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
Chapter 3: Transport Layer

Our goals:
- understand principles behind transport layer services:
  - multiplexing/demultiplexing
  - reliable data transfer
  - flow control
  - congestion control
- learn about transport layer protocols in the Internet:
  - UDP: connectionless transport
  - TCP: connection-oriented transport
  - TCP congestion control
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
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 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 segment header
- no congestion control: UDP can blast away as fast as desired
**UDP: more**

- often used for streaming multimedia apps
  - loss tolerant
  - rate sensitive
- other UDP uses
  - DNS
  - SNMP
- reliable transfer over UDP: add reliability at application layer
  - application-specific error recovery!

**UDP segment format**

<table>
<thead>
<tr>
<th>Source Port #</th>
<th>Destination Port #</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length</td>
<td>Checksum</td>
</tr>
</tbody>
</table>

Length, in bytes of UDP segment, including header

Application data (message)

UDP segment format
**UDP checksum**

**Goal:** detect “errors” (e.g., flipped bits) in transmitted segment

**Sender:**
- treat segment contents as sequence of 16-bit integers
- checksum: addition (1’s complement sum) of segment contents
- sender puts checksum value into UDP checksum field

**Receiver:**
- compute checksum of received segment
- check if computed checksum equals checksum field value:
  - NO - error detected
  - YES - no error detected. *But maybe errors nonetheless? More later* ....
Internet Checksum Example

- Note: when adding numbers, a carryout from the most significant bit needs to be added to the result
- Example: add two 16-bit integers

\[
\begin{array}{cccccccccccccccccccc}
1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\
1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
\hline
1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\
\end{array}
\]

wraparound

\[
\begin{array}{cccccccccccccccccccc}
1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\
\hline
\end{array}
\]

sum

\[
\begin{array}{cccccccccccccccccccc}
1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\
\end{array}
\]

checksum
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
   - Bit error: Ack, seq.#
   - Loss: Time out
   - Pipelining
   - Selective Repeat
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
Principles of Reliable data transfer

- Important in app., transport, link layers
- Top-10 list of important networking topics!

- Characteristics of unreliable channel will determine complexity of reliable data transfer protocol (rdt)

(a) provided service
Principles of Reliable data transfer

- important in app., transport, link layers
- top-10 list of important networking topics!

- characteristics of unreliable channel will determine complexity of reliable data transfer protocol (rdt)
Principles of Reliable data transfer

- important in app., transport, link layers
- top-10 list of important networking topics!

- characteristics of unreliable channel will determine complexity of reliable data transfer protocol (rdt)
Reliable data transfer: getting started

- **rdt_send()**: called from above, (e.g., by app.). Passed data to deliver to receiver upper layer.

- **deliver_data()**: called by rdt to deliver data to upper.

- **udt_send()**: called by rdt, to transfer packet over unreliable channel to receiver.

- **rdt_rcv()**: called when packet arrives on rcv-side of channel.
Reliable data transfer: getting started

We’ll:

- incrementally develop sender, receiver sides of reliable data transfer protocol (rdt)
- consider only unidirectional data transfer
  - but control info will flow on both directions!
- use finite state machines (FSM) to specify sender, receiver

**State**: when in this “state” next state uniquely determined by next event

**Event** causing state transition

**Actions** taken on state transition
Rdt1.0: **reliable transfer over a reliable channel**

- underlying channel perfectly reliable
  - no bit errors
  - no loss of packets
- separate FSMs for sender, receiver:
  - sender sends data into underlying channel
  - receiver read data from underlying channel
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
   - Bit error: Ack, seq.#
   - Loss: Time out
   - Pipelining
   - Selective Repeat
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
Rdt2.0: *channel with bit errors*

- underlying channel may flip bits in packet
  - checksum to detect bit errors
- the question: how to recover from errors:

*How do humans recover from “errors” during conversation?*
Rdt2.0: channel with bit errors

- underlying channel may flip bits in packet
  - checksum to detect bit errors

- the question: how to recover from errors:
  - acknowledgements (ACKs): receiver explicitly tells sender that pkt received OK
  - negative acknowledgements (NAKs): receiver explicitly tells sender that pkt had errors
  - sender retransmits pkt on receipt of NAK

- new mechanisms in rdt2.0 (beyond rdt1.0):
  - error detection
  - receiver feedback: control msgs (ACK, NAK) rcvr->sender
**rdt2.0: FSM specification**

sender

\[
\text{rdt\_send}(\text{data}) \\
\text{sndpkt} = \text{make\_pkt}(\text{data}, \text{checksum}) \\
\text{udt\_send}(\text{sndpkt}) \\
\text{rdt\_rcv}(\text{rcvpkt}) \land \text{is\_ACK}(\text{rcvpkt}) \\
\land
\]

receiver

\[
\text{rdt\_rcv}(\text{rcvpkt}) \land \text{is\_NAK}(\text{rcvpkt}) \\
\text{udt\_send}(\text{sndpkt}) \\
\text{rdt\_rcv}(\text{rcvpkt}) \\
\text{udt\_send}(\text{NAK}) \\
\text{Wait for call from below} \\
\text{rdt\_rcv}(\text{rcvpkt}) \land \text{not\_corrupt}(\text{rcvpkt}) \\
\text{extract}(\text{rcvpkt}, \text{data}) \\
\text{deliver\_data}(\text{data}) \\
\text{udt\_send}(\text{ACK})
\]
**rdt2.0: operation with no errors**

```
rdt_send(data)

snkpkt = make_pkt(data, checksum)
udt_send(snkpkt)

wait for call from above

rdt_rcv(rcvpkt) && isACK(rcvpkt)
udt_send(snkpkt)

wait for ACK or NAK

rdt_rcv(rcvpkt) && isNAK(rcvpkt)
udt_send(snkpkt)

corrupt(rcvpkt)
udt_send(NAK)

wait for ACK or NAK

extract(rcvpkt, data)
deliver_data(data)
udt_send(ACK)
```
**rdt2.0: error scenario**

```
rdt_send(data)
snkpkt = make_pkt(data, checksum)
udt_send(sndpkt)
```

```
wait for call from above
```

```
rdt_rcv(rcvpkt) &&
isNAK(rcvpkt)
udt_send(sndpkt)
```

```
wait for ACK or NAK
```

```
rdt_send(data)
```

```
wait for call from below
```

```
rdt_send(NAK)
```

```
rdt_send(ACK)
```

```
extract(rcvpkt, data)
deliver_data(data)
```

```
notcorrupt(rcvpkt)
```

```
```
```
```
```
```
```
```
```
rdt2.0 has a fatal flaw!
rdt2.0 has a fatal flaw!

What happens if ACK/NAK corrupted?
- sender doesn’t know what happened at receiver!
- can’t just retransmit: possible duplicate

Handling duplicates:
- sender retransmits current pkt if ACK/NAK garbled
- sender adds sequence number to each pkt
- receiver discards (doesn’t deliver up) duplicate pkt

stop and wait
Sender sends one packet, then waits for receiver response