UQC109S2
Computer Networks & Operating Systems

Ian Johnson (Networks – term 1)
Nigel Gunton (Module leader O.S. – term 2)

Room 3P16
Telephone: extension 3167

Email: Ian.Johnson@uwe.ac.uk
Nigel.Gunton@uwe.ac.uk

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

Lab Tutors:

John Counsell : **Groups 5, 3, 7**
Graham Darke : **Groups 1, 6, 8**
Nigel Gunton : **Group 2**
Ian Johnson : **Group 4**

All labs are held in 3P27

Dual boot Windows/Linux PC's
2 Gb Jaz drive equipped systems

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!

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

Website for the networks text

Search engines, learn to use them effectively

Linux Documentation Project

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

RFC's

2. Dead Trees

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

The library!

Course Texts:

"*Operating Systems with Linux*",
O'Gorman, Palgrave

"*Computer Networks & Internets*",
Comer, Prentice Hall, 2nd Ed.

3. Worksheets

These are intended to help and assist! Do them!

We're starting with installing slackware linux - (worksheet from Ian's web page, logbook from Nigels)

WARNING!

The University takes allegations of collusion and plagiarism very seriously.

If you copy or extract material, make this very clear!

Acknowledge always, e.g.

This function is based on source code obtained from … (in a comment)

Cite sources

The university standard is Harvard referencing (see Nigel's web page)

Highlight the material is not your own!

According to X ….

X (2003) argues "…"

Indentation, quotation marks, italics, references

What is this module about?

Networks & Operating Systems ☺

2 inter-related topics, e.g.

- **Network administration**
- **Security**
- **Does *ifconfig* belong under Networks or OS?**

Lab Sessions and lectures will not synchronise!

To start, installing slackware linux on the Jaz drives working through to the network tools worksheet.

You will <u>NEED</u> to program in C!

Networking Topics

- Introduction
  - 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 Security (chap 40 Comer)

- Network equipment: routers, bridges, hubs switches

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.

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

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:

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.

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.

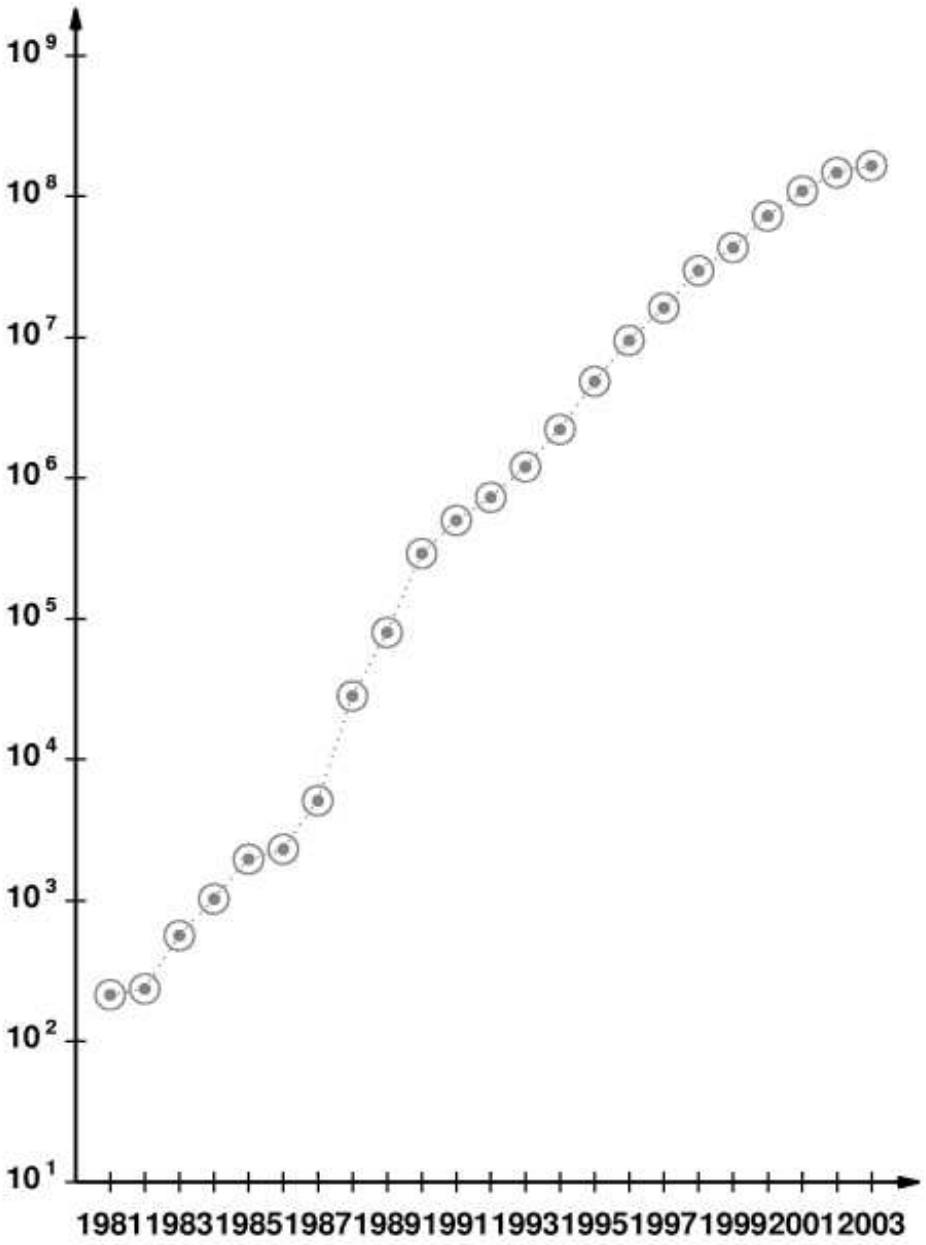
1991 **Explosion!**

- 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.

1993

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



From Comer, D. op. cit. p.10. Internet Growth plotted on a $\log_{10}$ scale to demonstrate approximately 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

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. if 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.

Traceroute

Traceroute gives us information on a route packets may be 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.

Tutorial support is for C only!

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, (most peer-to-peer systems are more complex).

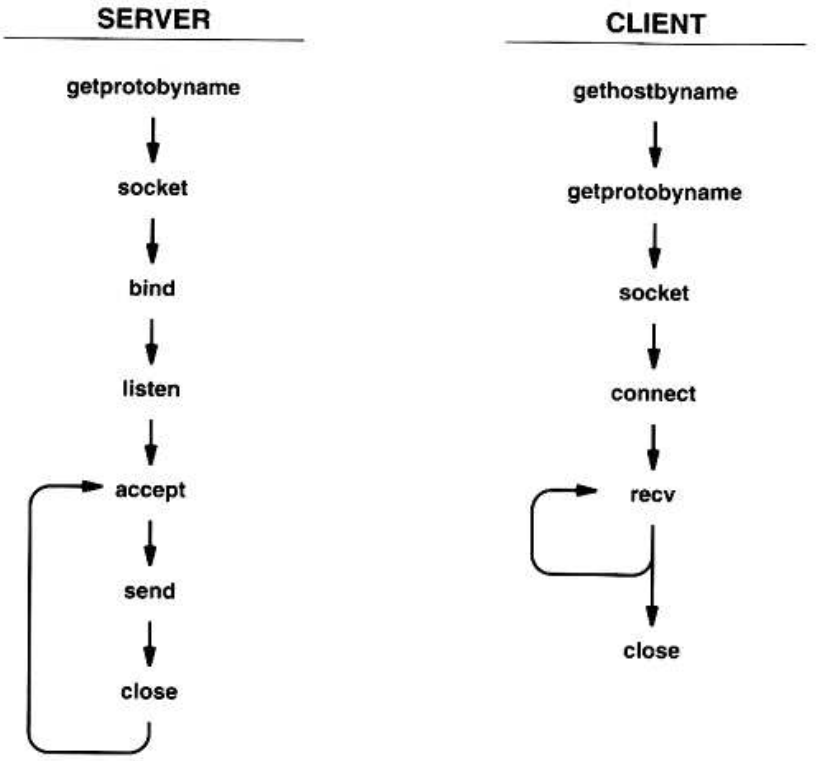
The basic idea:

- 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.

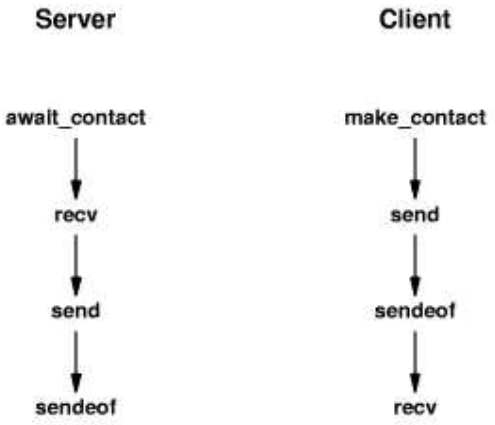
Comer API (*Comer op. cit.*) :

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

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;
}
```

A small Java client program (more limited):

```
// tcpClient.java by fpont 3/2000
// usage : java tcpClient <server> <port>
// default port is 1500

import java.net.*;
import java.io.*;

public class tcpClient {

    public static void main(String[] args) {
      int port = 1500;
      String server = "localhost";
      Socket socket = null;
      String lineToBeSent;
      BufferedReader input;
      PrintWriter output;
      int ERROR = 1;

      // read arguments
      if(args.length == 2) {
          server = args[0];
          try {
            port = Integer.parseInt(args[1]);
          }
          catch (Exception e) {
            System.out.println("server port = 1500 (default)");
            port = 1500;
          }
      }


// connect to server
      try {
          socket = new Socket(server, port);
          System.out.println("Connected with server " +
                          socket.getInetAddress() +
                          ":" + socket.getPort());
      }
      catch (UnknownHostException e) {
          System.out.println(e);
          System.exit(ERROR);
      }
      catch (IOException e) {
          System.out.println(e);
          System.exit(ERROR);
      }
```

```
try {
      input = new BufferedReader(new InputStreamReader(System.in));
      output = new PrintWriter(socket.getOutputStream(),true);

          // get user input and transmit it to server
          while(true) {
            lineToBeSent = input.readLine();
            // stop if input line is "."
            if(lineToBeSent.equals(".")) break;
            output.println(lineToBeSent);
          }
      }
      catch (IOException e) {
          System.out.println(e);
      }

      try {
          socket.close();
      }
      catch (IOException e) {
          System.out.println(e);
      }
    }
}
```

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

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

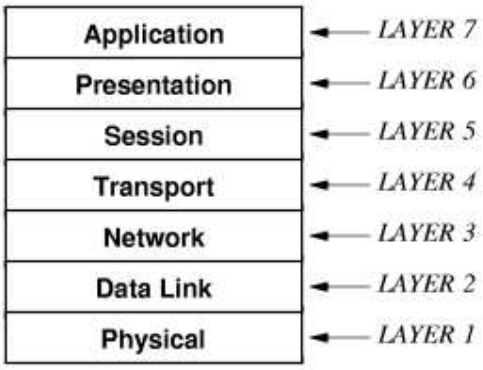
Layered Models

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. These are here.

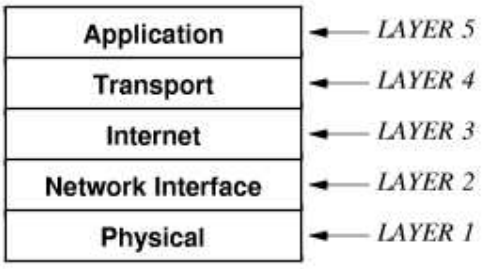
How does TCP/IP fit into this model?

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



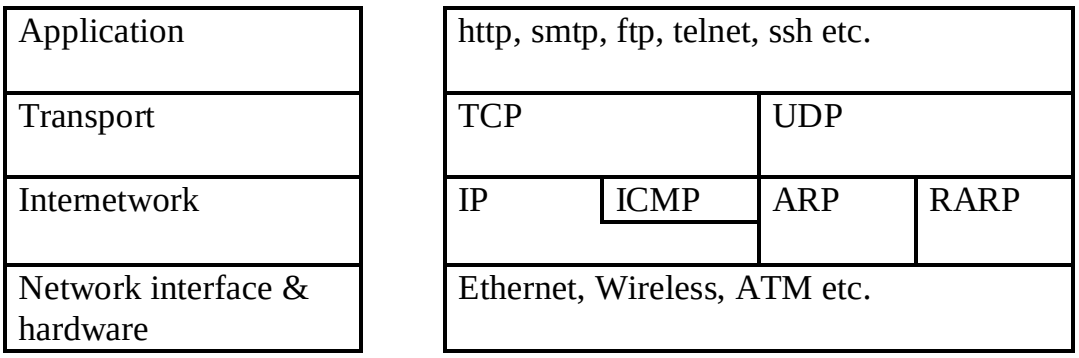
Stacks

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 (URN formerly URL):
   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.

Protocol is fully defined by the RFC