Overview: Transport Layer & Applications

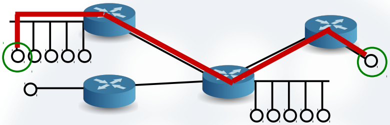
Jose L. Muñoz, Oscar Esparza, Juanjo Alins, Jorge Mata
Telematics Engineering
Universitat Politecnica de Catalunya (UPC)

Outline

Outline

The Network Layer

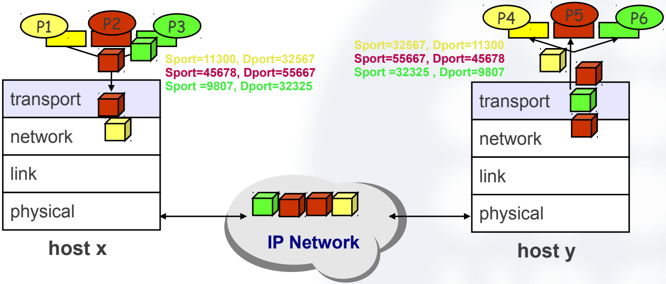
- The network layer is a scalable way of interconnecting data link layer technologies.
- Basic IP provides an interface to interface (NIC-to-NIC) best effort service for delivering datagrams.



- A best effort service means that:
  - A correct delivery of datagrams is not guaranteed.
  - There might be lost datagrams, incorrect datagrams or disordered datagrams.

Transport Layer

- The main goal: **implement communications between processes (running applications) that are in general in different systems.**
- Also called end-to-end communications.
- Introduces the concept of **PORT** for multiplexing and demultiplexing.



- The transport layer **provides the main network API** (Application Programming Interface) to allow using the network to user space processes.

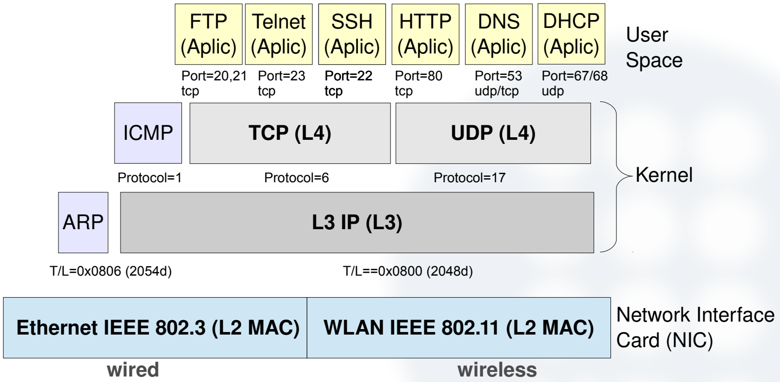
Transport Ports

- Identifiying processes:
  - Each OS technology identifies its currenty running processes.
  - Unix-like systems use the Process Identifier (PID).
  - However, we want a generic identifier (different of the PID) for multiplexing transport communications.
- The **port** is a parameter for multiplexing that is dynamically assigned to any running process that requires a transport communication with another process.
- Each transport PDU carries:
  - A source Port (SPort) that identifies the process sending the PDU.
  - A destination port (DPort) that identifies the process in the destination host.
- For the main transport protocols of Internet (TCP and UDP), ports have 16 bits (65536 ports).

Basic Transport Protocols

- **User Datagram Protocol (UDP):**
  - UDP is the simplest transport protocol.
  - UDP is a message-oriented protocol (datagram protocol).
  - Each UDP datagram (message) is encapsulated over an IP datagram.
  - UDP only offers multiplexing capability (using ports) and a checksum for discarding wrong data for its users.
  - UDP does not provide error, flow or congestion control.
- **Transport Control Protocol (TCP):**
  - TCP provides applications with a full-duplex communication, encapsulating its data over IP datagrams.
  - TCP communication is connection-oriented because there is a handshake of three messages before data can be sent.
  - The TCP communication is managed as a data flow (TCP is not message-oriented).
  - Apart from multiplexing capabilities, TCP is a reliable protocol because it adds support to detect errors or lost data and retransmit them (**ARQ end-to-end error control**).
  - TCP also supports **an end-to-end flow control** and a **congestion control**.

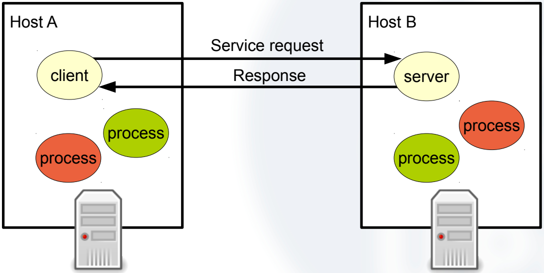
TCP/IP Protocol Stack



Outline

Client/Server Model I

- The client/server model is the most widely used model for communication between processes.
- Clients make requests to servers.
- Servers respond and they can generally support numerous clients.



Client/Server Model II

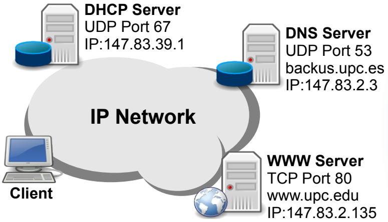
- In Unix-systems, server processes are also called daemons.
- In general, a daemon is a process that runs in the background, rather than under the direct control of a user.
- Typically daemons have names that end with the letter ”d“ (e.g. telnetd, ftpd, httpd or sshd).
- **Clients initiate the interprocess communication,** so they must know the address of server.

Client/Server TCP/IP

- In the TCP/IP domain, the address of a process is composed of:
  1. An identifier called **IP address** (@IP) that allows reaching the destination "user space" or host in which the server process is running.
  2. An identifier of the process called **transport port**.
  3. The **transport protocol** used.
- So, the client process needs to know these three parameters to establish a TCP/IP socket with a network daemon (server).
- Internet uses a scheme called: well-known services or **well-known ports**.

Well-known Ports I

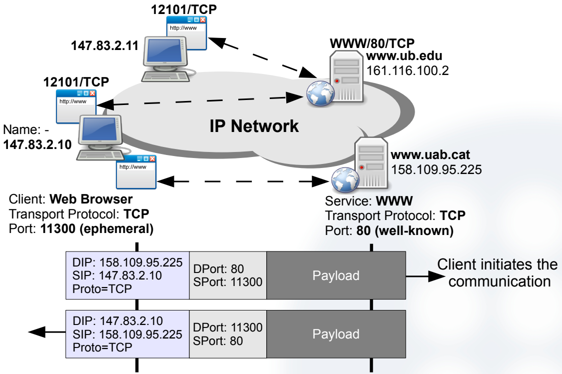
- The client needs to know IP, protocol and port to create a ”socket“ (communication) with a server.
- The client usually knows the IP (or **name**) of the server and there is a well-known transport protocol and port per service (determined by the application used).



- For example, HTTP servers (for the Web) use TCP/80, DNS servers use UDP/53 for name queries and the DHCP servers use UDP/67.

- Example clients: `host` command (DNS), `firefox` (DNS and HTTP), etc.

Well-known Ports II



- Servers can manage multiple clients, identified by different `@IP/L4_Proto/Port` tuples.
- Typically, each client process asks for a free port to its OS kernel (ephemeral SPort).

Outline

`ifconfig`

- The `ifconfig` command is the short for interface configuration.
- This command is present at any Unix-like system.
- In its simplest form, `ifconfig` can be used to set the IP address and mask of an interface. Syntax:

```
# ifconfig IF @IP netmask MASK
```

Example:

```
# ifconfig eth0 192.168.0.1 netmask 255.255.255.0
```

route

- The `route` command is used to define routes statically.
- This command is present at any Unix-like system.
- The most commonly used syntax of `route` is the following:

```
# route (add|del) -net @NET netmask MASK [gw @IP dev IF]
```

- We can also use the CIDR notation `@NET/X` and
- we can view the current routing table with `-n` (without name resolution).
- Example:

```
alice:~# route add -net 10.0.0.192/26 gw 10.0.0.31
alice:~# route -n
Kernel IP routing table
Destination     Gateway      Genmask          Flags Metric Ref   Use Iface
10.0.0.192      10.0.0.31    255.255.255.192 UG    0      0        0 eth0
10.0.0.0        0.0.0.0      255.255.255.128 U     0      0        0 eth0
```

Permanent Configuration

- The configurations made with `route` and `ifconfig` commands are ephemeral.
- To make the network configuration permanent, in Linux distros like Debian or Ubuntu, the majority of network setup can be done via the interfaces configuration file at /etc/network/interfaces.
- Example:

```
auto eth0
    iface eth0 inet static
        address 192.168.0.10
        netmask 255.255.255.0
        gateway 192.168.0.1
auto eth1
    allow-hotplug eth1
    iface eth1 inet dhcp
nameserver 10.1.1.1
```

- If you change the configuration of this file, you have to restart “networking” to enable the changes:

```
alice:~# /etc/init.d/networking restart
```

Services

- The file /etc/services is used to map port numbers and protocols (TCP/UDP) to service names.
- Service names can be used by programs.
- Example:

```
alice:~# less /etc/services
# Network services, Internet style
# Note that it is presently the policy of IANA to assign a single well-known
# port number for both TCP and UDP; hence, officially ports have two entries
# even if the protocol doesn't support UDP operations.
# Updated from http://www.iana.org/assignments/port-numbers and other
# sources like http://www.freebsd.org/cgi/cvsweb.cgi/src/etc/services .
# New ports will be added on request if they have been officially assigned
# by IANA and used in the real-world or are needed by a debian package.
# If you need a huge list of used numbers please install the nmap package.

tcpmux          1/tcp                           # TCP port service multiplexer
echo            7/tcp
echo            7/udp
discard         9/tcp           sink null
discard         9/udp           sink null
systat          11/tcp          users
daytime         13/tcp
daytime         13/udp
...
```

netstat

- The command `netstat` (network statistics) shows established or listening sockets and several related statistics.
- Options:

-t	TCP connections.
-u	UDP connections.
-l	listening sockets.
-n	addresses and port numbers are expressed numerically and no attempt is made to determine names.
-p	show which processes are using which sockets (you must be root to do this).
-r	contents of the IP routing table.
-i	displays network interfaces and their statistics.
-c	continuous display.
-v	verbose display.
-h	displays help at the command prompt.

- Example:

```
alice:~# netstat  -tnlp
Active Internet connections (only servers)
Proto Local Address   Foreign Address   State    PID/Program name
tcp    0 0.0.0.0:80   0.0.0.0:*         LISTEN   1690/apache2
tcp    0 :::22        :::*              LISTEN   1037/sshd
```

Outline

Socket API I

- TCP/IP communications where developed in the context of Unix systems.
- One of the main ways of implementing TCP/IP communications is to use the ”socket“ API.
- An application programming interface (API) is an interface implemented by a software program to enable interaction with other software.
- Usually, an API is presented as set of functions collected in a library (C/C++ library).
- It may include specifications for routines, data structures, object classes and protocols used to communicate between the consumer and implementer of the API.
- **In Unix-like systems, sockets are the default API implemented by the kernel for providing an interface to networks to user-space processes.**

Socket API II

- A socket is an endpoint of a bidirectional inter-process communication.
- The sockets API implemented in the kernel forbids two user-space processes to choose the same socket (`L4_Proto/Port`).
- With the socket API we can create TCP or UDP network sockets as client or server.
- By now, we will simplify this issue saying that:
  - Servers open sockets for ”listening“ for clients.
  - Clients open sockets for connecting to servers.
- In general, a server listens for traffic PDUs comming from ”any” interface (but system calls also let you select a particular interface or interfaces).
- The system calls for servers also allow to serve multiple clients simultaneously (multi-client server).

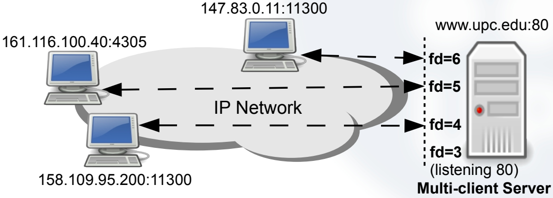
Socket API III

- One of the main system calls is *socket()*, which returns a file descriptor.
- File descriptors of network communications receive the name of **socket descriptors** (*sd*).
- Socket descriptors are used to write (send) data to the "network" and to read (receive) data from the "network".
- As "network", we are referring to TCP and UDP communications.
- Unix Sockets.
  - In Unix-like systems, we have also Unix sockets.
  - Unix sockets are similar to TCP/IP sockets but they are local to the system [1].
  - They use filenames as addresses instead of tuples of network parameters (`@IP/L4_Proto/Port`).

[1] A detailed description of these sockets is beyond the scope of this document.

Multi-client Servers

- Multi-client servers can serve several clients simultaneously.
- They work as follows:
  - The server creates a socket using the desired port in listening state.
  - The kernel returns the associated file descriptor (in our example fd=3).
  - Then, the server reads service requests through this fd.
  - For each new service request, clients are distinguished by `@IP/L4_Proto/Port`.
  - Finally, for each new client, the server creates a new socket (in our example fds 4,5 and 6).



`lsof` and Sockets

- Recall that the `lsof` command shows us the “list of open files” (including socket descriptors).
- Example:

```
alice:~# $ lsof -a -p 4578 -d0-10
COMMAND PID    USER    FD   TYPE DEVICE SIZE/OFF NODE NAME
nc      4578   user1    0u   CHR  136,2     0t0     5 /dev/pts/2
nc      4578   user1    1u   CHR  136,2     0t0     5 /dev/pts/2
nc      4578   user1    2u   CHR  136,2     0t0     5 /dev/pts/2
nc      4578   user1    3u  IPv4 149667     0t0   TCP localhost:
48911->localhost:12345(ESTABLISHED)
```

- The previous command is used to view the file descriptors of the process with PID 4578.
- We see that the fd=3 is associated with an established TCP connection.

Outline

netcat I

- The `netcat` application can be used to create a process that opens a raw TCP or UDP socket as client or server.
- It is very useful tool for testing networks (known as “Swiss Army Knife of networking“).
- Note. Make sure that your `netcat` command is the “traditional” one.
- `netcat` as client:

```
$ nc hostname port
```

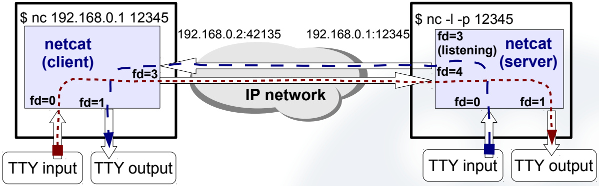
- We can use the -l (listening) option to make `netcat` work as a server:

```
$ nc -l -p port
```

- A server with `netcat` **is not multi-client.**

netcat II

- Once a client is connected, the behavior of `netcat` until it dies (e.g. CRL+c) is as follows.



- The `netcat` processes read data from stdin (fd=0) and write these data to fd=3 in the client or to fd=4 in the server.
- On the other hand, the data received from the network is read from fd=3 (client) or fd=4 (server) and then, written to fd=1 (stdout).
- Note. To implement `netcat` we can use the C system calls `write()` and `read()`, these system calls use as parameter the fd.

netcat III

- Options of `netcat` (traditional):

-h	show help.
-l	listening or server mode (waiting for incoming client connections).
-p port	local port.
-u	UDP mode.
-e `cmd`	execute `cmd` after the client connects..
-v	verbose debugging (more with -vv).
-q secs	quit after EOF on stdin and delay of secs.
-w secs	timeout for connects and final net reads.

- Transfer files:

```
$ cat file.txt | nc -l -p 12345 -q 0
```

- Create a remote terminal:

```
$ nc -l -p 12345 -e /bin/bash
```

- Echo server with `netcat` (only one client at one time).

netcat IV

```
#!/bin/bash
# nc-echo.sh
while true
do
nc -l -p 12345 -e /bin/cat
done
```

- Execute client and server and take a look at open files.
- View established connections:

```
$ netstat -tnp
Active Internet connections (w/o servers)
Proto Local Address     Foreign Address   State       PID/Program name
tcp   127.0.0.1:41426   127.0.0.1:12345   ESTABLISHED 14688/nc
tcp   127.0.0.1:12345   127.0.0.1:41426   ESTABLISHED 14687/cat
```

- View open files of `cat`:

```
$ lsof -a -p 14687 -d0-10
COMMAND PID   USER FD TYPE DEVICE NAME
cat     14687 user 0u IPv4 39942  TCP localhost:12345->localhost:41426
cat     14687 user 1u IPv4 39942  TCP localhost:12345->localhost:41426
cat     14687 user 2u CHR  136,6  /dev/pts/6
```