The TCP Protocol Stack

Michael Brockway

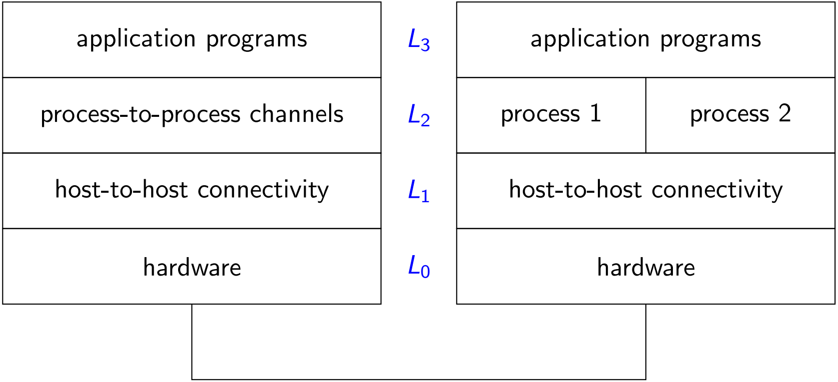
February 16, 2018

Introduction - Layered archtecture

Networking software is desgined in a layered fashion

- The bottom layer is the services offered by the underlying hardware and their device drivers: 'Ethernet', 'Wifi', 'BlueTooth', 'Zigbee' all employ electronics to get digital signals from one computing *host* to another.
- The next layer is software that employs the lower layer functionality to provide a higher (more abstract) level of service.
- There will be a sequence of layers each employing the services of the layer below.

Example



- The host-to-host connectivity software employs the hardware and its device-drivers to send data to another host.
- The software driving a process uses this to exchange data with a peer process on another host.
- Application software uses process-to-process service software to exchange data with another app on another host.

Layered architecture

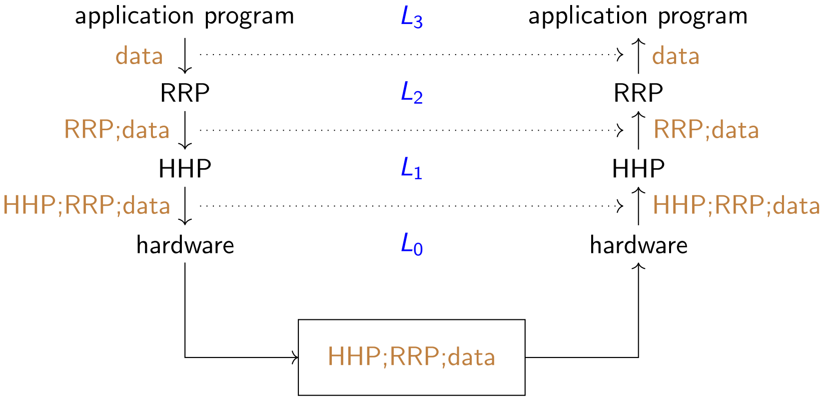
Each layer implements one (or more) protocol.

Each protocol defines

- a *service interface*: in later L_i, defines operations provided by this protocol for layer L_{i+1}
- a *peer-to-peer interface*: defines the messages exchanged with a *peer* in layer L_i.
- At the hardware level L_0 peer-to-peer communication is directly over a link;
- at a higher level L_i, L_i to L_i communication is *conceptual*; in reality it happens by L_i making use of services of L_{i-1} which uses services of L_{i-2} and so on down to L_0.

Example

A application program sending data to a peer using request-reply protocol over host-to-host protocol.



Example

The application on host 1 sends a message to an application on host2. In practice this the application calls a function in service interface of the Request Reply Protocol software module.

The dotted lines show *virtual* communication between peer entities.

- RRP attaches some control information in an *RRP header* to data so that its peer RRP will know what to do when the data is received by it. This combined message is sent to the local host-to-host protocol.
- HHP attaches HHP-specific header, and
- the entire message is sent to host 2

Each layer attaches a header (encapsulates the message) as the message goes down.

At host 2, each layer removes its header, performs header specific processing and passes the message up.

The ISO seven layer *open systems interconnection* model

- L_1 Physical Layer: network hardware; mechanical and electrical connections.
- L_2 Data Link Layer: managed the transmission of data across the physical network. Framing, data transparency and error control.
- L_3 Network Layer: define how addresses are assigned and how data is forwarded from one network to another: routing
- L_4 Transport Layer: Provides reliable, transparent transfer of data between end points. End to end Error recovery and flow control
- L_5 Session Layer: Provides the control structure for communication between applications. Establishes, manages and terminates dialogues between application entities. Specifies security details.
- L_6 Presentation Layer: Provides independence to the application process from differences in data representation.
- L_7 Application Layer: Each protocol specifies how a particular application uses the network and how an application program on one computer makes a request and how the application on another machine responds.

ISO/OSI and TCP/IP

The seven-layer model had this many layers to provide for compatibility between network teachnologies.

In practice TCP is *the* standard network technology and protocol suit of the internet. It manages with four layers which correspond to the seven-layer model as follows -

<table>
<tr><th>OSI</th><th>TCP/IP</th><th></th></tr>
<tr><td>Application</td><td rowspan="2">Application</td><td rowspan="2">appliction</td></tr>
<tr><td>Presentation</td></tr>
<tr><td>Session</td><td rowspan="2">TCP, UDP, ...</td><td rowspan="2">transport</td></tr>
<tr><td>Transport</td></tr>
<tr><td>Network</td><td>IP</td><td>network</td></tr>
<tr><td>Data link</td><td rowspan="2">Network layer of the internet</td><td rowspan="2">link/physical</td></tr>
<tr><td>Physical</td></tr>
</table>

TCP/IP

- TCP/IP (Transmission Control Protocol / Internet Protocol) is the internets communications standard
- It is a complete suite of protocols and includes network, transport and application layers.
- It is used in many Unix systems, began in the Unix world; Unix is used widely throughout the Internet.

TCP/IP applications (application-layer protocols) include

FTP
: file transfer protocol: nowadays wrapped in a security layer; eg Filezilla

Telnet
: for remote log-in to a server. Now in a security later; eg SSH

DHS
: Domain Name Service.

SNMP
: Simple Network Management Protocol.

SNTP
: Simple Mail Transfer Protocol.

HTTP
: Hypertext transfer protocol - used by web browsers.

TCP (Transport layer)

- *Transmission Control Protocol*.
- Reliable: TCP service takes responsibility for correct delivery of the data to peer; arranges for re-send if a data packet is lost or corrupt;
- *connection-oriented* (see below);
- fragments an incoming byte stream into messages for IP layer;
- reassembles received messages (passed up from IP layer) in correct order, to create an output stream;
- manages flow control.

Connection-oriented

- Communication starts by establising a connection or *virtual circuit* to peer;
- Data sent either way between peers all follows the same route: goes via the connection;
- Connection is closed (virtual circuit destroyed) at end of 'conversation'.
- This software construct is how TCP manages reliable tranport of data in correct order and without data duplication or loss.

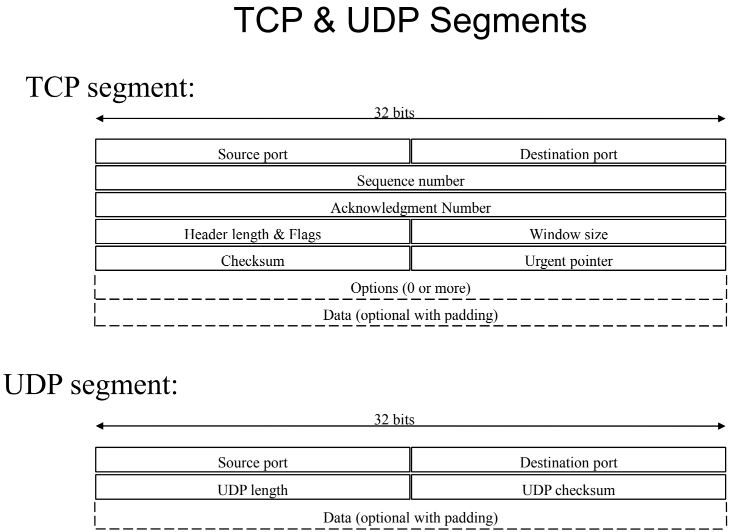
UDP (Transport layer)

- *User Datagram Protocol*.
- A ‘best effort’ service: data errors are detected but no responsibility re-sending. The application has to handle lost or corrupt data.
- *connectionless*; Data transimissions - *UDP datagrams* are separate from one another; each is individually addressed to recipient.
- no sequencing or flow-control.

TCP or UDP?

- UDP is light-weight compared with TCP; it has much less work to do compared with TCP.
- Preferred in applications which require many short messages to be sent at high speed: VOIP (telephony), audio, video streaming, ...
- TCP preferred where data exchanges are more ‘heavy-weight’: HTTP, WWW

Transport layer data



Transport layer data

Port numbers define the ends of logical connections.

- a message from a process on a host will go to a process on the destination host using the appropriate port number.

IP layer

IP

- *Internet Protocol*:
- an 'unreliable' connectionless best effort IP packet delivery service.
- A question for you to think about: the IP service interface supports both *reliable* TCP and emphbest-effort UDP. How does an 'unreliable' or best-effort-only service sipport a *reliable* service?
- Addressing
- Routing

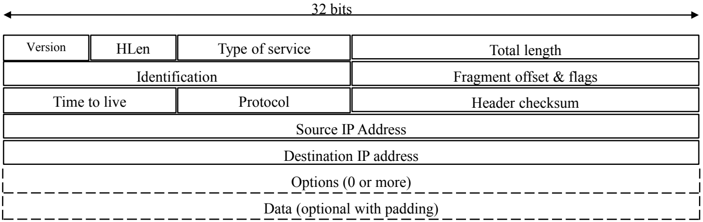
ARP / RARP

- (Reverse) Address Resolution Protocol
- Maps IP addresses onto data link layer addresses such as Ethernet card addresses: eg 193.63.32.233 to (MAC address) 008002B39DD10
- Its functions are defined as part of TCP/IP but its implementation is dependent on the network type.
- TCP/IP does not define what happens below the IP layer.

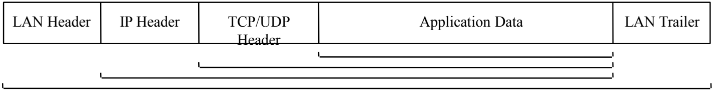
Transport layer data

IP Datagram and TCP/IP Packet Format

- IP datagram format:



- TCP/IP packet structure:



32-bit IP Addresses

32 bit addresses specify sources and target hosts.

- normally represented in *dotted decimal format*, each block describing 8 bits: thus 193.63.32.233 is the sames as 0xC13F20E9 or in binary, 11000001 00111111 00100000 11101001
- Each IP address has two components: the higher bits identify a local network; the lower bits an individual host or interface to a router, within a network.
- Hosts on the same network must have IP addresses with the same network part, but different host parts.
- Hosts with different network address parts might be connected by a router (eg, with two interfaces with network parts agreeing with the two hosts).
- Originally the network part was 8, 16, or 24 bits (class A, B or C); now using CIDR (classless interdomain routing) and VLSM (variable-length subnet masking) can be any size.
  - eg 223.1.252.3 within the subnet 223.1.252.0/22 would need subnet mask 255.255.252.0, in binary, 11111111 11111111 11111100 00000000: 22 bits' network part.

Domain names

To avoid referring to individual numerical IP addresses the concepts of *domain names* and *host names* developed.

- Domain and host names are mapped to IP addresses.
- cougar.unn.ac.uk $\longrightarrow$ 193.63.32.233;
- *no* logical relationship between the parts of an IP address and its domain name.
- The mapping is done using *Domain Name Resolution*, normally with the help of a *Domain Name Server*.

There are not enough 32-bit addresses!

- More than 2^{32} (4 billion) addresses are needed.
- For many years the internet has 'coped' by allowing addresses to be duplicated, with routers doing IP address 'translation' to prevent address clashes outside of LANs.
- this makes DNS especially handy!
- IP v 6 specifies 64-bit addresses - clean resoltution of the problem but IP v 6 is very slow to be adopted by users.

Programming TCP connection

The focus is on applications using transport layer services, especially TCP, UDP; useful for development of *distributed* applications. The basic software entity is a *socket*.

- role similar to a file-handle;
- First, the socket is connected to a remote host (compare with opening a file); then data is input/output through the socket; when complete, the socket is closed the connection is broken.
- Implemented in Java using package `java.net`, especially class `java.net.Socket`.
- C provides the socket class and a library of socket functions prototyped in `<sys/socket.h>`.

Programming TCP connections

```
#include <sys/socket.h>
```

- `int socket(...)`; creates a new socket
- `int gethostname(char *name, int namelen)`; translates a host name to an ip address
- `int bind(int s, struct sockaddr *name, int namelen)`; binds a socket to a specific address (and port)
- `int connect(int s, struct sockaddr *addr, int *addrlen)`; used by client to request a socket connection to a remote address (and port)
- `void listen (socket_id s, int backlog )`; causes server to start listening for requests for a connection
- `int accept(int s, struct sockaddr *addr, int *addrlen)`; used by server to accept a connection request from a client
- `int read(int d, char *buf, int nbytes)`;
- `int write(int d, char *buf, int nb)`; d = socket id; reads from/writes to a socket
- `int close(int d)`;

TCP server logic

In general, the behaviour you have to program is dependent on the state of the system. There you tend to write such constructs as

```
Fix the port number
Create a socket for the server
Start listening on the socket for requests to connect
Repeat
  Wait for a request for a connection.
  Accept function returns id of a new socket which will manage t
    connection; also gets the name of the client host.
  Spin off a new thread to serve the client;
```

Serving the client

- A subroutine running in a new thread
- Uses socket returned by Accept function
- Uses socket I/O functions to send data to / receive data from client according to protocol
- When finished, closes the socket.

TCP client logic

In general, the behaviour you have to program is dependent on the state of the system. There you tend to write such constructs as

```
Specify server name/addr, port num to which we wish to connect;
Create a socket;
Bind this socket to host name and port number;
Request a connection to the server host / port;
if (return value indicates connection successful)
 use socket I/O functions to send data to/receive data;
Close the socket when finished.
```

- The port number is same at both ends: identifies the virtual circuit between the client and server.

Programming UDP

Again use a *socket* but no connection is established. Instead, a version of the send function is used which incorporates a destination address parameter:

```
int sendto(int sockID, char * msg,
   unsigned int msgLen, int flags,
   struct sockaddr * destAddress,
   unsigned int addressLen)
```

...and a version of the receive function is used which incorporates a source address parameter:

```
int recvfrom(int sockID, char * msg,
   unsigned int msgLen, int flags,
   struct sockaddr * sourceAddress,
   unsigned int * addressLen)
```

The return value in each case is the number of bytes sent/received. Note is the use of pointers to provide in/out parameters: note the int * (rather than int) for the receive function's address length parameter.

Further reading

RFC1122 Requirements for Internet Hosts Communication Layers

- https://en.wikipedia.org/wiki/Internet_protocol_suite
- https://www.w3.org/People/Frystyk/thesis/TcpIp.html
- https://en.wikibooks.org/wiki/A-level_Computing/AQA/Computer_Components,_The_Stored_Program_Concept_and_the_Internet/Structure_of_the_Internet

Requests for comment (RFCs): the following are easily found by internet search:

- RFC1123 Requirements for Internet Hosts Application and Support
- RFC768 User Datagram Protocol
- RFC793 TRANSMISSION CONTROL PROTOCOL