07. Berkeley Sockets Addressing

Overview of the daytime Server

The operation of the Daytime application is as follows:

- Client calls CONNECT to connect to the server
- Server calls ACCEPT to accept the connection request
- Server returns a formatted string using the SEND primitive
- Server applications calls CLOSE to close the connection
- Client calls RECV to retrieve the data from the connection
- Client closes the connection using the CLOSE primitive
- The client application terminates using exit(0)

Key points to note in the Daytime server code:

- Server address structure
- Calls to bind(), listen() and accept()

The echo client-server

- The client application sends (send()) a string from the command-line to the server across an open connection
- The server application reads (recv()) the string and returns it to the client application (send()) exactly as it came in
- Both applications call for the connection to be closed (close())

The recv() primitive

The while loop is necessary as the data may not arrive across the connection in a single transfer.

- Data arriving from the remote socket is stores in the RECV-Q buffer within the local TCP entity.
- Repeated calls to recv() are needed to transfer this data from TCP (Transport Layer) into the application (Application Layer).

- numBytes is the return value from recv()
  - it represents the number of bytes read from the socket
  - < 1 → error condition
  - 0 → closed connection (remote application has called close())
  - > 1 → open connection with potentially more data to be received

Addressing

A key aspect of Networked Applications is **Addressing**. In order for the Client and Server applications to communicate with each other, some form of explicit addressing is required.

The Destination Server application requires an unambiguous (**unique**) **address** in order for the Source Client application to initiate a connection request. Uniqueness can only be achieved using both **IP** and **TCP addressing**.

The IP layer is responsible for delivering datagrams/packets to remote hosts across an internetwork. It uses IP addressing to achieve this host-to-host delivery.

However, the data encapsulates inside these datagrams/packets is typically destined for an application in the Application Layer.

Transport Addressing

Transport protocols such as TCP provide services to the application layer.

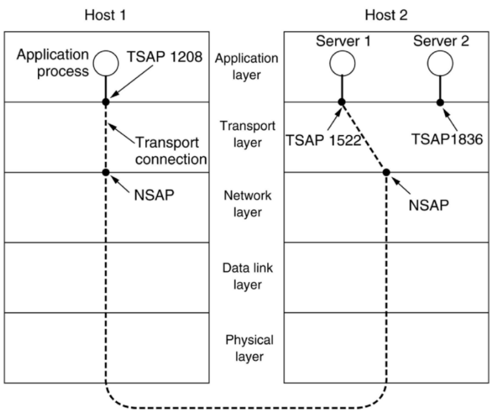
To ensure unambiguous (unique) addressing of individual applications, the Transport Layer provides its own addressing schema separate to the IP layer.

- These are generally known as Transport Service Access Points (TSAPs)
- These TSAPs uniquely identify entities in the Transport layer known as **end points**
- In TCP parlance these end points are known as **port numbers** (ports)

Port numbers are used by:

- Server applications to advertise their services and to listen for Connection Requests
- Client applications to uniquely identify a Server application when making a Connection Request

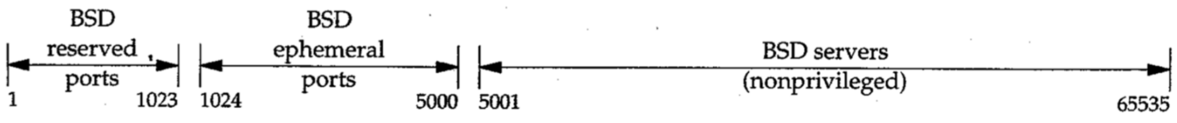
The network layer also defines end points. These are known as Network Service Access Points (NSAPs). IP addresses are examples of NSAPs.



Relationship between NSAPs, TSAPs and transport connections

The TCP Port Number Range

TCP Port numbers are 16 bits long, this creates a port number address space comprising approx. 65K addresses (16 bits → 2^16 addresses)



Reserved addresses are for well known applications: HTTP (port 80), FTP (ports 20 and 21), Telnet (port 23), ... These can only be allocated by users with SU privileges.

Ephemeral addresses are allocated by TCP to Client applications. It is not immediately obvious that Client applications require a port number, however, there needs to be a return address for data from the Server application.

Non-privileged addresses are for any other applications. This is the range used in the lab exercises.

The Ephemeral and Non-privileged ranges differ on different OS. To view the range:

`sudo sysctl net.ipv4.ip_local_port_range`

Socket Identifiers

The concept of a socket or end-point is complicated by the manner with which it is referenced within each of the Application and Transport layers:

- From an Application layer perspective, sockets are referenced using a file descriptor.
- From a TCP layer perspective, a socket is identified by a combinations of an NSAP and TSAP separated by a colon (:)
- Even though both layers are referencing the same entity, the socket, the descriptors are used very different.

This difference is further complicated by the fact that the call to accept() returns a new socket (the listening socket and the connected socket).

- Referencing the connected socket from within the application is straightforward. A separate and unique int descriptor is returned. All interactions with the listening and connected sockets are obvious.
- Referencing the connected socket from within the TCP layer is complicated. Each of the sockets will use the same NSAP:TSAP combination. To differentiate between the sockets, the socket-pair need to be considered.

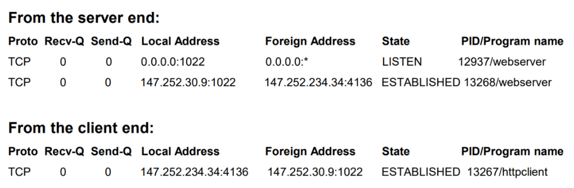
Examining Socket Details

Socket-pairs are used to identify TCP connections.

TCP connections can be viewed using the netstat utility: `netstat -ntap`. This returns details of active TCP connections within the host OS.

- 'n' reveals IP addresses in dotted-decimal notation
- 't' filters on TCP address only
- 'a' shows all connections
- 'p' reveals the process id details for each connection

Socket Identifiers



```
From the server end:
Proto  Recv-Q  Send-Q  Local Address        Foreign Address      State        PID/Program name
TCP      0       0     0.0.0.0:1022         0.0.0.0:*            LISTEN       12937/webserver
TCP      0       0     147.252.30.9:1022    147.252.234.34:4136  ESTABLISHED  13268/webserver

From the client end:
Proto  Recv-Q  Send-Q  Local Address        Foreign Address      State        PID/Program name
TCP      0       0     147.252.234.34:4136  147.252.30.9:1022    ESTABLISHED  13267/httpclient
```

netstat output example

- Server with listening and connected sockets on port 1022
- Client on an ephemeral port 4136

The columns Local Address and Remote Address contain details of the socket at each end of a connection. Notice that TCP refers to a socket using a NSAP:TSAP combination (IP:Port).

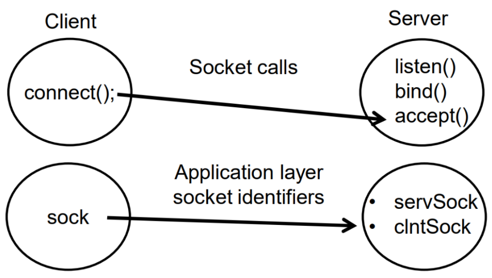
Socket Pairs

Each row in the output from netstat relates to a connection. A TCP connection can be considered as a connection between two sockets, a **local** socket and a **remote** socket.

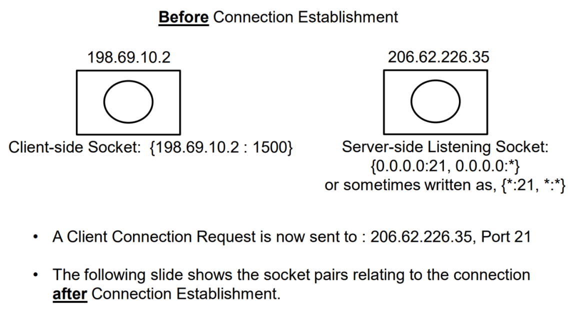
The combination of Local Address and Remote Address is the **identifier** for a connection, known as a **Socket Pair**.

Socket Pairs are how TCP views connections. For each connection, the detail contained in each row is reversed depending on which end of the connection the netstat command is run.

Using socket-pair identifiers enables TCP to separately and unique identify a server’s listening and potentially multiple connected sockets. This is important when incoming data (incoming to the TCP layer) needs to be passed to a local socket.



How connections are established from a Socket Pair perspective



<u>Before</u> Connection Establishment

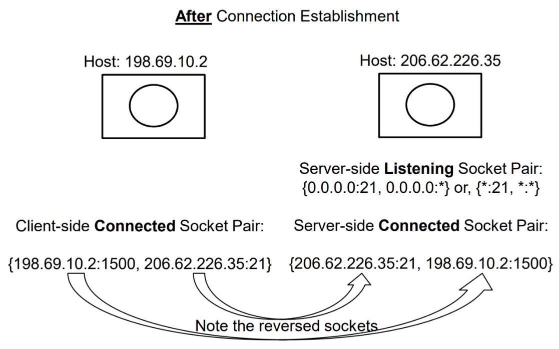
198.69.10.2

Client-side Socket: {198.69.10.2 : 1500}

206.62.226.35

Server-side Listening Socket:
{0.0.0.0:21, 0.0.0.0:*}
or sometimes written as, {*:21, *:*}

- A Client Connection Request is now sent to : 206.62.226.35, Port 21
- The following slide shows the socket pairs relating to the connection **<u>after</u>** Connection Establishment.



<u>After</u> Connection Establishment

Host: 198.69.10.2

Host: 206.62.226.35

Server-side **Listening** Socket Pair:
{0.0.0.0:21, 0.0.0.0:*} or, {*:21, *:*}

Client-side **Connected** Socket Pair:

{198.69.10.2:1500, 206.62.226.35:21}

Server-side **Connected** Socket Pair:

{206.62.226.35:21, 198.69.10.2:1500}

Note the reversed sockets