UDP, TCP (Part I)

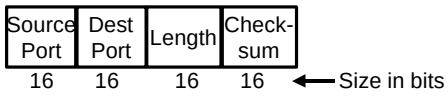
Shivkumar Kalyanaraman
Rensselaer Polytechnic Institute
shivkuma@ecse.rpi.edu
http://www.ecse.rpi.edu/Homepages/shivkuma

- **UDP: connectionless, end-to-end service**
- **UDP Servers, Interaction with ARP**
- **TCP features, Header format**
- **Connection Establishment**
- **Connection Termination**
- **TCP options**
- **TCP Servers**

Ref: Chap 11, 17,18; RFC 793, 1323

User Datagram Protocol (UDP)

- **Connectionless end-to-end service**
- **No flow control. No error recovery (no acks)**
- **Provides port addressing**
- **Error detection (Checksum) optional. Applies to pseudo-header (same as TCP) and UDP segment. If not used, it is set to zero.**
- **Used by SNMP, DNS, TFTP etc**



More UDP

- Port number: Used for (de)multiplexing. Client ports are ephemeral (short-lived). Server ports are "well known".
- UDP checksum similar to IP header checksum, but includes a pseudo-header (to help check source/destination). Fig 11.3
- UDP checksum optional, but RFC 1122/23 (host reqts) requires it to be enabled
- Application message is simply encapsulated and sent to IP => can result in fragmentation. Newer systems use some path MTU discovery algorithms at the IP layer.

UDP and ARP

- When UDP datagram fragments at the host, each fragment may generate an ARP request (results in an ARP reply: *ARP flooding)*
  - RFC 1122/23 limits max ARP rate to 1 request per second, and requires the ARP Q to be at least of size one

Other UDP effects

- Datagram truncation possible at destination if dest app not prepared to handle that datagram size ! (note: TCP does not have this problem because it has no message boundaries)
- UDP sources ignore source quench messages => can't respond to packet losses.

UDP Servers

- Client-Server architecture**: basis for most distributed apps today (eg Web, telnet, ftp)**
- **Most UDP servers are *"iterative"* => a single server process receives and handles incoming requests on a *"well-known"* port.**
- **Can filter client requests based on incoming IP address, client IP address, incoming port address, or wild card filters**
- **Port numbers may be reused, but packet is delivered to at most one end-point.**
- **Queues to hold requests if server busy**

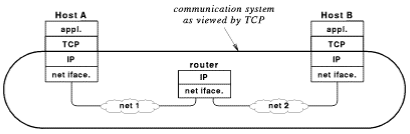
TCP: Key features

- **Connection-oriented**
- **Point-to-point: 2 end-points (no broadcast or multicast)**
- **Reliable transfer: Data is delivered in-order**
- **Full duplex communication**
- **Byte-stream I/f: sequence of octets**
- **Reliable startup: Data on old connection does not confuse new connections**
- **Graceful shutdown: Data sent before closing a connection is not lost. Reset or immediate shutdown also possible.**

Reliability

- **Reliability provided by:**
  - **Reliable connection startup: Data on old connection does not confuse new connections**
  - **Graceful connection shutdown: Data sent before closing a connection is not lost.**
  - **Data segmented for transmission and acknowledged by destination. Timeout + Retransmission provided if data unacknowledged**
  - **Checksum provided to catch errors.**
  - **Resequencing of out-of-order data; discarding of duplicate data.**
  - **Window flow control => sender cannot overrun receiver buffers**

End-to-end Service



TCP Header Format

Source **Port**	Dest Port	**Seq** No	Ack No	Header length	Resvd	Control	Window
16	16	32	32	4	6	6	16

Check-sum	Urgent	Options	Pad	Data
16	16	x	y	

← Size in ***bits***

Also see fig: 17.2 in text

TCP Header

- **Source Port (16 bits): Identifies source user process**

  20 = FTP, 23 = Telnet, 53 = DNS, 80 = HTTP, ...
- **Destination Port (16 bits)**
- **Sequence Number (32 bits): Sequence number of the first byte in the segment. If SYN is present, this is the initial sequence number (ISN) and the first data byte is ISN+1.**
- **Ack number (32 bits): Next byte expected**

- **Header length (4 bits): Number of 32-bit words in the header. 4 bits => max header size is 60 bytes**
- **Reserved (6 bits)**
- **Control (6 bits)**

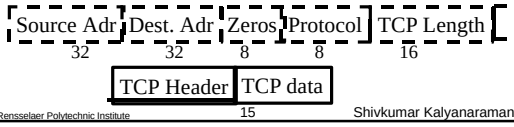
- **Window (16 bits): Will accept [Ack] to [Ack]+[window]**

TCP Header (Cont)

- **Checksum (16 bits): covers the segment + pseudo header. Protection from mis-delivery.**
- **Urgent pointer (16 bits): Points to the byte following urgent data. Lets receiver know how much data it should deliver right away.**
- **Options (variable):**
  Max segment size (does not include TCP header, default 536 bytes), Window scale factor, Selective Ack permitted, Timestamp, No-Op, End-of-options

TCP Checksum

- **Checksum is the 16-bit one's complement of the one's complement sum of a pseudo header, the TCP header, and the data, (padded with zero octets at the end if necessary to make a multiple of two octets.)**
  - **Checksum field filled with zeros initially**
- **Pseudo header (similar to UDP) used in calculations, but not transmitted. RFC 1071.**



Connection Establishment

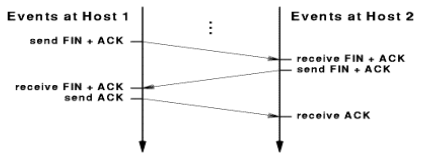
- **Fig 18.3**
- **Client sends SYN, with an initial sequence number (ISN) and a Max Segment Size (MSS). Called** *"active open".*
- **Server acks the SYN (for the *forward connection*), and also sets the SYN bit, with its own ISN (for the *reverse connection*). Called** *"passive open".*
- **Client acks the reverse direction SYN.**
- **3 segments transmitted.**

Connection Termination

- **Fig 18.3 again, also fig 18.5**
- **Client sends FIN. Server acks this and notifies its application. However it can keep its half-connection open. Each connection closed separately.**
- **Server app issues a "close" and server sends FIN to client. Client acks this.**
- **4 segments transmitted.**

Three-Way Handshake

- **3-way handshake: necessary and sufficient for unambiguous setup/teardown even under conditions of loss, duplication, and delay**



More Connection Establishment

- Socket**: BSD term to denote an IP address + a port number.**
  - **A connection is fully specified by a** *socket pair* **i.e. the source IP address, source port, destination IP address, destination port.**
- **Initial Sequence Number (ISN): counter maintained in OS.**
  - **BSD increments it by 64000 every 500ms or new connection setup => time to wrap around < 9.5 hours.**

- **SYN pkt lost => retransmitted. Exponential timeout backoff (6, 12, 24 s etc) Connection timeout is 75 s.**
- **Timer granularity is 500 ms => first timeout between 5.5 and 6s. See Fig. 18.7**

MSS

- **Largest "chunk" sent between TCPs.**
  - **Default = 536 bytes.**
  - **Announced in connection establishment. Not negotiated.**
  - **Different MSS possible for forward/reverse paths.**
  - **Does <u>not</u> include TCP header**
- **Many BSD systems restrict MSS to be multiples of 512 bytes: inefficient.**
- **Path MTU restricts size of MSS further.**

Half close, Half open, Reset

- Possible for one end to close while the other end sends data. Used in "rsh" command. Fig 18.10, 18.11
- Half-open: one side crashed and lost memory of connection while other side thinks connection is open. Usually connection is reset upon communication.
- Reset => used to abort connection. Queued data (if any) is dumped.
- Orderly release => FIN sent after queued data transmitted.

TCP state transition diagram

- Figure 18.12: client (dark line) , server (dashed line) transitions.
- 2MSL wait: wait for final segment to be transmitted before releasing connection (typically 2 min)
  - Socket *pair* cannot be reused during 2MSL
  - Delayed segments dropped
- Conn Establishment: SYN_SENT, SYN_RCVD, ESTABLISHED, LISTEN
- Close: FIN_WAIT_1, FIN_WAIT_2, CLOSING, TIME_WAIT, CLOSE_WAIT, LAST_ACK

Effect of 2MSL wait

- Can't kill server & restart immediately to use the same well known port (1-4 min!)
- Reason: TCP cannot reallocate the socket pair (i.e. the connection) till 2MSL.
- If you kill client and restart, it will get a different port
- 2MSL wait protects against delayed segments from the previous "incarnation" of the connection.
- If server crashes and reboots within 2 MSL wait, it is still safe because RFC 793 prevents having connections for 1 MSL after reboot.

Simultaneous open/close

- **Figs 18.17 and 18.19**
- **Simultaneous open is very rare. Requires same socket** pair **i.e. both the ports must be well known too.**
  - **Two simultaneous telnets (A to B and B to A) will not create this because client ports are not well-known.**
- **Possible in long RTT cases**
- **Requires 4 messages**

TCP Options

Kind	Length	Meaning
0	1	End of Valid options in header
1	1	No-op
2	4	Maximum Segment Size
3	3	Window Scale Factor
8	10	Timestamp

- **End of Options: Stop looking for further option**
- **No-op: Ignore this byte. Used to align the next option on a 4-byte word boundary**

TCP Servers

- **Most TCP servers are *concurrent* i.e. separate process to handle each client - for ease of connection management**
- **Server listens to well-known port.**
  - **Socket pair distinguishes connections**
  - **A separate "endpoint" in the ESTABLISHED state is associated with each connection**
  - **One endpoint is used to listen (LISTEN state) for new connections**

TCP Servers (Contd)

- Endpoints in the ESTABLISHED state cannot receive SYN packets
- Possible to wildcard or select specific interfaces (local IP addresses) to listen to.
- Multiple connection requests => backlog queue of connections established but new process not yet created by server to handle it.
- Queue full => send RESET to new connection requests

Summary

- UDP is connectionless and simple. No flow/error control.
- TCP provides reliable full-duplex connections.
- TCP state diagram, 3-way handshake, Options
- UDP and TCP servers