CIS520 – Operating Systems
Handout 18
UDP, TCP

- IP delivers packets to machines. But, need higher level abstractions. UDP delivers packets to ports on machines; TCP provides reliable, stream-based communication to a port on a machine.
- What is a port? It is an abstraction for a communication point. A process can read and write data to and from a port. One machine can have multiple ports. Usually, dedicate ports to different functions. Have an ftp port, a default telnet port, a mail port, etc.
- UDP message contains a header and data. UDP header contains:
  - Source Port: port from which message was sent.
  - Dest Port: destination port on destination machine.
  - Length: length of UDP packet.
  - UDP checksum: checksum.

  To send a UDP packet, encapsulate it in an IP packet, then send the IP packet to the appropriate machine. The whole UDP packet is in the data area of the IP packet.
- The OS on the machine will receive the IP packet, realize that it contains a UDP packet, then pass the UDP packet on to the process waiting for input on the destination UDP port. The OS realizes that the IP packet contains a UDP packet by looking at the protocol field in the IP header.
- When sending a UDP packet to a machine, what port should it be sent to? There are a set of well-known ports that provide standard services available via UDP. Well-known ports go from 0 to 255, and only root processes can read or write from ports with these numbers. An example: port 79 is the finger port. Port 69 is the Trivial File Transfer Protocol (TFTP) port. By convention, all machines use these port numbers for these services.
- Two applications running on different machines can also agree to use their own port numbers for their own communication. Typically, OS dynamically allocates port numbers on request and the applications use those (after setting up the port number communication via some other mechanism).
- UDP does not provide reliable delivery. Users must implement their own reliability. There is a need for a reliable protocol. So, have TCP.
- TCP provides abstraction of a reliable, two way data stream. It is layered on top of IP just like UDP, but is a heavier weight protocol.
- Concept of a stream abstraction. A stream is just a sequence of characters. There are no packet boundaries like there are with UDP and IP. The software breaks the stream up into packets, with the subdivision invisible to the application program. Typically, TCP connections buffer up more than one character before issuing an IP packet. But, can force data delivery if want to.
- To get reliability use a positive acknowledgement with timeout scheme. When send something over a TCP connection, expect to get an ACK back in a fixed amount of time. If don't get the ACK, assume data was lost and retransmit.

- TCP optimizes communication by using a sliding window. Instead of sending one IP packet and waiting for the ACK, it can have multiple outstanding unACKed packets. Basic idea is to fill the network pipe between sender and receiver with data. Keep a steady state with sender always sending and receiver always ACKing.
- TCP also has port abstraction - a TCP connection goes between two ports. To set up a TCP connection, use a three-way handshake. Initiator sends a request, receiver sends an ACK, then sender sends back an ACK to establish connection.
- Can close a TCP connection.
- Building services on top of TCP and UDP. Typically, use a client-server architecture. Basic idea:
  - Server waits at a well-known port for client requests to come in. Typical TCP server ports: 23 is Telnet, 25 is SMTP (Simple Mail Transport Protocol), 21 is FTP.
  - Client allocates a local port number and sends it to server at well-known port. Server will communicate with client using the allocated local port number.
  - When server gets request, it allocates its own local port number and spawns a process to provide service. The process connects up with client over the local ports.
  - Client and spawned server process communicate over the established connection.
- Almost all Internet services done this way. FTP, telnet, etc. all work this way. Many services are just character-oriented streams, and can interact with them at the terminal.
- In general, layer services on top of communications protocols. Can build arbitrary layers. The packets for each service just get encapsulated in the packets of lower-level services. Example from snooped network here at UCSB. Someone is using NFS to look up a file (in this case awk).

```
ETHER:  ----- Ether Header -----
ETHER:
ETHER:  Packet 17 arrived at 9:44:30.37
ETHER:  Packet size = 194 bytes
ETHER:  Destination = 8:0:20:12:6e:8d, Sun
ETHER:  Source      = 8:0:20:12:77:2a, Sun
ETHER:  Ethertype = 0800 (IP)
ETHER:
IP:   ----- IP Header -----
IP:
IP:   Version = 4
IP:   Header length = 20 bytes
IP:   Type of service = 0x00
IP:         xxx. .... = 0 (precedence)
IP:         ...0 .... = normal delay
IP:         .... 0... = normal throughput
IP:         .... .0.. = normal reliability
IP:   Total length = 180 bytes
IP:   Identification = 16822
IP:   Flags = 0x4
IP:         .1.. .... = do not fragment
IP:         ..0. .... = last fragment
IP:   Fragment offset = 0 bytes
IP:   Time to live = 254 seconds/hops
IP:   Protocol = 17 (UDP)
IP:   Header checksum = de8e
IP:   Source address = 128.111.42.18, goofy
IP:   Destination address = 128.111.49.3, comics
IP:   No options
```

```
IP:
UDP:  ----- UDP Header -----
UDP:
UDP:  Source port = 1022
UDP:  Destination port = 2049 (Sun RPC)
UDP:  Length = 160
UDP:  Checksum = 094B
UDP:
RPC:  ----- SUN RPC Header -----
RPC:
RPC:  Transaction id = 821432913
RPC:  Type = 0 (Call)
RPC:  RPC version = 2
RPC:  Program = 100003 (NFS), version = 2, procedure = 4
RPC:  Credentials: Flavor = 1 (Unix), len = 72 bytes
RPC:     Time = 05-Jun-95 17:45:04
RPC:     Hostname = goofy
RPC:     Uid = 0, Gid = 1
RPC:     Groups = 1 0 2 3 4 5 6 7 8 9 12
RPC:  Verifier   : Flavor = 0 (None), len = 0 bytes
RPC:
NFS:  ----- Sun NFS -----
NFS:
NFS:  Proc = 4 (Look up file name)
NFS:  File handle = 0080000400000002000A00000002F2C5
NFS:                546FA3D2000A0000000000025C69FA25
NFS:  File name = awk
NFS:
```