TCP

[Wait! If you are not familiar with file descriptors and the UNIX read and write system calls, read chapter 10 of Bryant and O'Hallaron and/or my summary before going on.]

Recap: Using the IP protocols (IPv4 or IPv6), we can move packets from host to host even if they are on LANs that are on opposite sides of the globe.

The IP protocols have some shortcomings.

- It is not robust (reliable).
  - ∗ In IP packets can be lost.
  - ∗ Receivers do not acknowledge receipt of packets
  - ∗ Receivers can not complain about missing packets
  - ∗ Senders can not re-send packets
- Packet based
  - ∗ A packet is an isolated communication.
    - · Limited in size.
    - · Limited in time — arrives at application all at once, as a unit.
    - · Disjoint: Packets are unordered and may arrive in a different sequence than they are sent in.
- One way.
  - ∗ Information flows from one host to another
- Host to host
  - ∗ It allows communication between hosts (computers), but not between processes.

- · Suppose `fred.astair.com` is running both a web server and an ftp server.
- · IP does keep messages for one separate from messages for the other.

TCP (*Transmission Control Protocol*) is a layer 4 protocol that allows applications (processes) to set up connections

1	Physical	Move bits across some physical medium (wire, optical fiber, radio) Ethernet physical layer, IEEE 802.11
2	Data link	Move data frames between 2 physically connected nodes Ethernet, IEEE 802.2
3a	Network	Move data frames around a local area network (LAN) Ethernet
3b	Internetwork	Move packets between hosts on different LANs IPv4, IPv6
4	Transport	Move streams or datagrams between ports TCP, UDP

TCP supports

- Reliable transmission.
  - ∗ If packets are lost, the protocol recovers. Data is resent.

- Streaming.
  - ∗ Once a *TCP connection* is established, it acts as pair of FIFO (first-in-first out) queues of bytes.
  - ∗ Data put in at one end of a queue appears at the other end *in the same sequence*.
- Bidirectional (two way) communication
  - ∗ One queue goes one way and the other goes the other.
- Process to process.
  - ∗ On each host the connection is associated with a 16-bit port number.
  - ∗ By default only one OS process is allowed to use the connection associated with a given port number.
  - ∗ Thus communication is securely between two processes.

TCP messages are exchanged using IP. Thus TCP messages are the payload for IP packets.

Other level 4 protocols

The *User Datagram Protocol* (UDP is an alternative to TCP.

- It provides process-to-process communication that is packet (datagram) based, unidirectional, and unreliable.

TCP connections

Port numbers

There are 2^{16} TCP ports for each host numbered from 0 through $65,535$.

- Ports 0 through $49,151$ are used to represent services.
- Each service is associated with a port number. Some common examples include
  - 80 — http (web)
  - 8080 — https (web)
  - 22 — secure shell (ssh)
  - 115 — secure ftp
  - 25 — smtp (sending email)
  - 143 — imap (getting email)
- I.e., if I am writing a client for a particular service, I need to know what port number is associated with that service, and I need to know the host name (or IP address) of the host machine.
- If I am creating a new service, I'll just pick a number from 48,620 through 49,150.
  - Why this range? These numbers are not currently registered with the IANA.
  - Numbers above 49,151 are assigned by the operating system (see below) and so could be in use already.
- Ports $49,152$ through $65,535$ are used for connections.

Servers and clients

The words “server” and “client” have many meanings

- In software architecture: “server” can mean a process whose job it is to service requests coming from other processes. Those processes are called “clients”.
- “server” is also used to mean the computer on which server processes run.

When a TCP/IP connection is made between two processes, the two processes have different behaviors.

In this section of the notes “server” is used to mean the process that *listens for connections*. And “client” means the process that *sends a connection request*.

Often the server and client, in the TCP sense are servers and clients in the software architecture sense, but they don’t need to be.

Making a connection

A *server process* creates an end-point that *listens* one a port.

- For example a web server on `www.boole.uk` listens on port 80.

Clients send connection requests to a port on the server’s host.

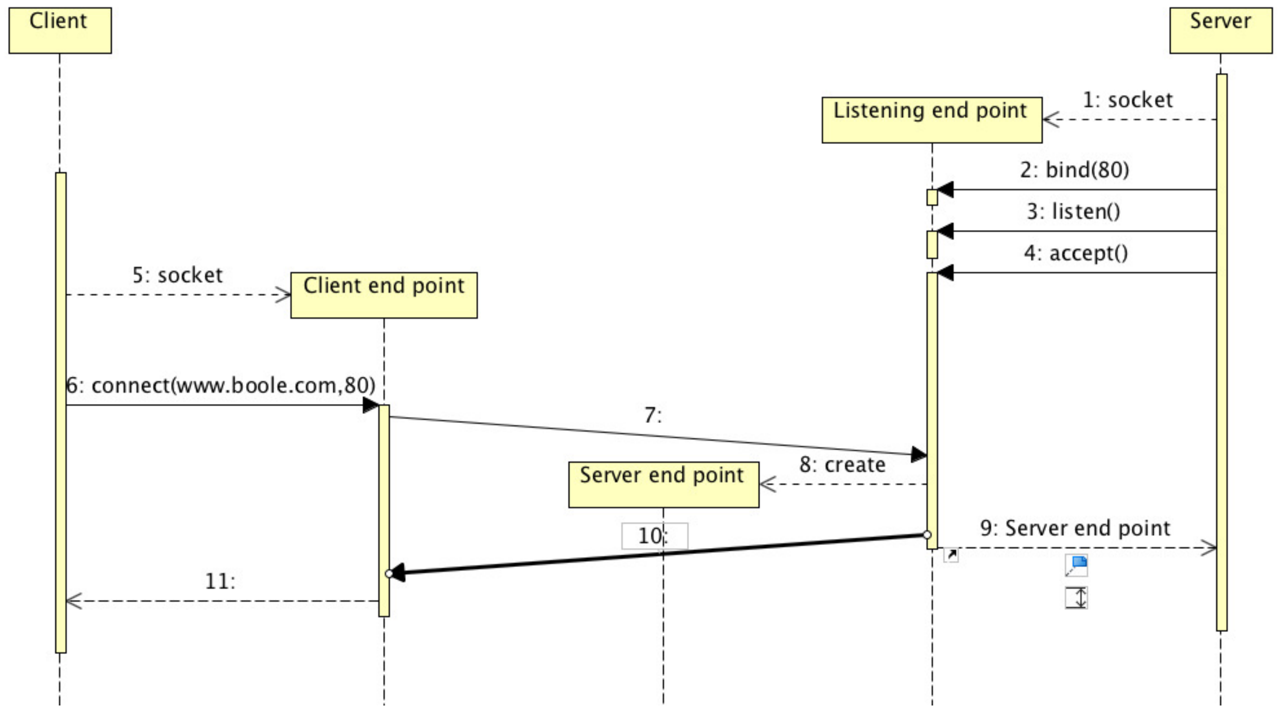
- For example a web browser on `claude.shannon.com` sends a connection request packet to port 80 of mona.engr.mun.ca

When server *accepts* the connection, the client and server OSs pick unused ports in the range 49151 to 65535.

- For example the web server might get port 54,321.
- The browser might get 61,234

So, every time the server process accepts a connection, a new endpoint is created with a new port number.

The diagram below shows this conceptually. (In reality messages are not sent to the end points, but to the OS).



Note on diagrams:

- These are "message sequence diagrams"
- Time runs from top to bottom.
- Horizontal sold arrows are calls from application to OS.

- Horizontal dashed lines are returns from calls.
- Diagonal lines are packets going across the internet.

Data transmission

Once the connections are made, the processes can send data to each other.

1. For example the client might send a sequence of bytes to port 54,321 of `www.boole.uk`
2. Time passes as the bytes travel through the network.
3. Some packets might get lost, but they are resent.
4. Eventually, the server reads the byte sequence
5. The server recognizes this sequence as a request for a particular web page.
6. The server sends a sequence of bytes to port 61,234 of `claude.shannon.com` as a response.
7. Time passes.
8. The client reads those bytes.
9. The client recognizes the sequence as containing an HTML page and displays the page.

And the client and server can continue to send bytes to each other until one or the other closes the connection.

Multiple connections

At any time the server can accept more connections, e.g. from other clients.

The server can then service several clients at once.

Consider a simple chat server that hosts a single conversation among several clients.

```
start listening on port 1234
while true
    wait until one of the following is possible
    either
        accept a new connection
    or
        read some bytes from an existing connection
    send any new bytes to all connections tagged with their
    origin
end while
```

End of slides