

Transport Layer

- Review principles:
 - Reliable data transfer
 - Flow control
 - Congestion control
- Instantiation in the Internet
 - UDP
 - TCP

1

UDP: User Datagram Protocol [RFC 768]

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

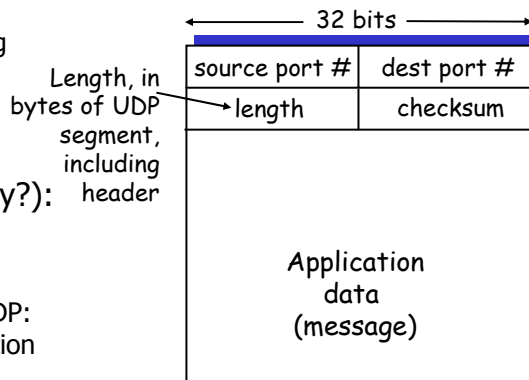
Why is there a UDP?

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

2

UDP: more

- Often used for streaming multimedia apps
 - Loss tolerant
 - Rate sensitive
- Other UDP uses (why?):
 - DNS
 - SNMP
- Reliable transfer over UDP: add reliability at application layer
 - Application-specific error recover!



UDP segment format

3

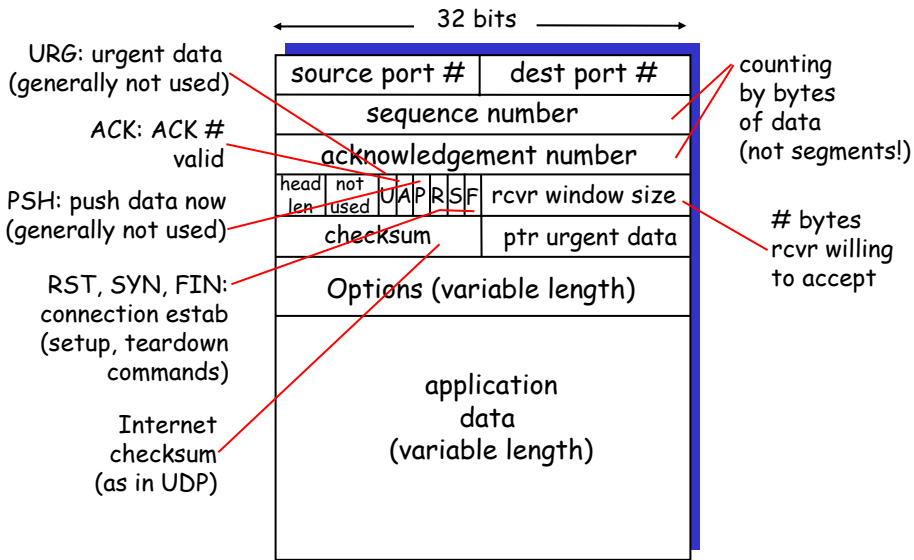
TCP: Overview

RFCs: 793, 1122, 1323, 2018, 2581

- **Point-to-point:**
 - One sender, one receiver
- **Reliable, in-order *byte stream*:**
 - No "message boundaries"
- **Pipelined:**
 - TCP congestion and flow control set window size
- **Full duplex data:**
 - Bi-directional data flow in same connection
 - MSS: maximum segment size
- **Connection-oriented:**
 - Handshaking (exchange of control msgs) init's sender, receiver state before data exchange
- **Flow controlled:**
 - Sender will not overwhelm receiver

4

TCP segment structure



5

TCP seq. #'s and ACKs

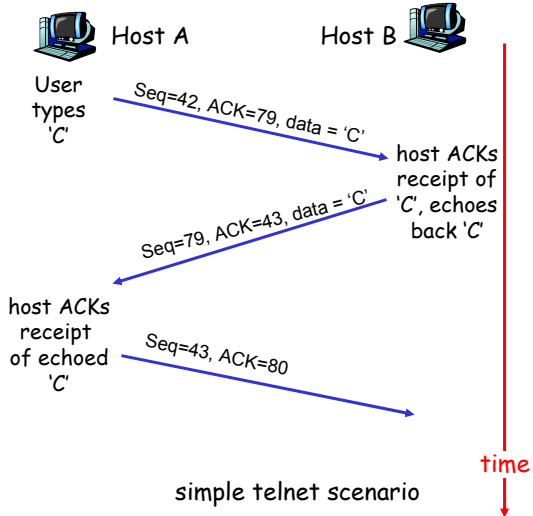
Seq. #'s:

- Byte stream "number" of first byte in segment's data

ACKs:

- Seq # of next byte expected from other side
- Cumulative ACK

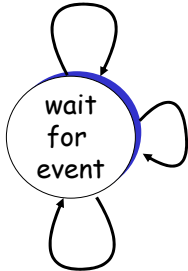
- Q: How receiver handles out-of-order segments
- A: TCP spec doesn't say – up to implementor



6

TCP: reliable data transfer

event: data received from application above
create, send segment



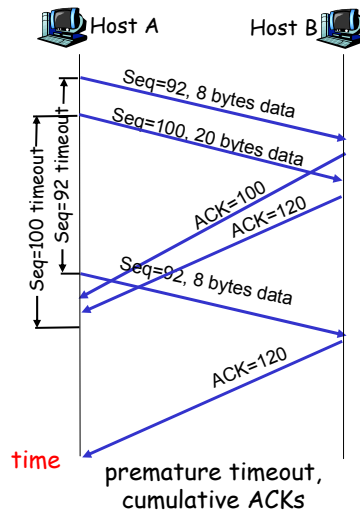
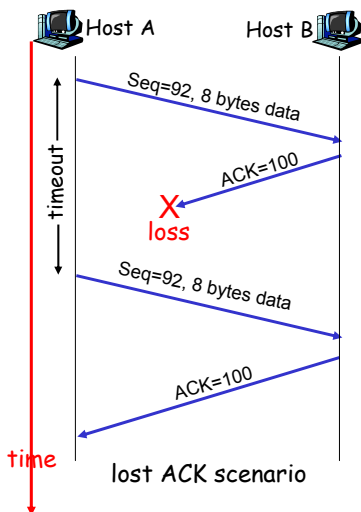
event: timer timeout for segment with seq # y
retransmit segment

event: ACK received, with ACK # y
ACK processing

Simplified sender, assuming

- One way data transfer
- No flow, congestion control

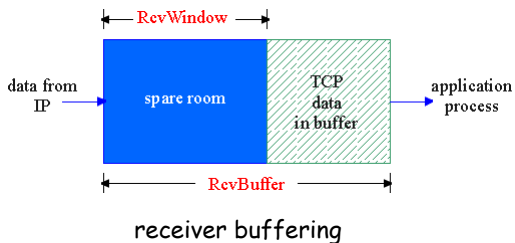
TCP: retransmission scenarios



TCP Flow Control

flow control

sender won't overrun receiver's buffers by transmitting too much, too fast



Receiver: explicitly informs sender of (dynamically changing) amount of free buffer space

- **rcvr window size** field in TCP segment

Sender: amount of transmitted, unACKed data less than most recently-receiver **rcvr window size**

9

TCP Round-trip Time and Timeout Estimation

- Wait at least one RTT before retransmitting
- Importance of accurate RTT estimators:
 - Low RTT → unneeded retransmissions
 - High RTT → poor throughput
- RTT estimator must adapt to change in RTT
 - But not too fast, or too slow!
- Spurious timeouts
 - "Conservation of packets" principle – more than a window worth of packets in flight

13

TCP Connection Management

Recall: TCP sender, receiver establish "connection" before exchanging data segments

- Initialize TCP variables:
 - Seq. #s
 - Buffers, flow control info (e.g. `RcvWindow`)

- *Client:* connection initiator

```
Socket clientSocket = new Socket("hostname", "port number");
```

- *Server:* contacted by client

```
Socket connectionSocket = welcomeSocket.accept();
```

14

TCP Connection Management (2)

Three way handshake:

Step 1: Client end system sends TCP SYN control segment to server

- Specifies initial seq #
- Specifies initial window #

Step 2: Server end system receives SYN, replies with SYNACK control segment

- ACKs received SYN
- Allocates buffers
- Specifies server → receiver initial seq. #
- Specifies initial window #

Step 3: Client system receives SYNACK

15

TCP Connection Management (3)

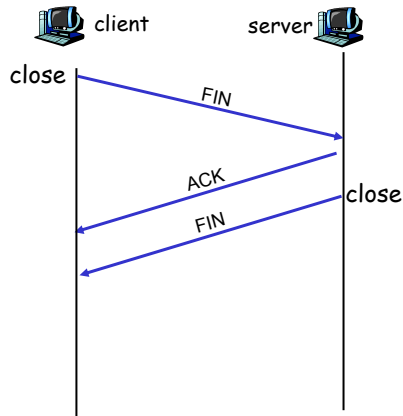
Closing a connection:

Client closes socket:

```
clientSocket.close();
```

Step 1: Client end system sends TCP FIN control segment to server

Step 2: Server receives FIN, replies with ACK. Closes connection, sends FIN.



16

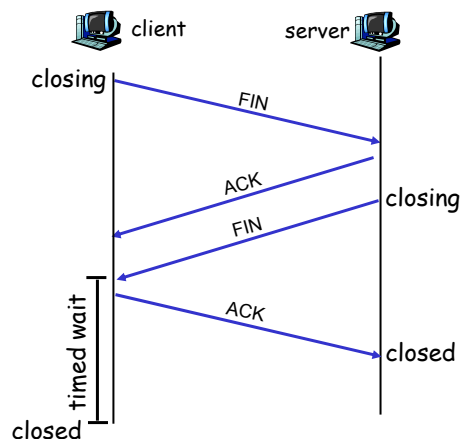
TCP Connection Management (4)

Step 3: Client receives FIN, replies with ACK.

- Enters "timed wait" – will respond with ACK to received FINs

Step 4: Server, receives ACK. Connection closed.

Note: With small modification, this can handle simultaneous FINs.



17

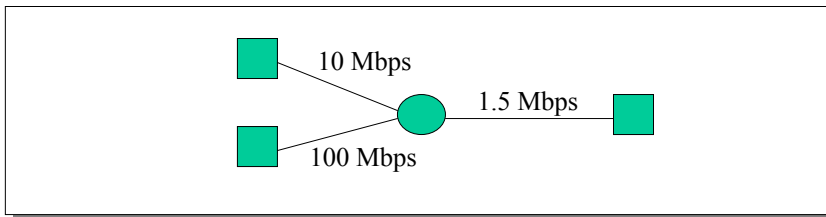
Principles of Congestion Control

Congestion:

- ❑ Informally: “too many sources sending too much data too fast for *network* to handle”
- ❑ Different from flow control!
- ❑ Manifestations:
 - Lost packets (buffer overflow at routers)
 - Long delays (queueing in router buffers)

19

Congestion



- ❑ Different sources compete for resources inside network
- ❑ Why is it a problem?
 - Sources are unaware of current state of resource
 - Sources are unaware of each other
 - In many situations will result in < 1.5 Mbps of throughput (congestion collapse)

20

Congestion Collapse

- ❑ Definition: *Increase in network load results in decrease of useful work done*
- ❑ Many possible causes
 - Spurious retransmissions of packets still in flight
 - Classical congestion collapse
 - How can this happen with packet conservation
 - Solution: better timers and TCP congestion control
 - Undelivered packets
 - Packets consume resources and are dropped elsewhere in network
 - Solution: congestion control for ALL traffic

21

Where to Prevent Collapse?

- ❑ Can end hosts prevent problem?
 - Yes, but must trust end hosts to do right thing
 - E.g., sending host must adjust amount of data it puts in the network based on detected congestion
- ❑ Can routers prevent collapse?
 - No, not all forms of collapse
 - Doesn't mean they can't help
 - Sending accurate congestion signals
 - Isolating well-behaved from ill-behaved sources

22

Congestion Control and Avoidance

- A mechanism which:
 - Uses network resources efficiently
 - Preserves fair network resource allocation
 - Prevents or avoids collapse
- Congestion collapse is not just a theory
 - Has been frequently observed in many networks

23

Approaches towards congestion control

End-end congestion control:

- No explicit feedback from network
- Congestion inferred from end-system observed loss, delay
- Approach taken by TCP

Network-assisted congestion control:

- Routers provide feedback to end systems
 - Choke packet router to sender
 - Single bit indicating congestion (SNA, DECbit, TCP/IP ECN, ATM)
 - Explicit rate sender should send at

24

TCP congestion control:

- “probing” for usable bandwidth:
 - Ideally: transmit as fast as possible (*Congwin* as large as possible) without loss
 - Increase *Congwin* until loss (congestion)
 - Loss: decrease *Congwin*, then begin probing (increasing) again
- Two “phases”
 - Slow start
 - Congestion avoidance
- Important variables:
 - *Congwin*
 - **threshold**: defines threshold between two slow start phase, congestion control phase

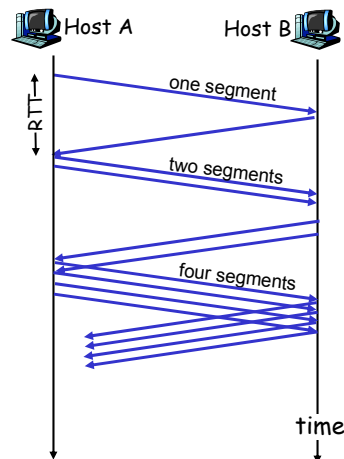
25

TCP Slowstart

- Exponential increase (per RTT) in window size (not so slow!)
- Loss event: timeout (Tahoe TCP) and/or or three duplicate ACKs (Reno TCP)

Slowstart algorithm

initialize: $Congwin = 1$
for (each segment ACKed)
 $Congwin++$
until (loss event OR
 $CongWin > threshold$)



26

Congestion Avoidance

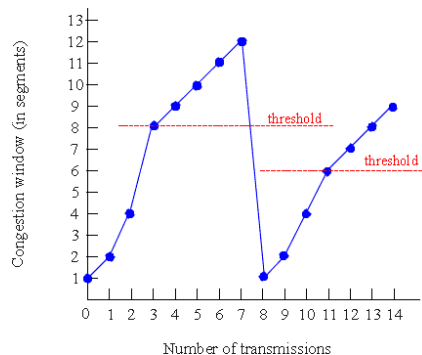
- Loss implies congestion – why?
 - Not necessarily true on all link types
- If loss occurs when $cwnd = W$
 - Network can handle $0.5W \sim W$ segments
 - Set $cwnd$ to $0.5W$ (multiplicative decrease)
- Upon receiving ACK
 - Increase $cwnd$ by $1/cwnd$
 - Results in additive increase

27

TCP Congestion Avoidance

Congestion avoidance

```
/* slowstart is over */
/* Congwin > threshold */
Until (loss event) {
  every w segments ACKed:
    Congwin++
}
threshold = Congwin/2
Congwin = 1
perform slowstart1
```



1: TCP Reno skips slowstart (fast recovery) after three duplicate ACKs

28

Return to Slow Start

- If packet is lost we lose self clocking
 - Need to implement slow-start and congestion avoidance together
- When timeout occurs set *ssthresh* to $0.5w$
 - If *cwnd* < *ssthresh*, use slow start
 - Else use congestion avoidance

29

TCP Congestion Control

- End-end control (no network assistance)
- TCP throughput limited by rcvr window (flow control)
- Transmission rate limited by congestion window size, *Congwin*, over segments:



- w segments, each with MSS bytes sent in one RTT

30

Summary

- Reviewed principles of transport layer:
 - Reliable data transfer
 - Flow control
 - Congestion control
 - (Multiplexing)
- Instantiation in the Internet
 - UDP
 - TCP