Chapter 3
Transport Layer

A note on the use of these ppt slides:
We’re making these slides freely available to all (faculty, students, readers). They’re in PowerPoint form so you can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a lot of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) in substantially unaltered form, that you mention their source (after all, we’d like people to use our book!)
- If you post any slides in substantially unaltered form on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

Thanks and enjoy! JFK/KWR

All material copyright 1996-2010
J.F Kurose and K.W. Ross, All Rights Reserved

Computer Networking:
A Top Down Approach
5th edition.
Jim Kurose, Keith Ross
Addison-Wesley, April 2009.
Chapter 3 outline

3.1 Transport-layer services
3.2 Multiplexing and demultiplexing
3.3 Connectionless transport: UDP
3.4 Principles of reliable data transfer

3.5 Connection-oriented transport: TCP
   - segment structure
   - reliable data transfer
   - flow control
   - connection management

3.6 Principles of congestion control
3.7 TCP congestion control
TCP reliable data transfer

- TCP creates rdt service on top of IP’s unreliable service
- pipelined segments
- cumulative acks
- TCP uses single retransmission timer

- retransmissions are triggered by:
  - timeout events
  - duplicate acks

- initially consider simplified TCP sender:
  - ignore duplicate acks
  - ignore flow control, congestion control
TCP sender events:

**data rcvd from app:**
- Create segment with seq #
- seq # is byte-stream number of first data byte in segment
- start timer if not already running (think of timer as for oldest unacked segment)
- expiration interval: TimeOutInterval

**timeout:**
- retransmit segment that caused timeout
- restart timer

**Ack rcvd:**
- If acknowledges previously unacked segments
  - update what is known to be acked
  - start timer if there are outstanding segments
NextSeqNum = InitialSeqNum
SendBase = InitialSeqNum

loop (forever) {
  switch(event)
  
  event: data received from application above
  create TCP segment with sequence number NextSeqNum
  if (timer currently not running)
    start timer
  pass segment to IP
  NextSeqNum = NextSeqNum + length(data)

  event: timer timeout
  retransmit not-yet-acknowledged segment with smallest sequence number
  start timer

  event: ACK received, with ACK field value of y
  if (y > SendBase) {
    SendBase = y
    if (there are currently not-yet-acknowledged segments)
      start timer
  }

} /* end of loop forever */
TCP: retransmission scenarios

Host A
Seq=100, 20 bytes data

Host B
Seq=92, 8 bytes data

SendBase = 100

SendBase = 120

Seq=92, 8 bytes data

ACK=100

ACK=100

Seq=92 timeout

X

loss

timeout

X

lost ACK scenario

Host A
Seq=92, 8 bytes data

Host B
Seq=100, 20 bytes data

SendBase = 100

SendBase = 120

SendBase = 120

Seq=92, 8 bytes data

ACK=100

ACK=120

Seq=92 timeout

premature timeout

time

timeout

time
TCP retransmission scenarios (more)

Host A
Seq=92, 8 bytes data
ACK=100

Host B
Seq=100, 20 bytes data
ACK=120

SendBase = 120

Timeout

Cumulative ACK scenario
# TCP ACK generation

**[RFC 1122, RFC 2581]**

<table>
<thead>
<tr>
<th>Event at Receiver</th>
<th>TCP Receiver action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arrival of in-order segment with expected seq #. All data up to expected seq # already ACKed</td>
<td>Delayed ACK. Wait up to 500ms for next segment. If no next segment, send ACK</td>
</tr>
<tr>
<td>Arrival of in-order segment with expected seq #. One other segment has ACK pending</td>
<td>Immediately send single cumulative ACK, ACKing both in-order segments</td>
</tr>
<tr>
<td>Arrival of out-of-order segment higher-than-expect seq. #. Gap detected</td>
<td>Immediately send <em>duplicate ACK</em>, indicating seq. # of next expected byte</td>
</tr>
<tr>
<td>Arrival of segment that partially or completely fills gap</td>
<td>Immediate send ACK, provided that segment starts at lower end of gap</td>
</tr>
</tbody>
</table>
Fast Retransmit

- time-out period often relatively long:
  - long delay before resending lost packet
- detect lost segments via duplicate ACKs.
  - sender often sends many segments back-to-back
  - if segment is lost, there will likely be many duplicate ACKs.
- if sender receives 3 ACKs for the same data, it supposes that segment after ACKed data was lost:
  - fast retransmit: resend segment before timer expires
Figure 3.37 Resending a segment after triple duplicate ACK
**Fast retransmit algorithm:**

**event:** ACK received, with ACK field value of $y$

  - if ($y > \text{SendBase}$)
    - $\text{SendBase} = y$
      - if (there are currently not-yet-acknowledged segments)
        - start timer
    
  - else
    - increment count of dup ACKs received for $y$
      - if (count of dup ACKs received for $y = 3$)
        - resend segment with sequence number $y$
Chapter 3 outline

3.1 Transport-layer services
3.2 Multiplexing and demultiplexing
3.3 Connectionless transport: UDP
3.4 Principles of reliable data transfer

- 3.5 Connection-oriented transport: TCP
  - segment structure
  - reliable data transfer
  - flow control
  - connection management

3.6 Principles of congestion control
3.7 TCP congestion control
TCP Flow Control

- receive side of TCP connection has a receive buffer:
  - flow control: sender won’t overflow receiver’s buffer by transmitting too much, too fast
  - speed-matching service: matching the send rate to the receiving app’s drain rate

- app process may be slow at reading from buffer
TCP Flow control: how it works

(suppose TCP receiver discards out-of-order segments)

- spare room in buffer
  - \( RcvWindow \) = \( RcvBuffer - [LastByteRcvd - LastByteRead] \)

- rcvr advertises spare room by including value of \( RcvWindow \) in segments
- sender limits unACKed data to \( RcvWindow \)
  - guarantees receive buffer doesn’t overflow
Chapter 3 outline

3.1 Transport-layer services
3.2 Multiplexing and demultiplexing
3.3 Connectionless transport: UDP
3.4 Principles of reliable data transfer
3.5 Connection-oriented transport: TCP
  ▪ segment structure
  ▪ reliable data transfer
  ▪ flow control
  ▪ connection management
3.6 Principles of congestion control
3.7 TCP congestion control
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
  
  ```java
  Socket clientSocket = new Socket("hostname","port number");
  ```
- **server:** contacted by client
  
  ```java
  Socket connectionSocket = welcomeSocket.accept();
  ```

**Three way handshake:**

**Step 1:** client host sends TCP SYN segment to server

- specifies initial seq #
- no data

**Step 2:** server host receives SYN, replies with SYNACK segment

- server allocates buffers
- specifies server initial seq. #

**Step 3:** client receives SYNACK, replies with ACK segment, which may contain data
TCP Connection Management (cont.)

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.
**TCP Connection Management (cont.)**

**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, can handle simultaneous FINs.
TCP Connection Management (cont)

TCP server lifecycle
- server application creates a listen socket
- receive SYN, send SYN & ACK

TCP client lifecycle
- wait 30 seconds
- receive FIN, send ACK
- receive ACK, send nothing
- receive ACK, send nothing
- receive FIN, send ACK
- close connection