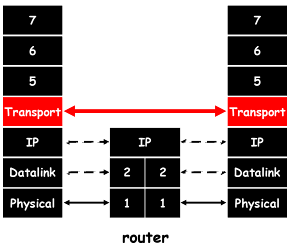
CS640: Introduction to Computer Networks

Aditya Akella

Lecture 14
TCP - I -
Transport Protocols: TCP Segments, Flow control and Connection Setup

Transport Protocols

- Lowest level end-to-end protocol.
  - Header generated by sender is interpreted only by the destination
  - Routers view transport header as part of the payload



Functionality Split

- Network provides best-effort delivery
- End-systems implement many functions
  - Reliability
  - In-order delivery
  - De-multiplexing
  - Message boundaries
  - Connection abstraction
  - Congestion control
  - …

Transport Protocols

- UDP provides just integrity and demux
- TCP adds…
  - Connection-oriented
  - Reliable
  - Ordered
  - Point-to-point
  - Byte-stream
  - Full duplex
  - Flow and congestion controlled
- Request-reply service
  - RPC-like
  - Not covered here

4

UDP: User Datagram Protocol

- "No frills," "bare bones" Internet transport protocol
- "Best effort" service, UDP segments may be:
  - Lost
  - Delivered out of order to app
- *Connectionless:*
  - No handshaking between UDP sender, receiver
  - Each UDP segment handled independently of others

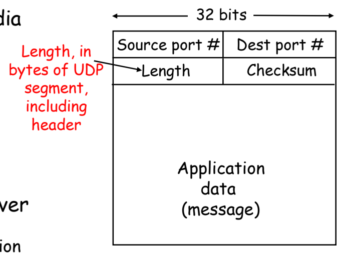
Why is there a UDP?

- No connection establishment (which can add delay)
- Simple: no connection state at sender, receiver
- Small header
- No congestion control: UDP can blast away as fast as desired

5

More on UDP

- Often used for streaming multimedia apps
  - Loss tolerant
  - Rate sensitive
- Other UDP uses (why?):
  - DNS, SNMP
- Reliable transfer over UDP
  - Must be at application layer
  - Application-specific error recovery



UDP segment format

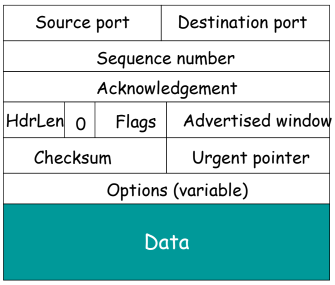
6

TCP

Reliable, Connection oriented,

In-order, Byte stream abstraction

Flags: SYN
FIN
RESET
PUSH
URG
ACK



7

Sequence and Acknowledge Numbers

- Sequence number → byte num of first byte in payload

- Acknowledgement number
  - TCP is full duplex
  - Sequence number of next byte expected in reverse direction

8

Advertised Window

- Used for "flow control"
  - Prevent receing app from getting overwhelmed

- Both sender and receiver advertise window
  - Sender action:
    lastSent – lastACK <= Receiver's advertised window

- Flow control coming up…

9

Sliding Window Again

- Sliding buffer at sender and receiver
  - Packets in transit ≤ sender buffer size
  - Advance when sender and receiver agree packets at beginning have been received
- Receiver has to buffer a packet until all prior packets have arrived
  - Also accommodates slow applications
- Goal: provides reliable, ordered delivery, and flow control
- Same as link layer sliding window algorithm, except that flow control is crucial and challenging

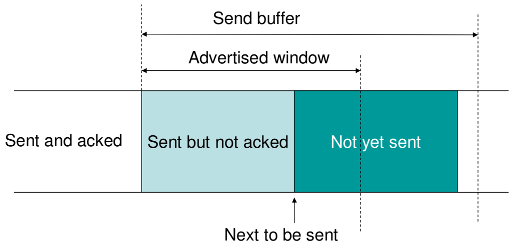
10

TCP Flow Control

- TCP is a sliding window protocol
  - For window size *n*, can send up to *n* bytes without receiving an acknowledgement
  - When the data is acknowledged then the window slides forward
- Each packet advertises a window size
  - Indicates number of bytes the receiver has space for
- Original TCP always sent entire window
  - Congestion control now limits this

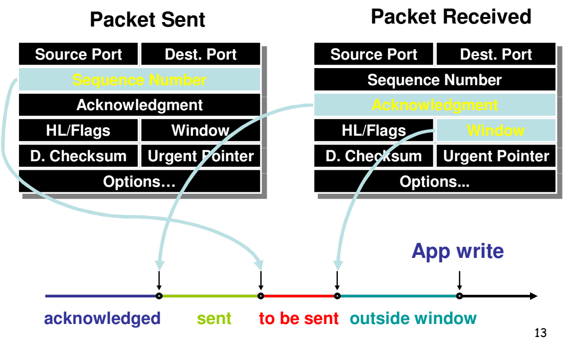
11

Window Flow Control: Send Side



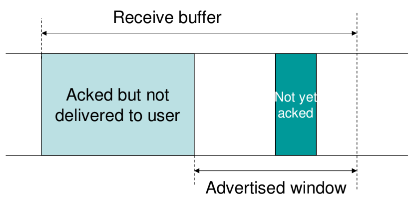
12

Window Flow Control: Send Side



13

Window Flow Control: Receive Side



14

TCP Persist

- What happens if window is 0?
  - Receiver updates window when application reads data
  - What if this update is lost?
- TCP Persist state
  - Sender periodically sends 1 byte packets
  - Receiver responds with ACK even if it can't store the packet

15

Performance Considerations

- The window size can be controlled by receiving application
  - Can change the socket buffer size from a default (e.g. 8Kbytes) to a maximum value (e.g. 64 Kbytes)
- The window size field in the TCP header limits the window that the receiver can advertise
  - 16 bits → 64 KBytes
  - 10 msec RTT → 51 Mbit/second
  - 100 msec RTT → 5 Mbit/second
  - TCP options to get around 64KB limit

16

Sequence Numbers

- How large do sequence numbers need to be?
  - Depends on sender/receiver window size
  - E.g.
    - Max seq = 7, window_size = 7
    - If pkts 0..6 are sent successfully and all acks lost
      - Receiver expects 7,0..5, sender retransmits old 0..6!!!
- Max sequence must be ≥ 2 * window_size
- TCP uses 32 bit sequence numbers
  - Window size limited to 16 bits
  - Sequence number space is ample

17

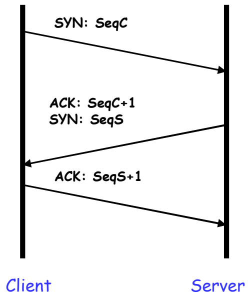
TCP Sequence Numbers

- Sequence Number Space
  - Each *byte in byte stream* is numbered.
  - 32 bit value
  - Wraps around
- Initial values selected at start up time
  - TCP breaks up the byte stream in packets.
- Packet size is limited to the Maximum Segment Size
  - Each packet has a sequence number.
  - Indicates where it fits in the byte stream

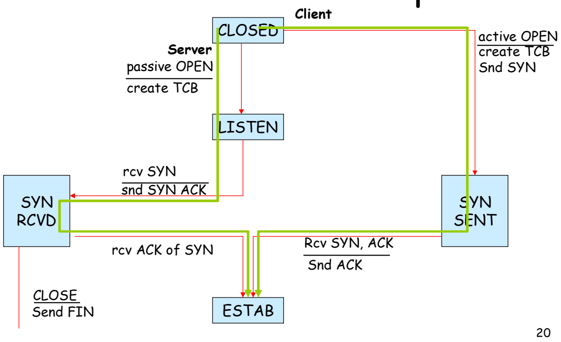
18

Establishing Connection: Three-Way handshake

- Each side notifies other of starting sequence number it will use for sending
  - Why not simply chose 0?
    - Must avoid overlap with earlier incarnation
- Each side acknowledges other's sequence number
  - SYN-ACK: Acknowledge sequence number + 1
- Can combine second SYN with first ACK

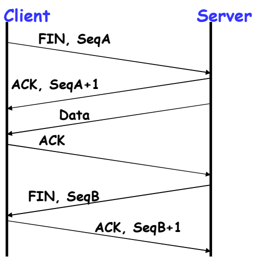


TCP State Diagram: Connection Setup

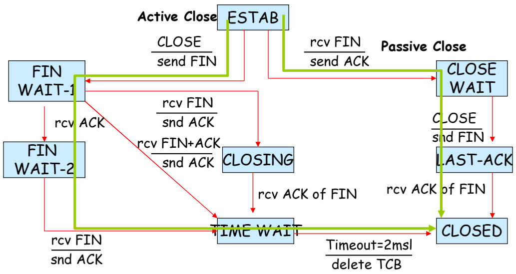


Tearing Down Connection

- Either side can initiate tear down
  - Send FIN signal
  - "I'm not going to send any more data"
- Other side can continue sending data
  - Half open connection
  - Must continue to acknowledge
- Acknowledging FIN
  - Acknowledge last sequence number + 1



State Diagram: Connection Tear-down



Time_Wait state is necessary in case the final ack was lost.

22