CPSC 317 COMPUTER NETWORKING

2023W2: Transport – Day 3 – Lost Segments and Timeouts

READING

- Reading: 3.4.1, 3.5.3

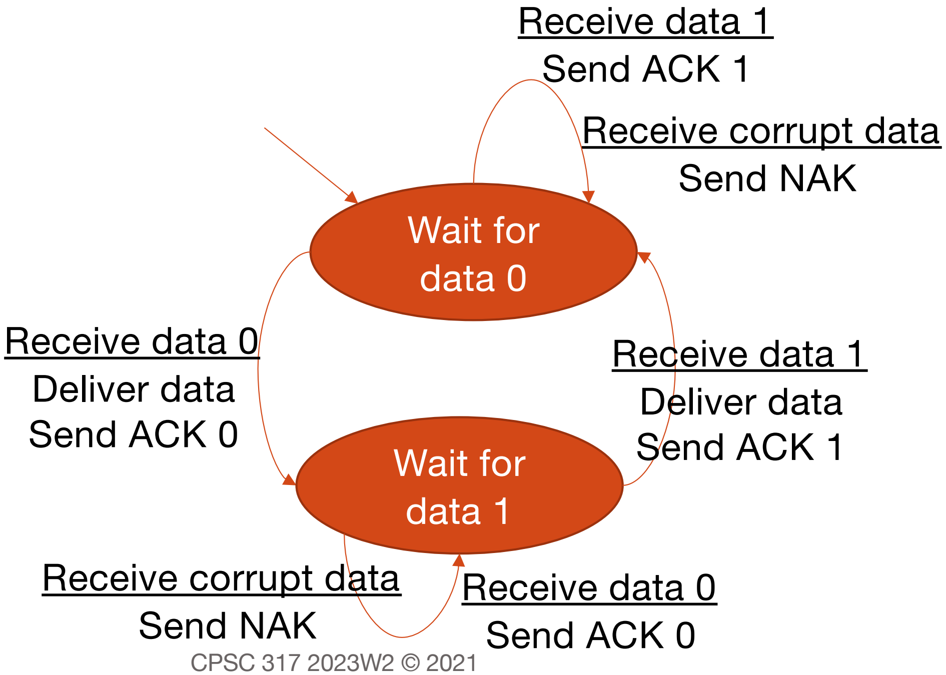
LEARNING GOALS

Alternating Bit Protocol

- Describe/build/execute/trace the various versions of the alternating bit protocol (stop and wait) protocol
- Explain how corruption in segments are detected and at what layer(s) in the protocol stack this is done
- Explain how data loss is handled by a reliable delivery protocol
- For a given scenario determine an appropriate timeout value

REVIEW: ALTERNATING BIT PROTOCOL

Receiver



Receive data 1
Send ACK 1

Receive corrupt data
Send NAK

Wait for data 0

Receive data 0
Deliver data
Send ACK 0

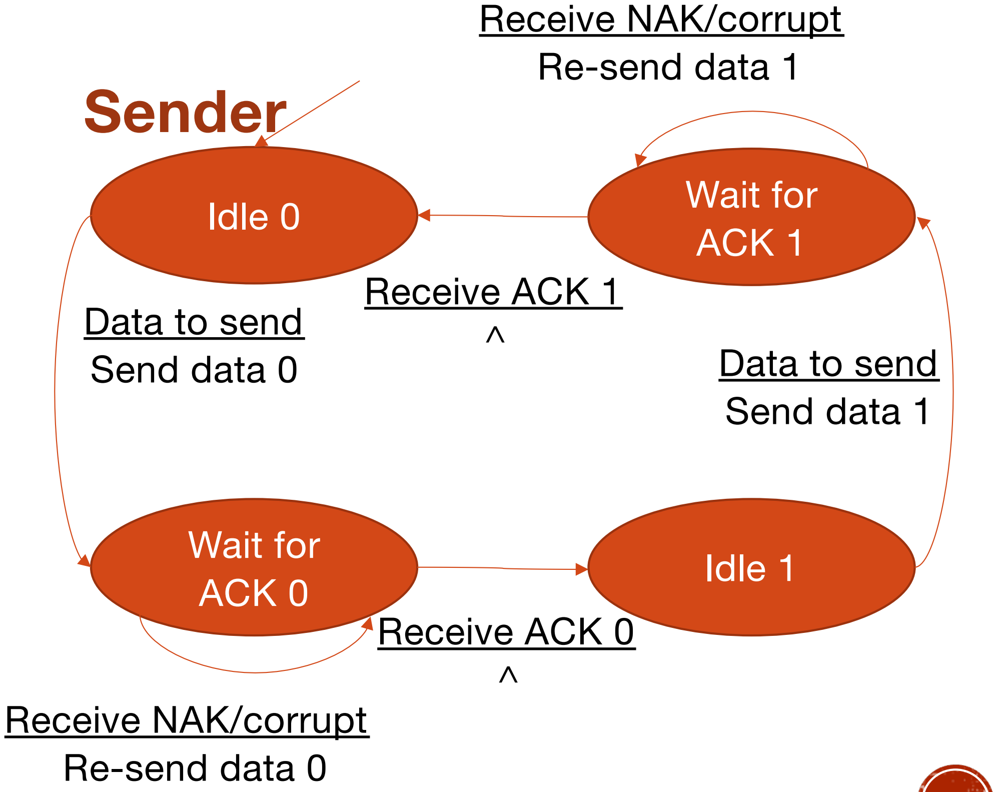
Receive data 1
Deliver data
Send ACK 1

Wait for data 1

Receive corrupt data
Send NAK

Receive data 0
Send ACK 0

Sender



Receive NAK/corrupt
Re-send data 1

Idle 0

Wait for ACK 1

Receive ACK 1
Λ

Data to send
Send data 0

Data to send
Send data 1

Wait for ACK 0

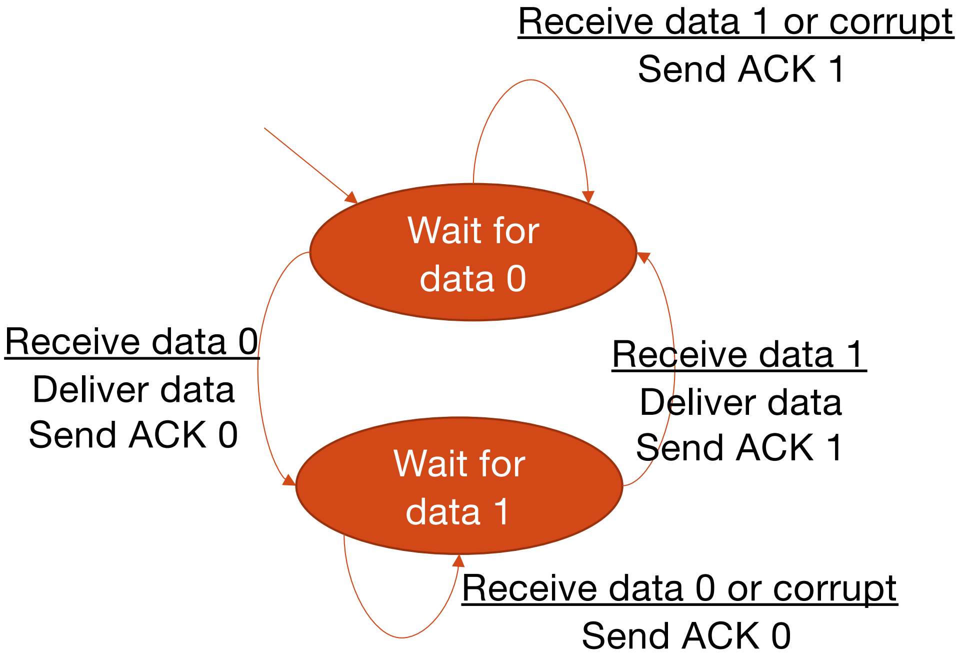
Idle 1

Receive ACK 0
Λ

Receive NAK/corrupt
Re-send data 0

REVIEW: ALTERNATING BIT PROTOCOL (NO NAK)

Receiver



Wait for data 0

Wait for data 1

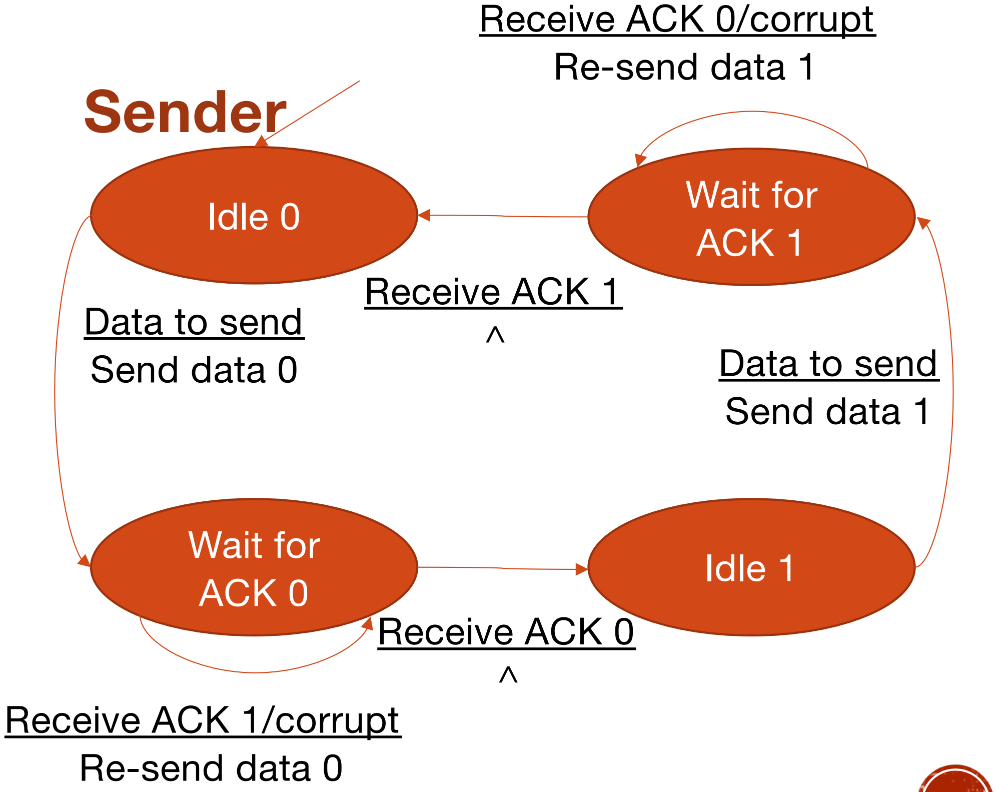
Receive data 1 or corrupt
Send ACK 1

Receive data 0
Deliver data
Send ACK 0

Receive data 1
Deliver data
Send ACK 1

Receive data 0 or corrupt
Send ACK 0

Sender



Idle 0

Wait for ACK 1

Wait for ACK 0

Idle 1

Receive ACK 0/corrupt
Re-send data 1

Receive ACK 1
^

Data to send
Send data 0

Data to send
Send data 1

Receive ACK 0
^

Receive ACK 1/corrupt
Re-send data 0

FEATURES TO HANDLE CORRUPTION

- Checksums – to detect errors
- ACK packets – to detect last correctly received data
- Retransmissions – to overcome errors
- Sequence numbers – to detect duplicates

BUILDING A RELIABLE PROTOCOL (V.2)

- Let's create a protocol for reliable delivery
  - Send only one segment at a time
  - Identify when sending is allowable action
  - Identify when re-sending is required
  - Enumerate events and actions for both sender and receiver
  - Draw state machine
- Next scenario assumptions
  - One sender, one receiver
  - Data to send comes from upper layer
  - Segments can be lost or corrupted

WHAT IF A SEGMENT IS LOST?

- What happens if a data segment is lost?
- What happens if an ACK is lost?
- How to determine if a data segment or ACK is lost?
  - Who determines it? Sender or receiver?

WHAT IF A SEGMENT IS LOST?

- What happens if a data segment is lost?
- What happens if an ACK is lost?
- How to determine if a data segment or ACK is lost?
  - Who determines it? Sender or receiver?

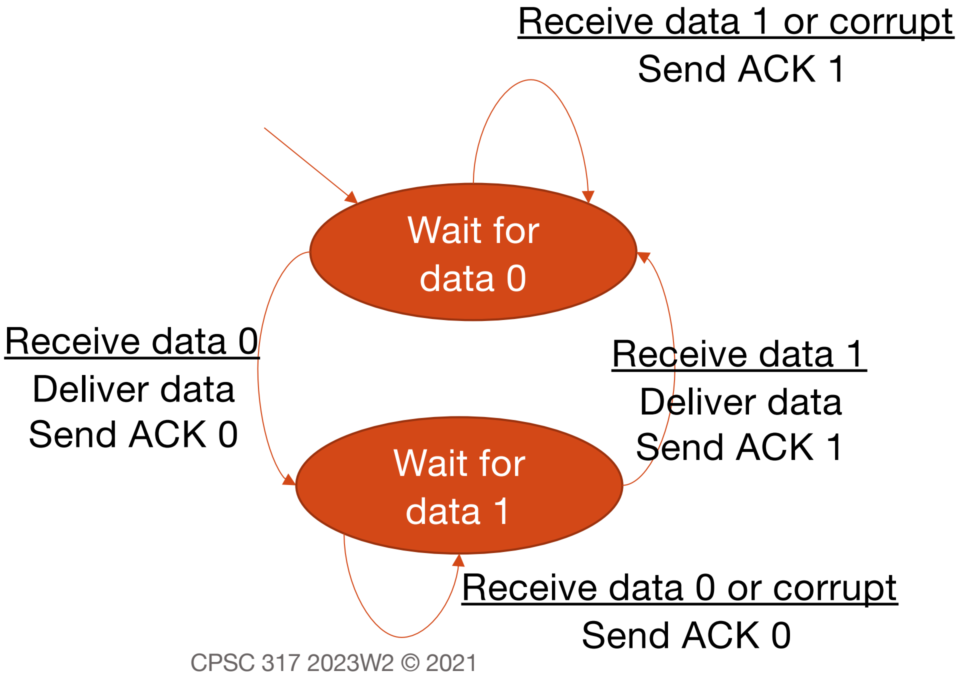
- What changes are needed on the receiver?
- What changes are needed on the sender?

CORRUPT OR LOST?

- Does it make sense to distinguish corrupt data from lost data?
  - If something is corrupt, we don't know what's corrupt
  - Headers may have been corrupted too
  - The corrupt segment may not even be for you!!!
  - So, corrupt data becomes lost data (usually at link layer)

ALTERNATING BIT PROTOCOL (NO NAK)

Receiver



Wait for data 0

Wait for data 1

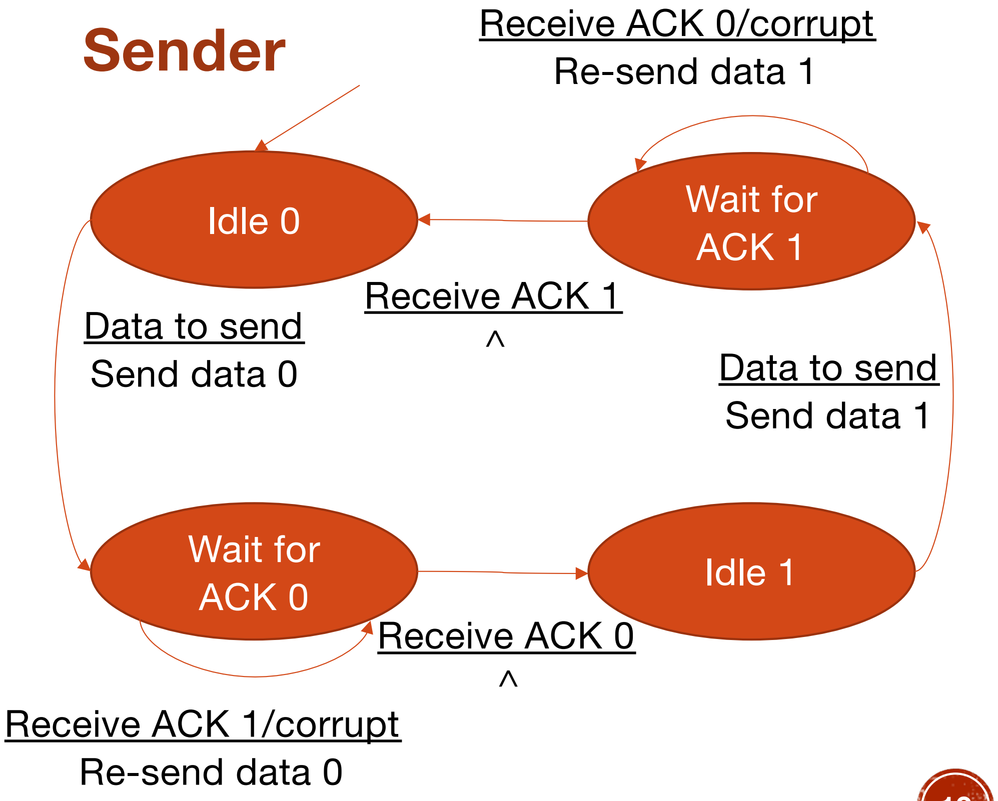
Receive data 1 or corrupt
Send ACK 1

Receive data 0
Deliver data
Send ACK 0

Receive data 1
Deliver data
Send ACK 1

Receive data 0 or corrupt
Send ACK 0

Sender



Idle 0

Wait for ACK 1

Wait for ACK 0

Idle 1

Receive ACK 0/corrupt
Re-send data 1

Receive ACK 1
^

Data to send
Send data 0

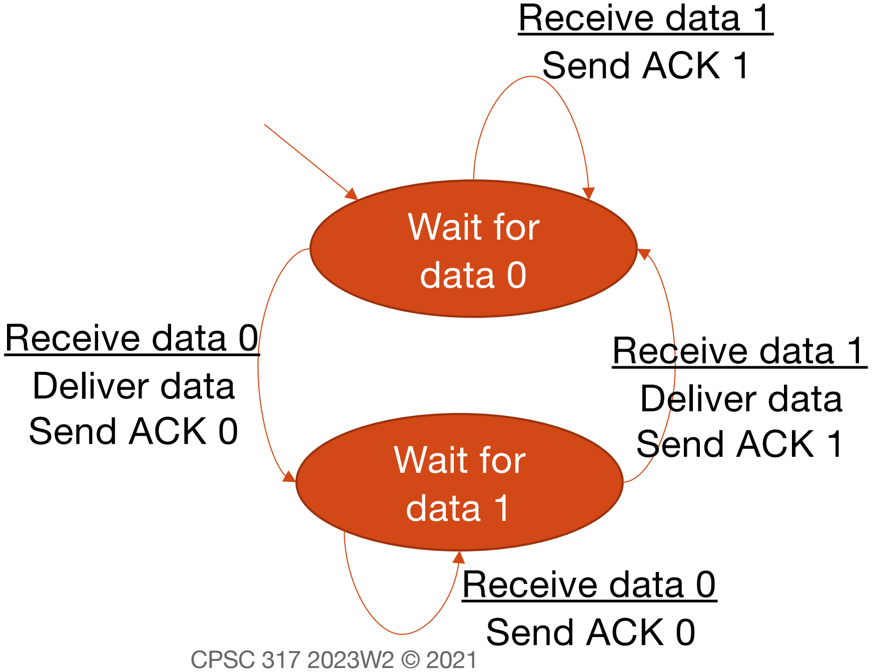
Data to send
Send data 1

Receive ACK 0
^

Receive ACK 1/corrupt
Re-send data 0

ELIMINATE CORRUPT SEGMENTS

Receiver



Wait for data 0

Wait for data 1

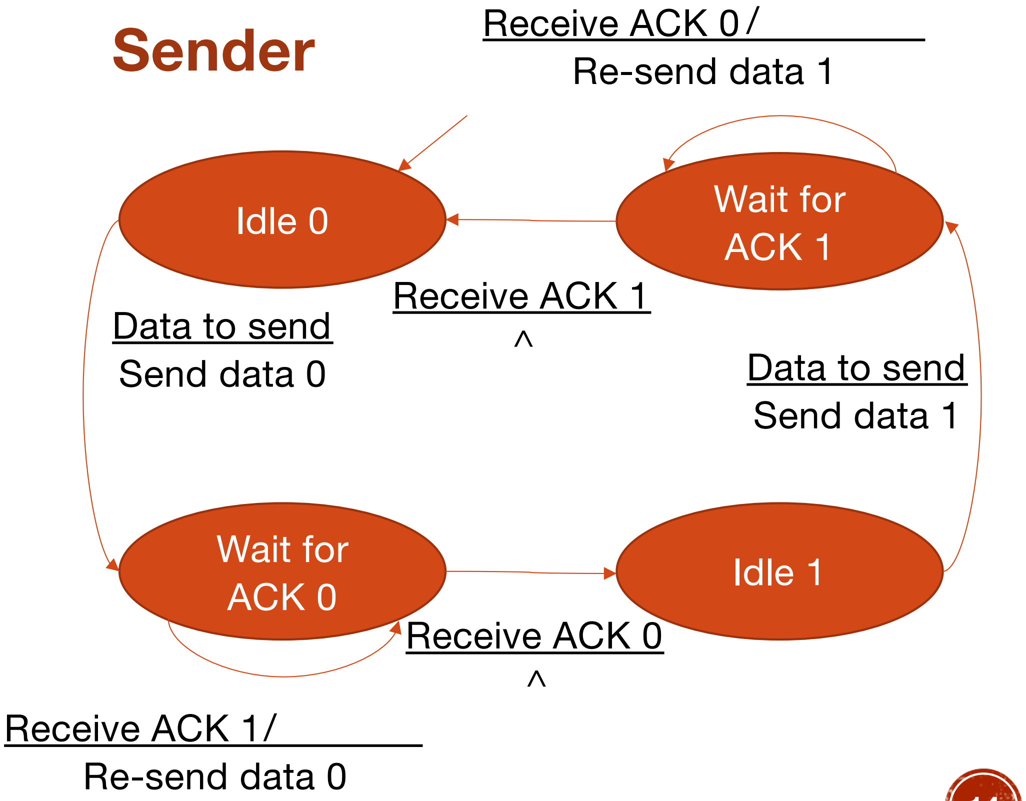
Receive data 1
Send ACK 1

Receive data 0
Deliver data
Send ACK 0

Receive data 1
Deliver data
Send ACK 1

Receive data 0
Send ACK 0

Sender



Idle 0

Wait for ACK 1

Wait for ACK 0

Idle 1

Receive ACK 0 / ______
Re-send data 1

Receive ACK 1
^

Data to send
Send data 0

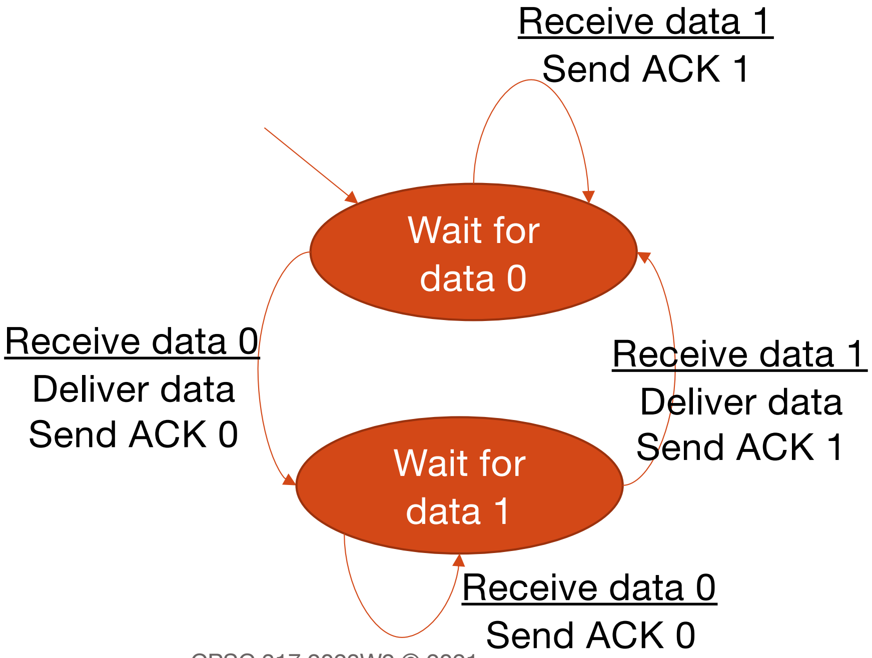
Data to send
Send data 1

Receive ACK 0
^

Receive ACK 1 / ______
Re-send data 0

HANDLE LOST SEGMENTS

Receiver



Wait for data 0

Wait for data 1

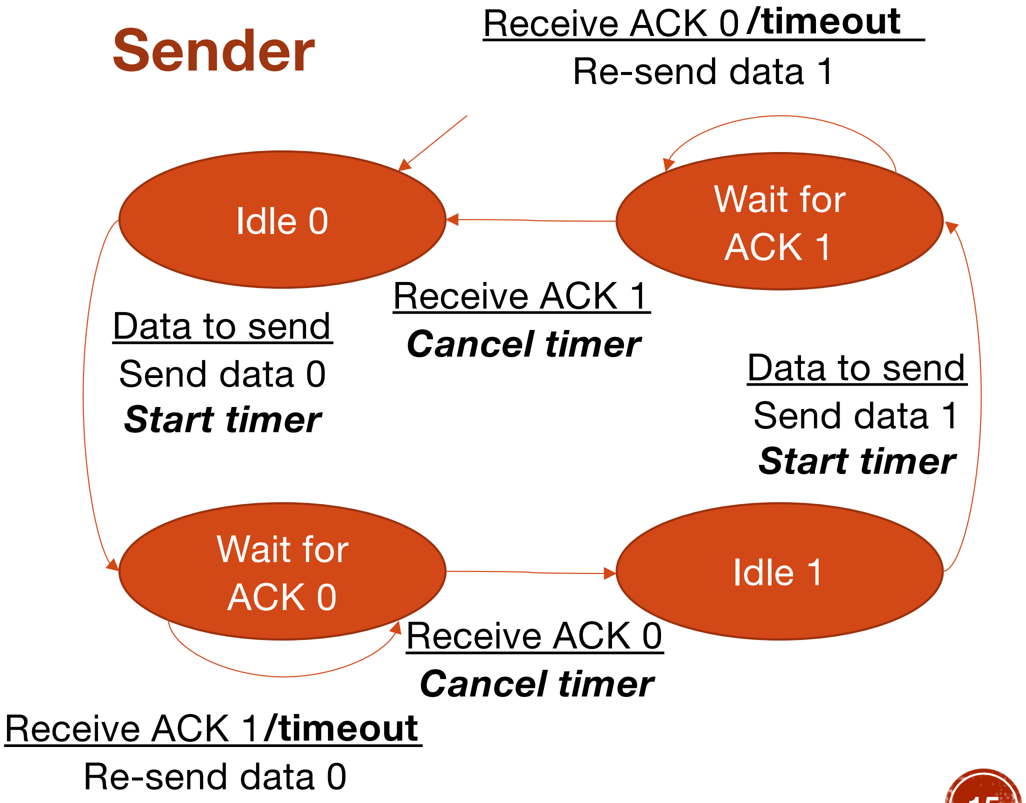
Receive data 1
Send ACK 1

Receive data 0
Deliver data
Send ACK 0

Receive data 1
Deliver data
Send ACK 1

Receive data 0
Send ACK 0

Sender



Idle 0

Wait for ACK 1

Wait for ACK 0

Idle 1

Receive ACK 0 **/timeout**
Re-send data 1

Receive ACK 1
Cancel timer

Data to send
Send data 0
Start timer

Data to send
Send data 1
Start timer

Receive ACK 0
Cancel timer

Receive ACK 1 **/timeout**
Re-send data 0

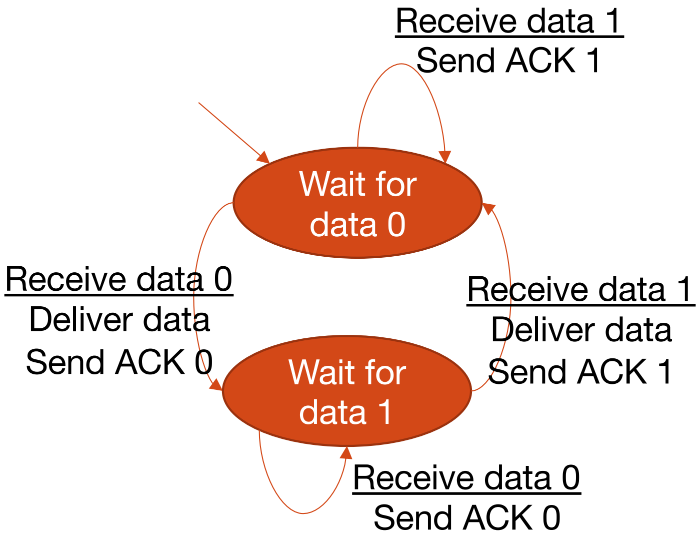
CLICKER QUESTION

Is this state machine complete?

A. Yes

B. No

Receiver



Receive data 1
Send ACK 1

Wait for data 0

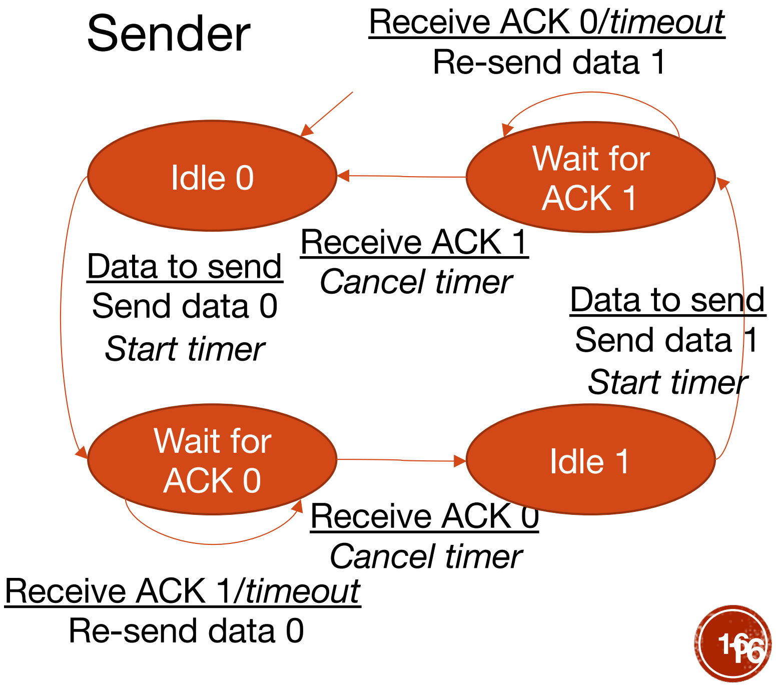
Receive data 0
Deliver data
Send ACK 0

Receive data 1
Deliver data
Send ACK 1

Wait for data 1

Receive data 0
Send ACK 0

Sender



Receive ACK 0/*timeout*
Re-send data 1

Idle 0

Wait for ACK 1

Receive ACK 1
Cancel timer

Data to send
Send data 0
Start timer

Data to send
Send data 1
Start timer

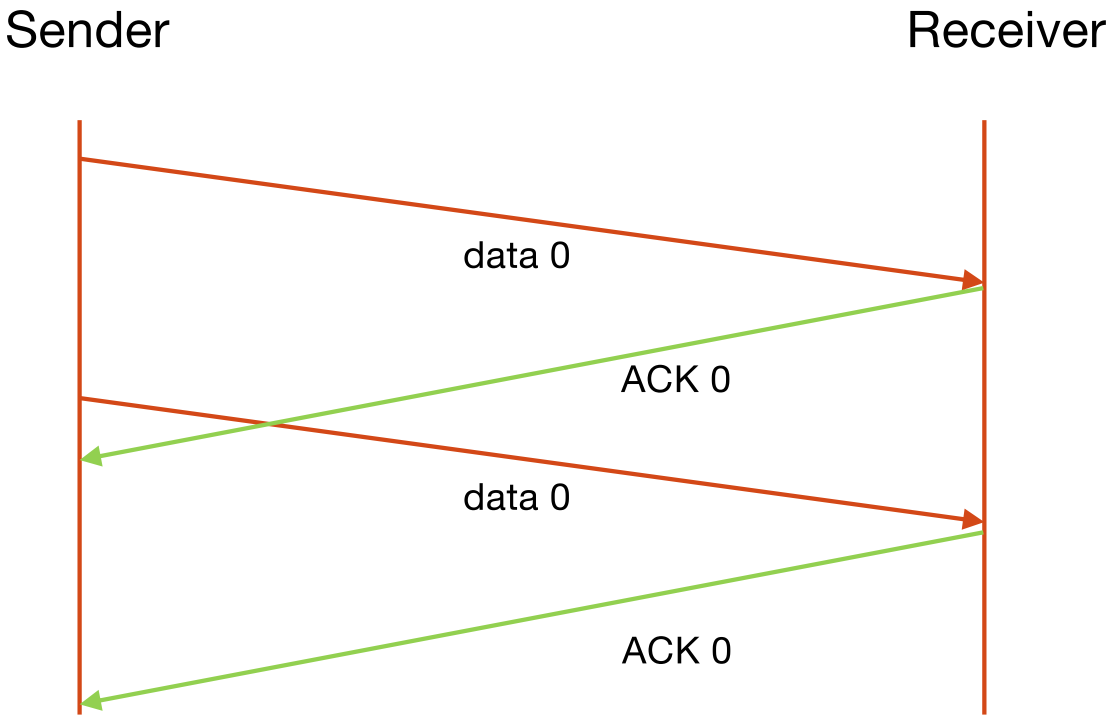
Wait for ACK 0

Idle 1

Receive ACK 0
Cancel timer

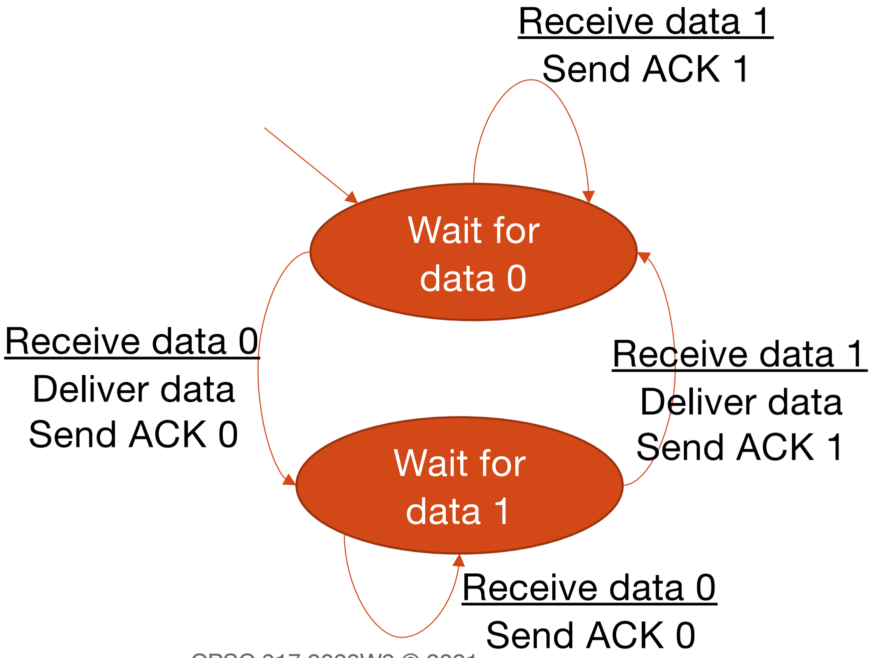
Receive ACK 1/*timeout*
Re-send data 0

TIME OUT TOO SOON



ALTERNATING BIT PROTOCOL (TIMEOUT)

Receiver



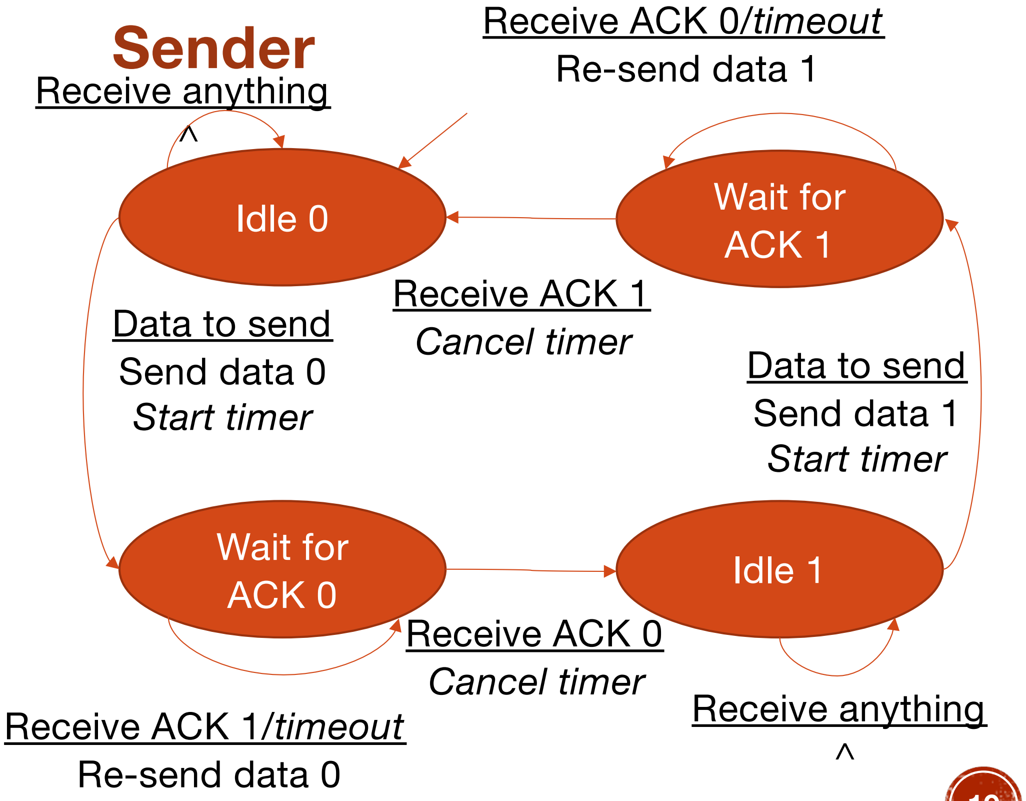
Receive data 1
Send ACK 1

Receive data 0
Deliver data
Send ACK 0

Receive data 1
Deliver data
Send ACK 1

Receive data 0
Send ACK 0

Sender



Receive anything
^

Receive ACK 0/*timeout*
Re-send data 1

Receive ACK 1
Cancel timer

Data to send
Send data 0
Start timer

Data to send
Send data 1
Start timer

Receive ACK 0
Cancel timer

Receive ACK 1/*timeout*
Re-send data 0

Receive anything
^

HOW LONG SHOULD THE TIMEOUT BE?

- Should it be the same across connections?
- Should it always be the same for the same connection?
- What happens if the timeout is too long?
- What happens if the timeout is too short?
- What measurable metric can be used to infer a proper timeout?

HOW LONG SHOULD THE TIMEOUT BE?

- Assume sequential segments experience the following sequence of RTTs. What should the timeout be?
  - 10, 10, 10, 10, 10, 10, 10, 10
  - 100, 10, 100, 100, 100, 100, 100
  - 1000, 10, 10, 10, 10, 10, 10, 10
  - 10, 100, 10, 100, 10, 100, 10, 100
  - 1, 1, 1, 5, 5, 5, 10, 10, 40, 40, 40, 60, 100, 100, 2, 2, 2, 4, 2

ESTIMATING TIMEOUT

- Simple average of RTTs doesn't respond quickly enough
  - Also doesn't capture jitter
- Timeout value must adapt:
  - Track changes in RTT over time
  - Accommodate segment-to-segment deviations due to jitter
- Discussion in more detail: textbook 3.5.3

TIMEOUT FORMULA

- Assuming measured RTT of t
- Estimated RTT:

$$ERTT_i = (1-\alpha)\times ERTT_{i-1} + \alpha\times t$$

- Deviation of RTT (captures jitter):

$$\Delta RTT_i = (1-\beta)\times\Delta RTT_{i-1} + \beta\times|t - ERTT_{i-1}|$$

- Timeout:

$$ERTT_i + 4\times\Delta RTT_i$$

- Suggested values: $\alpha = 0.125, \beta = 0.25$

FEATURES TO SUPPORT RELIABLE TRANSPORT

- Checksums – to detect errors
- ACK packets – to detect last correctly received data
- Retransmissions – to overcome errors, data loss
- Sequence numbers – to detect duplicates
- Timers – to handle data loss

PERFORMANCE

- Alternating bit protocol aka stop-and-wait protocol
- Sender does not send next segment until previous segment acknowledged

CLICKER QUESTION

- Assume a connection from Vancouver to Montreal
  - Round Trip Time is 30 ms
  - Link speed (bandwidth) is 1Gbps
  - Segment size is 1000 bytes, including overhead
- How much of the link are we actually using?
- I want a number between 0 and 1, e.g., 0.1

0.027% UTILIZATION??

- What can we do about this?

IN-CLASS ACTIVITY

- ICA43