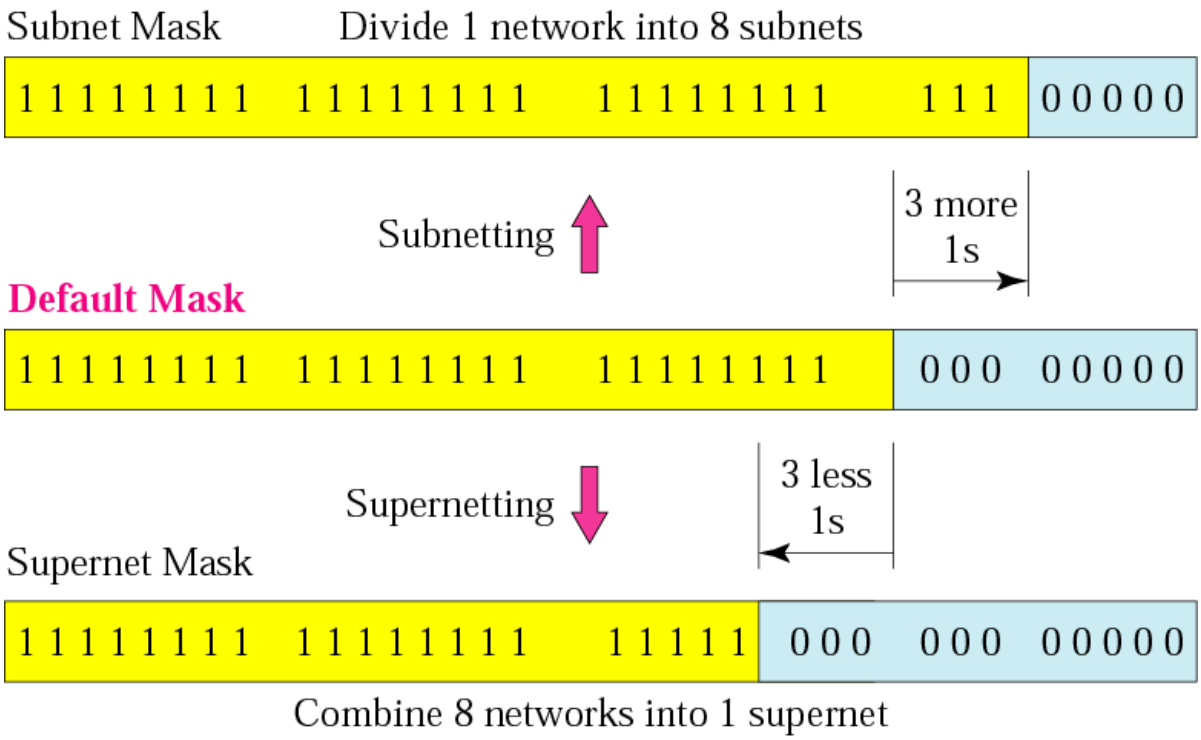
Specialised routers first apply one mask, and then a second to create a more flexible network configuration.

Supernetting

The opposite of subnetting.

Rules:

- The number of blocks is a power of 2
- The blocks are contiguous
- Byte three of the first address is divisible by the number of blocks


Subnet Mask
Divide 1 network into 8 subnets
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0
Subnetting
3 more 1s
Default Mask
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
Supernetting
3 less 1s
Supernet Mask
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
Combine 8 networks into 1 supernet

IP Routing

The problem:

How do we get an IP datagram from Host ‘A’ to Host ‘B’?

Host A & B on the same network (DIRECT routing)

Translate IP address to hardware address (e.g. Ethernet MAC address)

Send!

How do we find the hardware (data link) address?

Consider Ethernet (other hardware may vary)

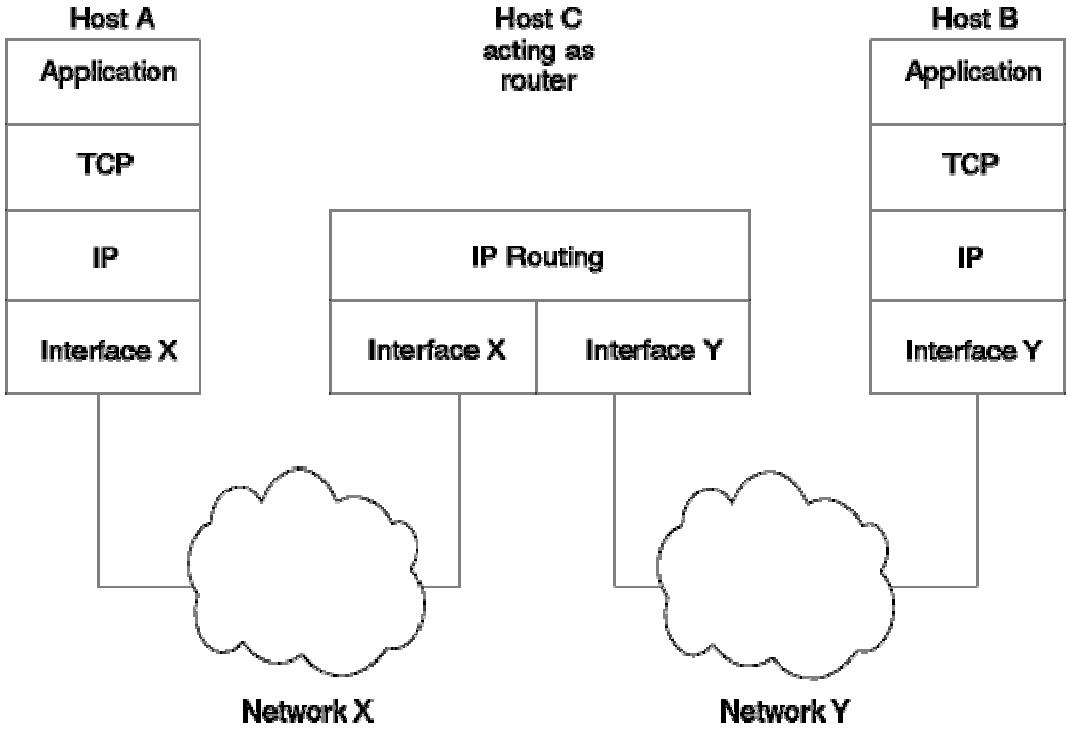
Remember however that ARP/RARP are specified by the IP layer.

Address Resolution Protocol (ARP – RFC 826)

- Look in ARP cache,
- If not present – broadcast “who owns this IP-number? ”
- Owner responds, forward datagram, update cache

RARP also exists to allow hosts to discover their own IP address.

Host A & B on different networks (INDIRECT routing)



Here Host A needs to know that the route to network Y is via C.
This is ALL that A needs to know.

C needs to know that B is reachable via network Y and that network Y is attached to interface Y.

A encapsulates & sends the datagram to C, C unwraps, processes (including decreasing the TTL), encapsulates & sends the datagram to B.

Options within the IP header however allow us to specify the exact route to our destination (strict source routing) or an approximate route (loose source routing)

How does the host A know to send via C?

All hosts/routers maintain a routing table

Static Routing – Routing table created manually by root

E.g.

```
route add –net 164.11.7.0 netmask
          255.255.255.0  metric 0 eth0
route add default gw 164.11.7.1 metric 0
```

Good for Simple/Stable networks

Cannot respond to changes in the network topology

Dynamic Routing – Routing table evolves dynamically under the control of a routing daemon (e.g. routed)

Requires a *Routing Protocol*

Simplest (but weak) method – RIP RFC 1058 (an enhanced version RIP-2 also exists)

RIP is essentially obsolete, having generally been replaced with OSPF (RFC 1247)

RIP

Vector/Distance based (router : network : hop count), using UDP on port 520

All hosts start with their directly connected routes.

Hosts are passive, Routers active.

Routing tables are maintained on each host/router

Routers broadcast their routes at intervals.

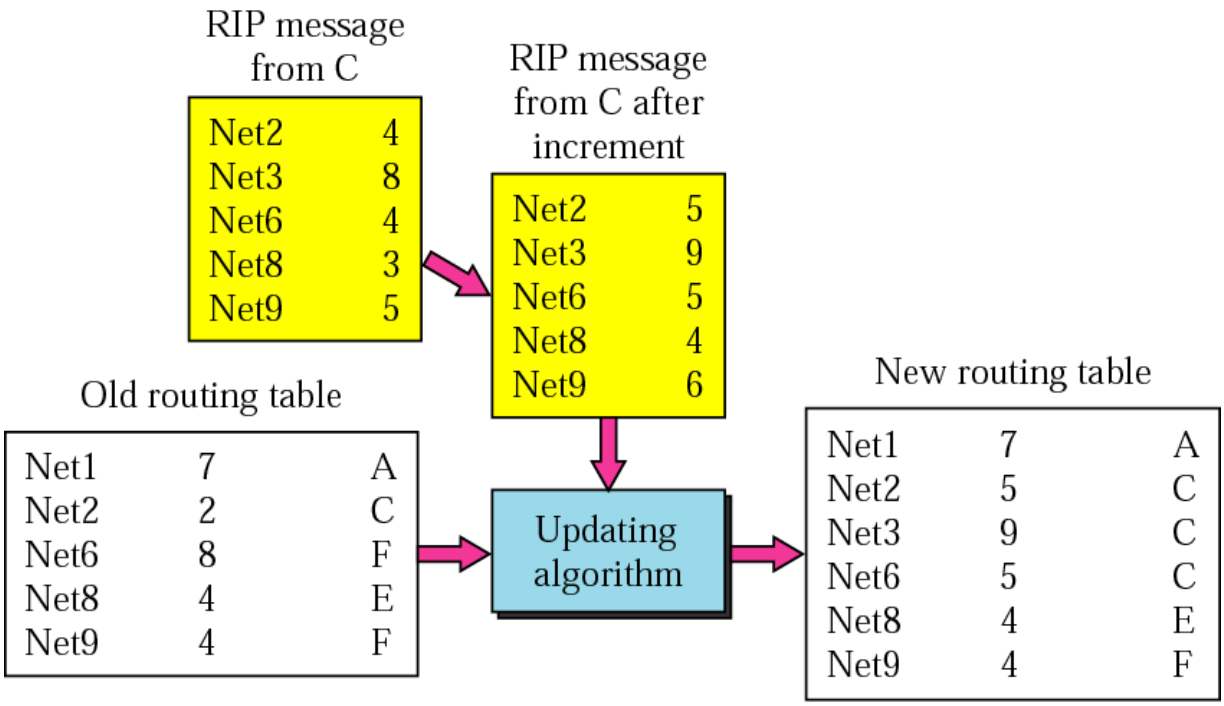
All systems compare these routes to their own routing tables and update if:

There are routes to new networks
There are shorter routes
A route is reported as unreachable

Routes are timed out if not reported after 6 update cycles

Routers broadcast route changes as they occur

Each host/router therefore "grows" a table of which router to use for each known network together with a default route for all others.



Net1: No news, do not change
Net2: Same next hop, replace
Net3: A new router, add
Net6: Different next hop, new hop count smaller, replace
Net8: Different next hop, new hop count the same, do not change
Net9: Different next hop, new hop count larger, do not change

RIP does however have many weaknesses

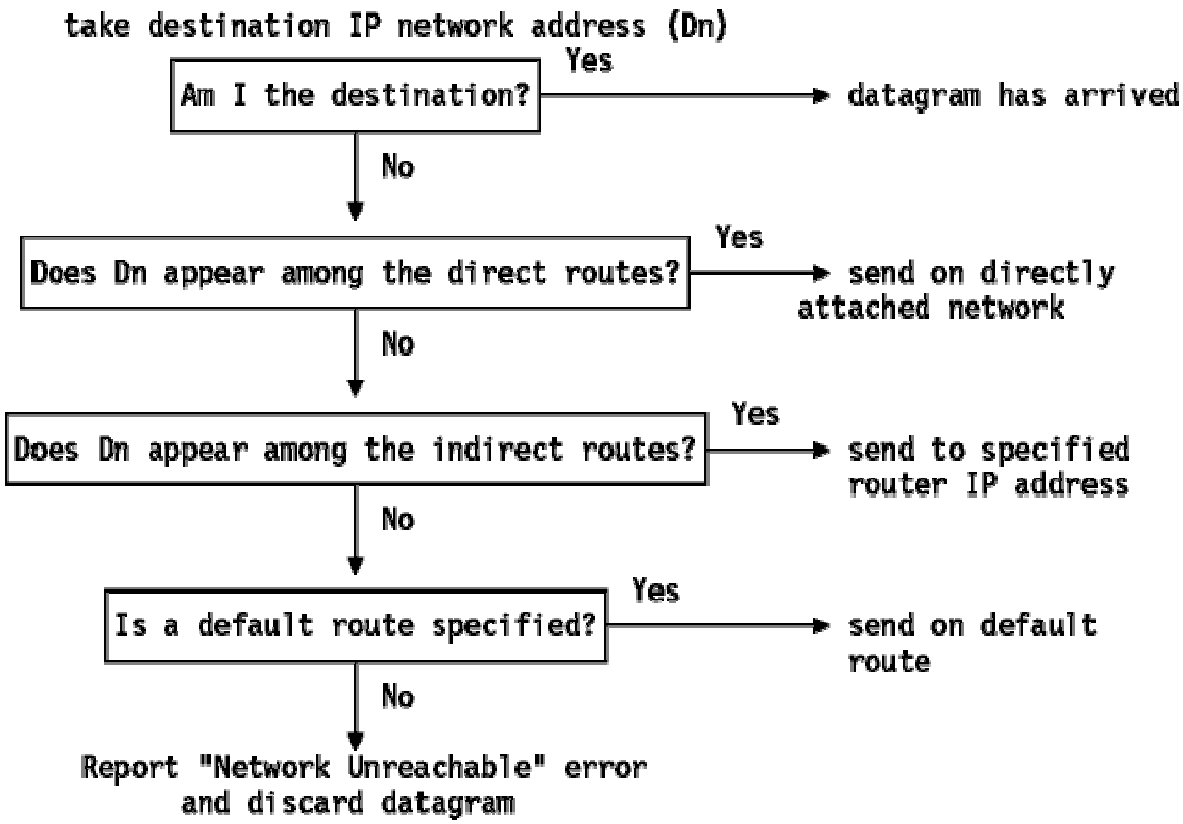
- Slow convergence

    - If routers report in every 30 seconds, a 10 hop propagation will take 150 seconds on average.

- Instability
    - Router X routes to network A
    - Router Y routes to network B
    - network A fails
    - Router X updates its table to show an infinite route to network A
    - Router Y (which uses router X to get to network A)
    - Router Y broadcasts its table before Router X
    - Router X assumes router Y has a “better” route to A

    Chaos until router Y gets the correct idea!

The basic routing algorithm

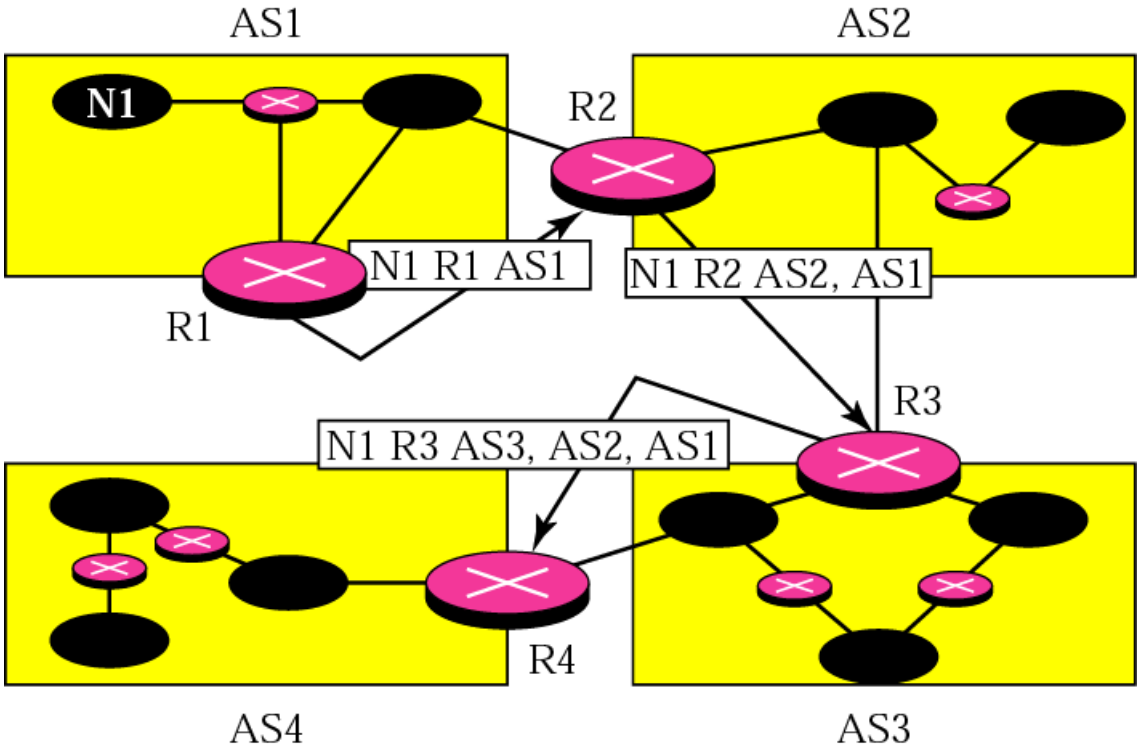


Generally, hosts **will** have a default route

So far we have only considered routing within a collection of networks under a single point of control (Interior Routing). These collections of networks are known as Autonomous Systems (AS).

How do we route between different AS's? – Exterior Protocols

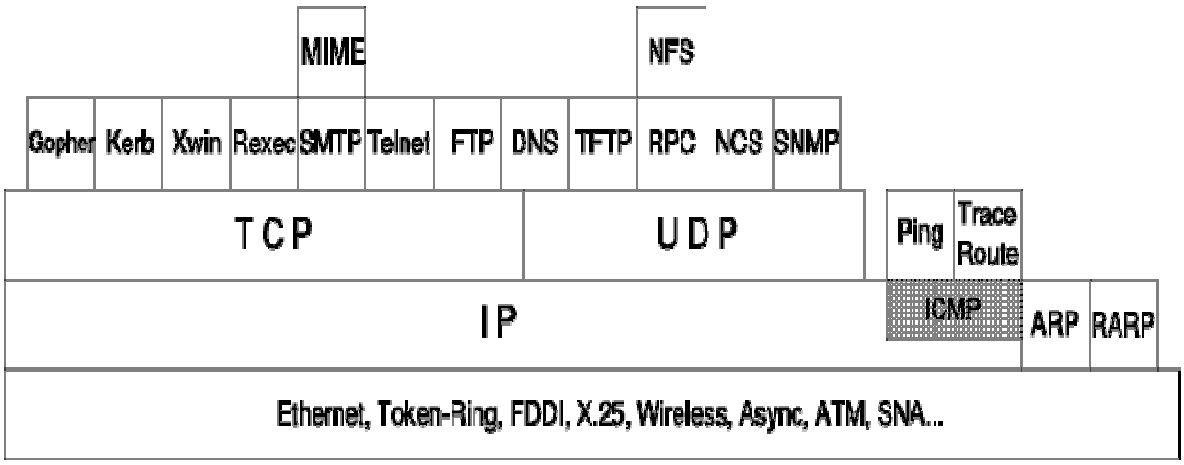
E.g. Border Gateway Protocol (BGP) – policy based routing.
Most systems never run these!



Routing Control

We've discussed how the IP manages routing, but how do systems "discover" errors?

ICMP (RFC 792 updated in RFC 950)



ICMP uses IP as if it were a higher level protocol **BUT** is a required component of the IP protocol.

ICMP messages:

REDIRECT messages are sent by a router to advise a better route.

UNREACHABLE messages inform the initiator that delivery is not possible.

ECHO requests generate an echo reply if the host is reachable. (Used by ping)

TIME EXCEEDED messages are generated by a router which receives an IP datagram with TTL of 1 or less. (Used by traceroute)

Summary:

- IP routing is performed by network. We want to reach the network to which a host is attached.

- Direct routing involves passing datagrams to hosts connected to the same network. This commonly involves using ARP.

- Indirect routing involves passing datagrams to a connected router, which then deals with forwarding it again itself.

- Interior routing is within an organisation (AS). If performed dynamically, routing protocol is usually RIP, RIP2 or OSPF. Here efficiency is the aim.

- Exterior routing uses different routing protocols. Current standard(s) are BGP or CIDR. Here socio-political & financial issues come into play.

- ICMP messages (amongst other things) let hosts & routers discover whether networks and hosts are reachable, improve routing, and are used by the ping and traceroute commands.

Network Hardware

So far we've traced our way down a TCP/IP stack, the only thing left is the subnet layer (not defined in TCP/IP)

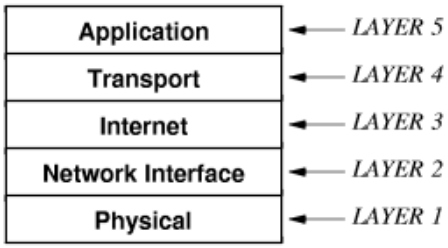
This is the hardware (e.g. Ethernet/Token Ring/ATM/FDDI etc.) that our network runs on.

We're not going to look in any detail at this layer, we do need to do some thinking about hardware when it comes to the practical issue of actually constructing a network.

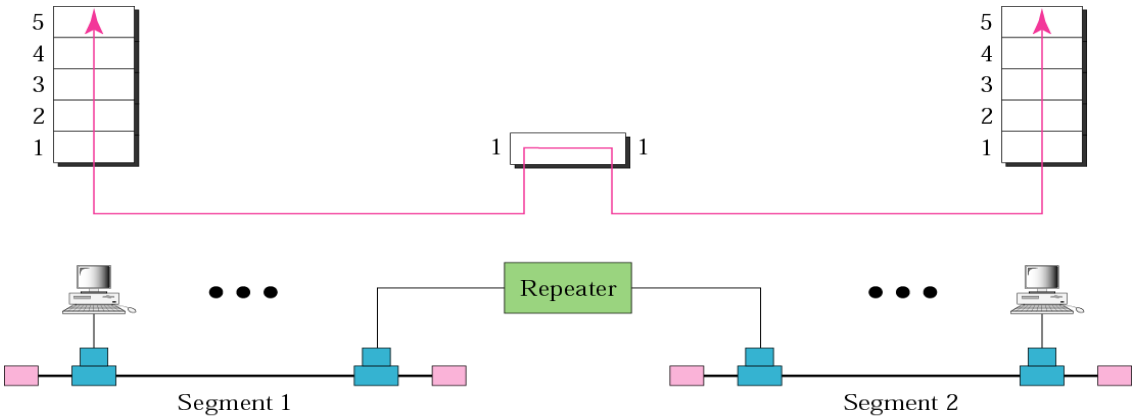
Connecting Networks

- The internet as the name suggests is a collection of interconnected networks.
- Networks themselves need to have devices connected to them.
- How this is achieved impacts:
    - Performance
    - Network Topology
    - Security
- Different types of devices operate at different levels within our network:

Lets revisit our TCP/IP stack:



Repeaters

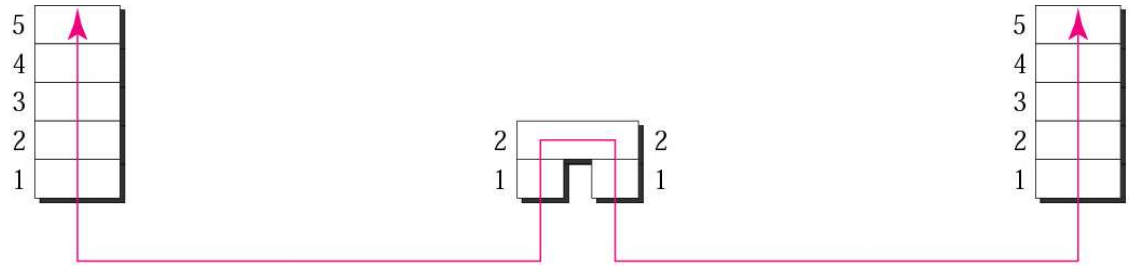


- Repeaters are the simplest form of interconnect.
- They connect segments of a LAN together
- Can be used to extend 10base5 ethernets beyond 500 metres
- Operate in the physical layer

Hubs

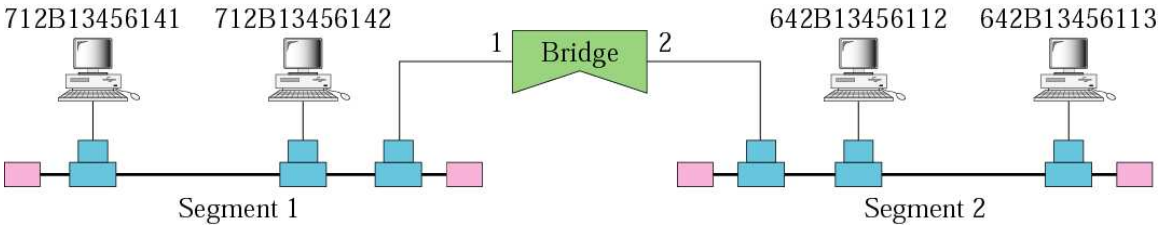
- Hubs are probably the most common LAN interconnect
- Technically multi-port repeaters
- Allow us to build physical hierarchical star topologies whilst maintaining a logical bus topology

Bridges



Address	Interface
712B13456141	1
712B13456142	1
642B13456112	2
642B13456113	2

Bridge table



- Bridges operate in both the physical and data link layer
- Bridges can be thought of as filtering repeaters
- Can filter at the MAC level
- LAN technology
- Modern bridges are often transparent (a.k.a. learning)
    - No need to create filter tables manually.

Routers

- Routers operate at the IP (level 3) level
- We've already discussed routing
- Routers connect networks together
- Routers are usually "visible" on our network

Switches

- Switches are fast gaining in popularity for both speed and security reasons.
- Layer 2 switch
    - Multi-segment bridge
    - Can filter
    - Each station on its own segment, hence no collision risk

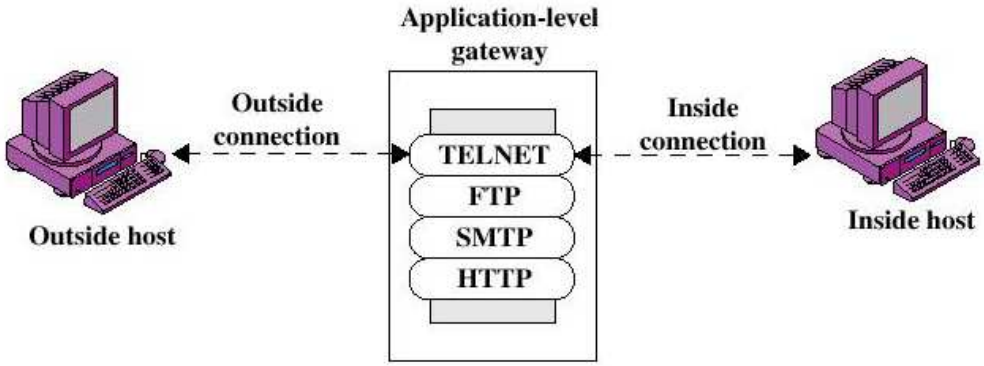
- Layer 3 switch
    - These are really routers!
    - Only tend to be described as such by salesdroids

Firewalls (see Cheswick & Bellovin)

1st edition is a bit dated (but still good) and is available at:
http://www.wilyhacker.com/1e/

- Firewalls exist to protect a network
- To be effective:
    - All traffic in or out of the protected network goes through the firewall
    - Only traffic permitted by the security policy is allowed through
    - The firewall is secure!
- Firewalls provide:
    - Service control via port/address or proxy
    - Direction control
    - User control – should this user do this?
    - Behaviour control: spam, viruses etc.
- At their simplest Firewalls are packet filtering routers
    - Port/address based control only
    - Cannot detect spam/viruses etc.

- More sophisticated firewalls are application relays
    - E.g. Mail
    - A.k.a. Application-level Gateway or Proxy server



- Another type of firewall is a circuit level gateway
    - Often circuit based for outgoing, proxy based for incoming.
    - Best known example – SOCKS

I Know I'm Paranoid, But am I paranoid enough?

Security is a process, not an event!

Policy, Mechanisms, Assurance, Response and Education

Who are you trying to protect what from?