Reading Quiz
Last class

- We are at the transport-layer protocol!
  - provide services to the application layer
  - interact with n/w layer below that provides routing
- UDP: User Datagram Protocol
  - No frills transport protocol
  - Simple 8-byte header with ports, length, checksum
- Checksum protects against most bit-flips
  - one’s complement of the total sum of 16-bit integers (data + payload)
Layering and encapsulation
Today

• Principles of reliability
  – The Two Generals Problem
• Automatic Repeat Requests
  – Stop and Wait
  – Timeouts and Losses
  – Pipelined Transmission
The Two Generals Problem

- Two army divisions (blue) surround enemy (red)
  - Each division led by a general
  - Both must agree when to simultaneously attack
  - If either side attacks alone, defeat
- Generals can only communicate via messengers
  - Messengers may get captured (unreliable channel)
The Two Generals Problem

- How to coordinate?
  - Send messenger: “Attack at dawn”
  - What if messenger doesn’t make it?
The Two Generals Problem

- How to be sure messenger made it?
  - Send acknowledgment: “I delivered message”
In the “two generals problem”, can the two armies reliably coordinate their attack? (using what we just discussed)

- A. Yes (explain how)
- B. No (explain why not)
The Two Generals Problem

- Result
  - Can’t create perfect channel out of faulty one
  - Can only increase probability of success
Give up? No way!

• As humans, we like to face difficult problems.
  – We can’t control oceans, but we can build canals
  – We can’t fly, but we’ve landed on the moon
  – We just need engineering!

(Unsinkable)
Engineering

• Concerns
  – Message corruption
  – Message duplication
  – Message loss
  – Message reordering
  – Performance

• Our toolbox
  – Checksums
  – Timeouts
  – Acks & Nacks
  – Sequence numbering
  – Pipelining
Engineering

• Concerns
  – Message corruption
  – Message duplication
  – Message loss
  – Message reordering
  – Performance

• Our toolbox
  – Checksums
  – Timeouts
  – Acks & Nacks
  – Sequence numbering
  – Pipelining

We use these to build Automatic Repeat Request (ARQ) protocols.

(We’ll briefly talk about alternatives at the end.)
Automatic Repeat Request (ARQ)

• Intuitively, ARQ protocols act like you would when using a cell phone with bad reception.
  – Message garbled? Ask to repeat.
  – Didn’t hear a response? Speak again.

• Refer to book for building state machines.
  – We’ll look at TCP’s states soon
ARQ Broad Classifications

1. Stop-and-wait
We have:
• a sender
• a receiver
• time: represented by downwards arrow
Stop and Wait

Sender sends data and waits till they get the response message from the receiver.
Stop and Wait

• Up next: concrete problems and mechanisms to solve them.
• These mechanisms will build upon each other
• Questions?
Corruption?

• Error detection mechanism: checksum
  – Data good – receiver sends back ACK
  – Data corrupt – receiver sends back NACK
Could we do this with just ACKs or just NACKs?

Error detection mechanism: checksum

- Data good – receiver sends back ACK
- Data corrupt – receiver sends back NACK

A. No, we need them both.
B. Yes, we could do without one of them, but we’d need some other mechanism.
C. Yes, we could get by without one of them.
Could we do this with just ACKs or just NACKs?

- With only ACK, we could get by with a timeout.
- With only NACK, we couldn’t advance (no good).

A. No, we need them both.
B. Yes, we could do without one of them, but we’d need some other mechanism.
C. Yes, we could get by without one of them.
Timeouts and Losses

- Sender starts a clock. If no response, retry.
Timeouts and Losses

- Sender starts a clock. If no response, retry.
Timeouts and Losses

- Sender starts a clock. If no response, retry.
- Probably not a great idea for handling corruption, but it works.
Timeouts and Losses

- Timeouts help us handle message losses too!
Timeouts and Losses

• Timeouts help us handle message losses too!
Adding timeouts might create new problems for us to worry about. How many? Examples?

A. No new problems (why not?)
B. One new problem (which is..)
C. Two new problems (which are..)
D. More than two new problems (which are..)
Adding timeouts might create new problems for us to worry about. How many? Examples?

A. No new problems (why not?)
B. One new problem (which is..)
C. Two new problems (which are..)
D. More than two new problems (which are..)
Sequence Numbering

**Sender**
- Add a monotonically increasing label to each msg

**Receiver**
- Ignore messages with numbers we’ve seen before
- When pipelining (a few slides from now)
  - Detect gaps in the sequence (e.g., 1,2,4,5)
What is our link utilization with a stop-and-wait protocol?

System parameters:
Link rate: 8 Mbps (one megabyte per second)
RTT: 100 milli-seconds
Segment size: 1024 bytes

A. < 0.1 %
B. ≈ 0.1 %
C. ≈ 1 %
D. 1-10 %
E. > 10 %
What is our link utilization with a stop-and-wait protocol?

System parameters:
Link rate: 8 Mbps (one megabyte per second)
RTT: 100 milli-seconds
Segment size: 1024 bytes

A. < 0.1 %
B. ≈ 0.1 %
C. ≈ 1 %
D. 1-10 %
E. > 10 %

Big Problem: Performance is determined by RTT, not channel capacity!
Pipelined Transmission

- Keep multiple segments “in flight”
  - Allows sender to make efficient use of the link
  - Sequence numbers ensure receiver can distinguish segments
  - We’ll talk about “how many” next time (windowing).
Keep multiple segments “in flight”

- Allows sender to make efficient use of the link
- Sequence numbers ensure receiver can distinguish segments
- We’ll talk about “how many” next time (windowing).
What should the sender do here?

What information does the sender need to make that decision?
What is required by either party to keep track?

A. Start sending all data again from 0.
B. Start sending all data again from 2.
C. Resend just 2, then continue with 4 afterwards.
ARQ Broad Classifications

1. Stop-and-wait

2. Go-back-N
Go-Back-N

- Retransmit from point of loss
  - Segments between loss event and retransmission are ignored
  - “Go-back-N” if a timeout event occurs
Go-Back-N

- Retransmit from point of loss
  - Segments between loss event and retransmission are ignored
  - “Go-back-N” if a timeout event occurs
Go-Back-N

Sender

Receiver

Time

Data-0
Data-1
Data-2
Ack-0
Data-3
...

Lecture 16 - Slide 38
Go-Back-N

Sender

Receiver

Data-0
Data-1
Data-2
Ack-0
Ack-1
Data-3
Data-4

Time

...
Go-Back-N

Sender

Receiver

<table>
<thead>
<tr>
<th>Time</th>
<th>Sender</th>
<th>Receiver</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Data-0</td>
<td>Ack-0</td>
</tr>
<tr>
<td></td>
<td>Data-1</td>
<td>Ack-1</td>
</tr>
<tr>
<td></td>
<td>Data-2</td>
<td>Ack-1</td>
</tr>
<tr>
<td></td>
<td>Ack-0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Data-3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Data-4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ack-1</td>
<td></td>
</tr>
</tbody>
</table>

...
Go-Back-N

Sender

Receiver

Time

Data-0
Data-1
Data-2
Ack-0
Ack-1
Ack-1
Ack-1
Ack-1

...
Go-Back-N

Time

Sender

Receiver

Data-0
Data-1
Data-2
Ack-0
Ack-1
Ack-1
Ack-1
Data-2
Data-3
Data-4
Data-3
Data-4
...

Timeout

Timeout
Go-Back-N

- Retransmit from point of loss
  - Segments between loss event and retransmission are ignored
  - “Go-back-N” if a timeout event occurs
- Fast retransmit
  - Don’t wait for timeout if we get N duplicate ACKs
Go-Back-N

- Retransmit from point of loss
  - Segments between loss event and retransmission are ignored
  - “Go-back-N” if a timeout event occurs
- Fast retransmit
  - Don’t wait for timeout if we get N duplicate ACKs
ARQ Broad Classifications

1. Stop-and-wait

2. Go-back-N

3. Selective repeat
   • a.k.a selective reject, selective acknowledgement
Selective Repeat

- Receiver ACKs each segment individually (not cumulative)
- Sender only resends those not ACKed
- Requires extra buffering and state on the receiver
Selective Repeat

Sender

Receiver

Time

Data-0
Data-1
Data-2
Ack-0
Ack-1

...
Selective Repeat

Sender

Data-0
Data-1
Data-2

Receiver

Ack-0
Ack-1
Ack-2

Data-3
Data-4

Ack-3
Ack-4

...
Selective Repeat

Sender

Receiver

Data-0
Data-1
Data-2

Ack-0
Ack-1

Data-3
Data-4

Ack-3
Ack-4

Data-5
Data-6

...
Selective Repeat

- Receiver ACKs each segment individually (not cumulative)
- Sender only resends those not ACKed
- Requires extra buffering and state on the receiver
ARQ Alternatives

• Can’t afford the RTT’s or timeouts?
• When?
  – Broadcasting, with lots of receivers
  – Very lossy or long-delay channels (e.g., space)
• Use redundancy – send more data
  – Simple form: send the same message N times
  – More efficient: use “erasure coding”
  – For example, encode your data in 10 pieces such that the receiver can piece it together with any subset of size 8.