CPSC 317 INTERNET COMPUTING

Module 6: Reliable Transmission and TCP

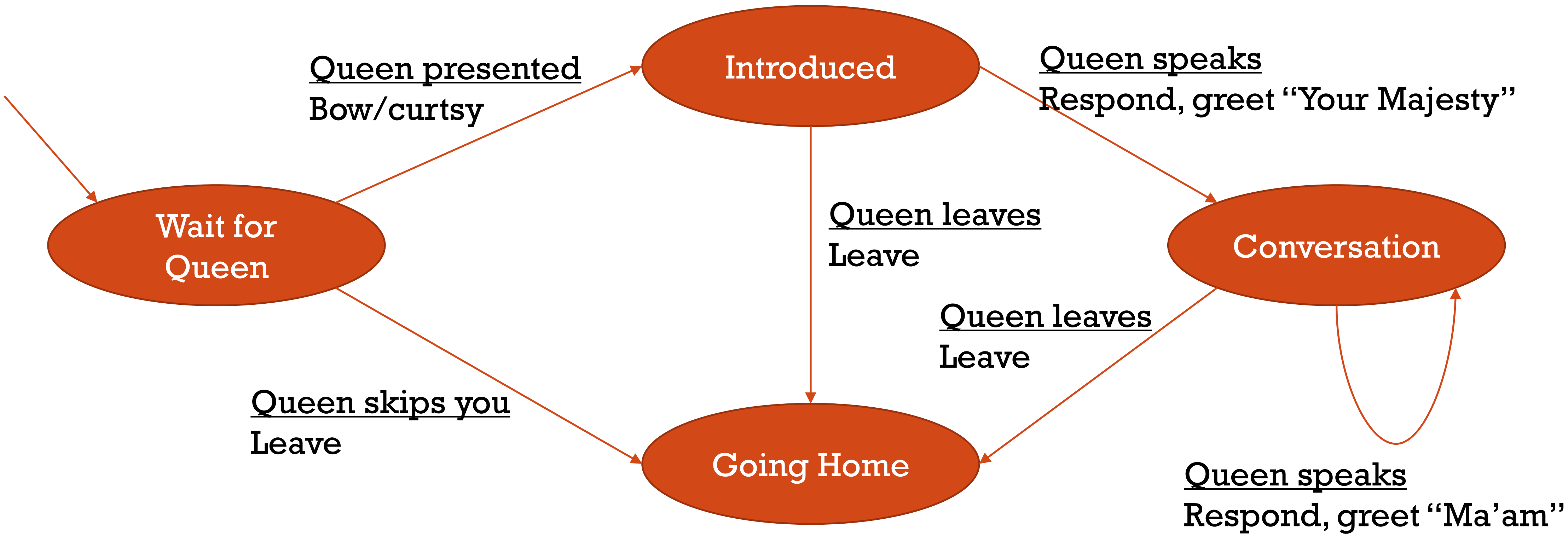
REMINDER: PROTOCOLS

- From textbook: A **protocol** defines the format and the order of messages exchanged between two or more communicating entities, as well as the actions taken on the transmission and/or receipt of a message or other event
- Key concepts:
  - Format and order of messages
  - Actions to be taken on events
- A fully-defined protocol must provide a proper action for any event in any state

PROTOCOL EXAMPLE: GREETING THE QUEEN

- Don't speak until spoken to
- Upon presentation to the Queen
  - Men do a neck bow
  - Women a slight curtsy
- Queen speaks to you
  - First salutation is "Your Majesty"
  - Subsequently it is "Ma'am"

PROTOCOLS AS STATE MACHINES



BUILDING A RELIABLE PROTOCOL

- Let's create a protocol for reliable delivery
  - Send only one packet at a time
  - Identify when sending is allowable action
  - Identify when resending is required
  - Enumerate events and actions for both sender and receiver
  - Draw state machine
- Initial scenario assumptions
  - One sender, one receiver
  - Data to send comes from upper layer
  - Packets are never lost, but may be corrupted

POSSIBLE EVENTS AND ACTIONS

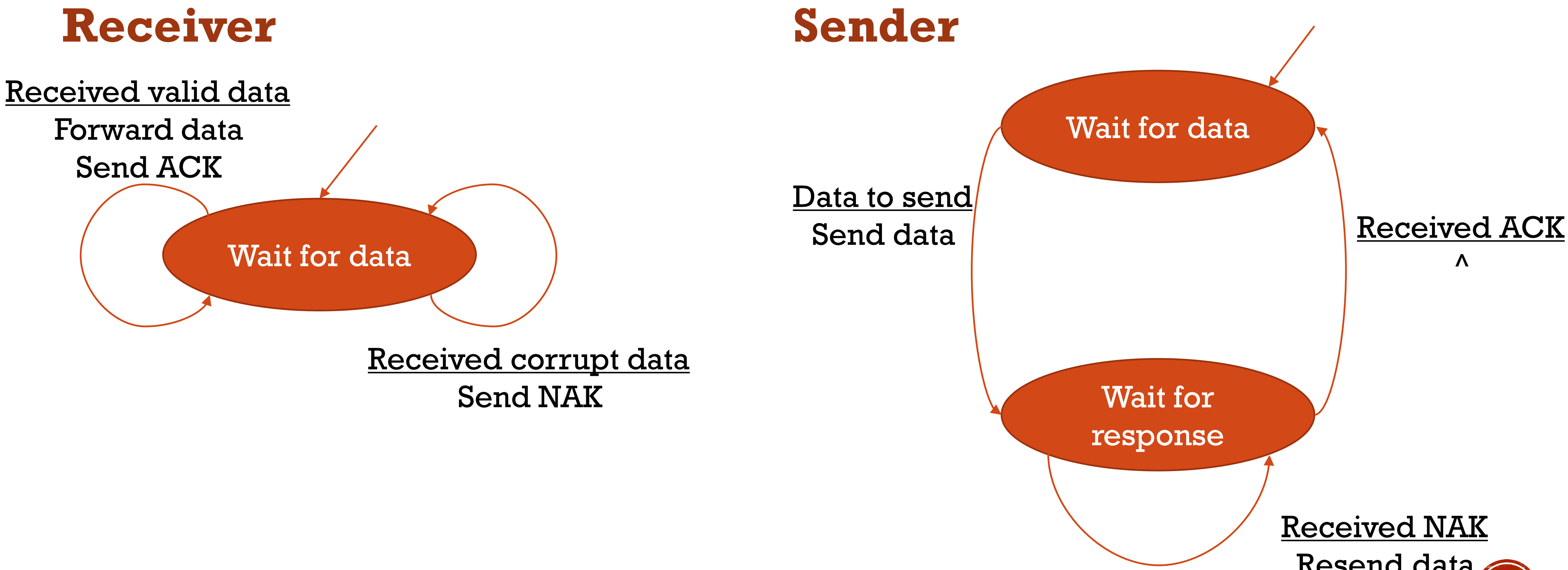
Receiver

- Packet received without problems
  - Send ACK
- Packet received corrupted
  - Send NAK

Sender

- Data ready to send
  - Send data
- Receive ACK
  - Get ready to send more data
- Receive NAK
  - Resent data

STATE MACHINE: CORRUPTION SCENARIO



CORRUPT RESPONSE

- What if the ACK/NAK is corrupted?
- What if:
  - We ignore the corrupt ACK/NAK?
  - We treat it as an ACK?
  - We treat it as a NAK?

DEALING WITH DUPLICATE TRANSFER

- How can the receiver deal with a duplicate packet?
- What information needs to be included to allow receiver to identify packet as duplicate?
- Reminder: scenario does not lose data, only corrupts

POSSIBLE EVENTS

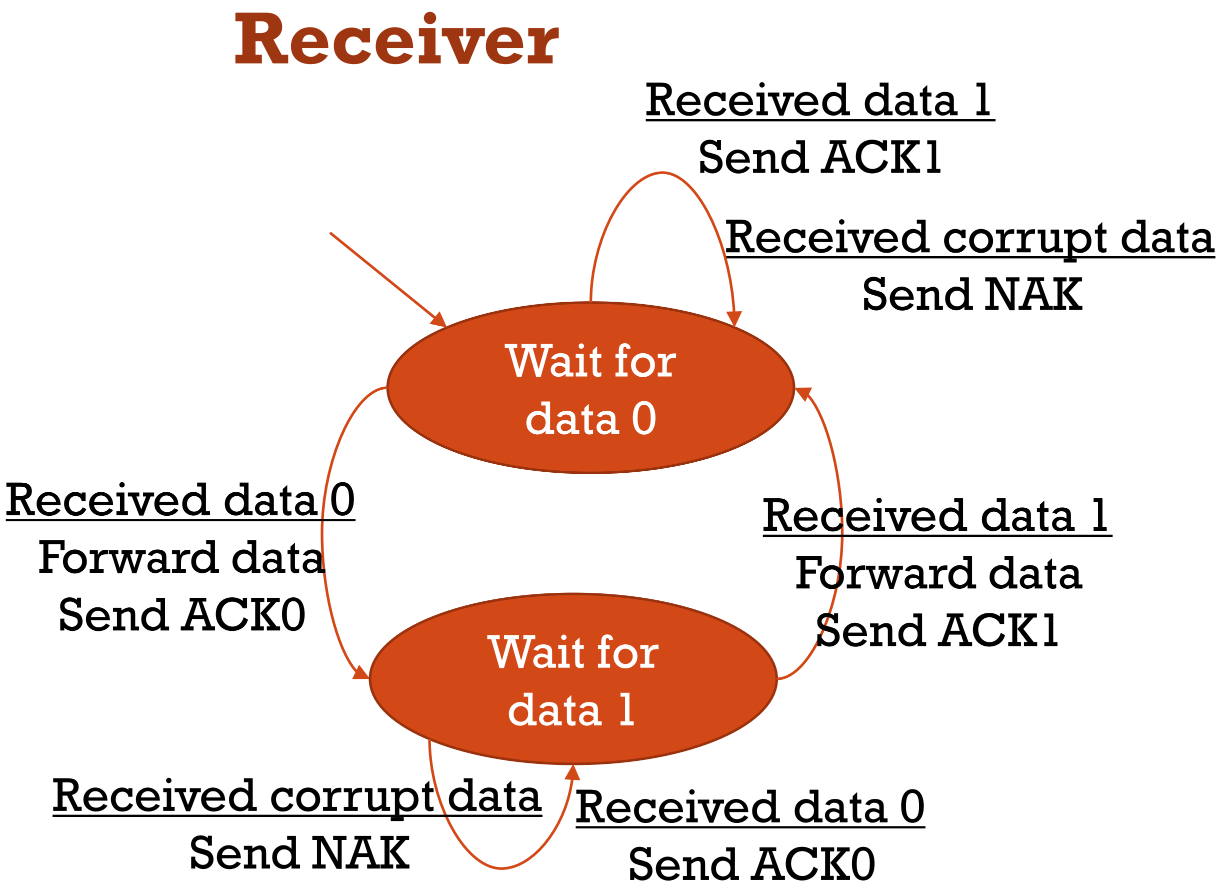
Receiver

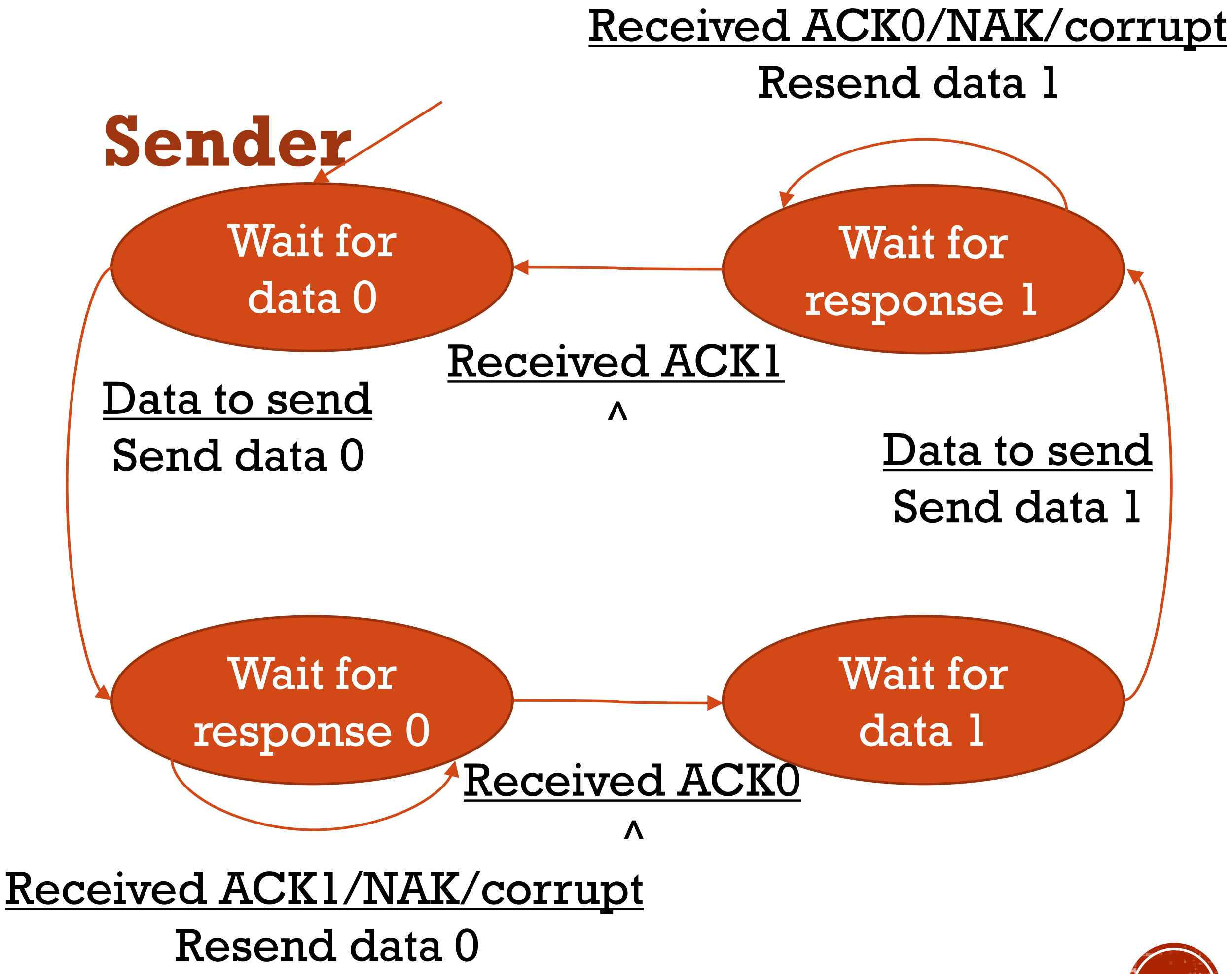
- Packet 0 received without problems
- Packet 0 received corrupted
- Packet 1 received without problems
- Packet 1 received corrupted

Sender

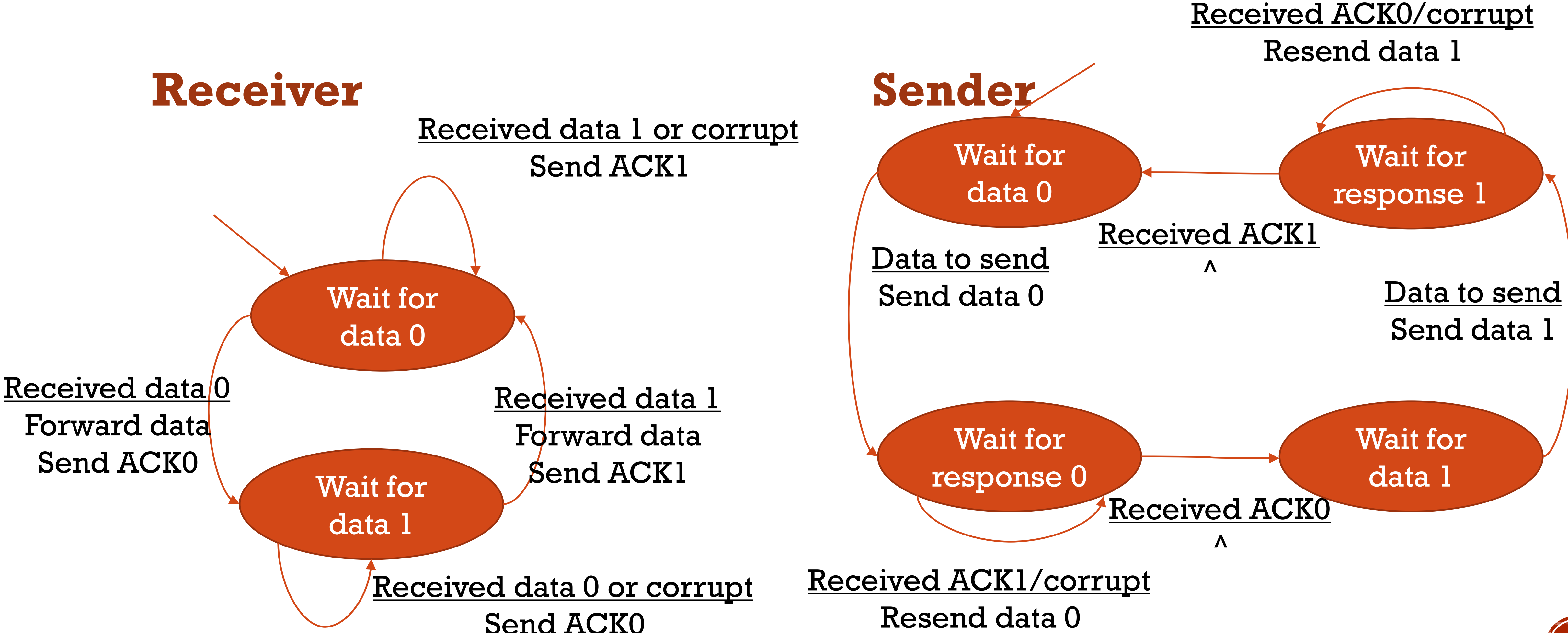
- Data ready to send
- Receive ACK0/ACK1
- Receive NAK
- Receive corrupt response

ALTERNATE BIT PROTOCOL





ALTERNATE BIT PROTOCOL (WITHOUT NAK)



WHAT IF A PACKET IS LOST?

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

WHAT IF A PACKET IS LOST?

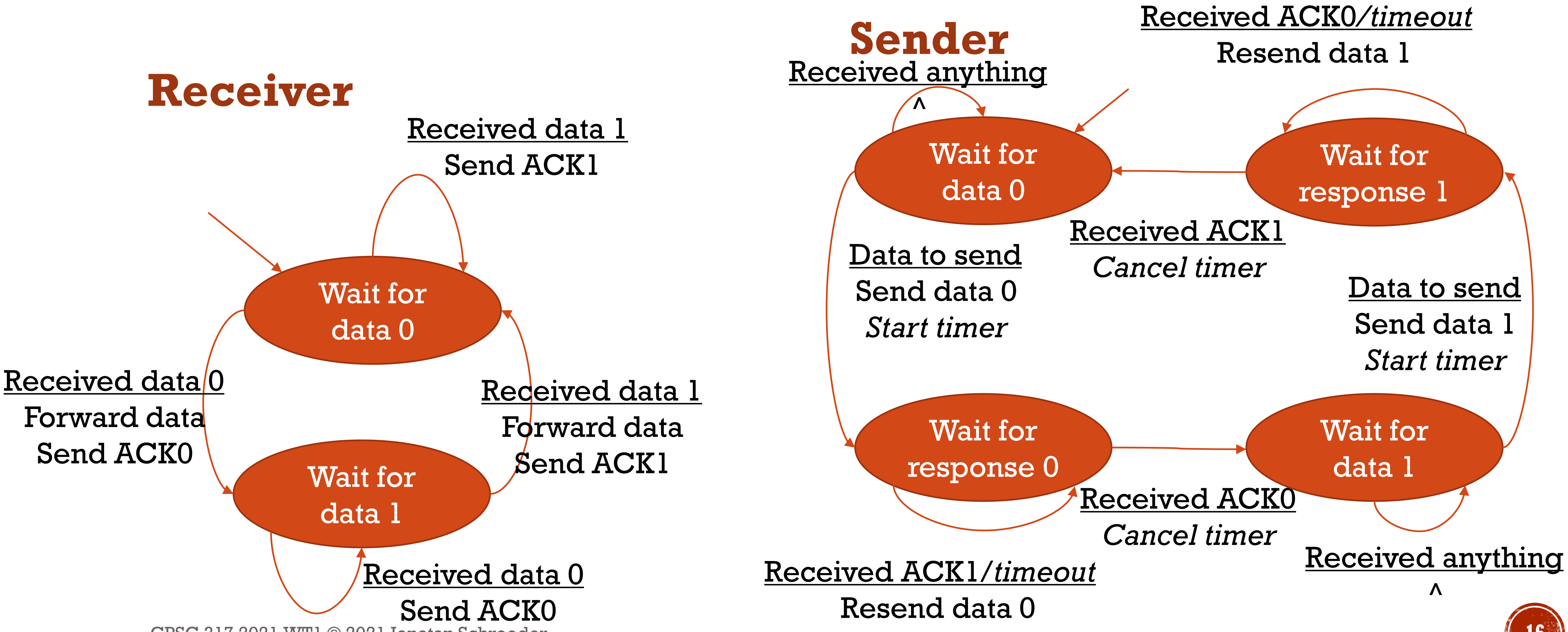
- What happens if a packet is lost?
- What happens if an ACK is lost?

- 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 packet may not even be for you!!!
  - So, corrupt data becomes lost data (usually at link layer)

ALTERNATE BIT PROTOCOL (TIMEOUT)



HOW LONG SHOULD TIMEOUT BE?

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

HOW LONG SHOULD TIMEOUT BE?

- Assume sequential packets take the following sequence of RTT to be ACKed. What should be the timeout?
  - 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 packet-to-packet 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: $\alpha = 0.125, \beta = 0.25$

ALTERNATE BIT PROTOCOL IN PRACTICE

- Assume a connection from Vancouver to Montreal
  - RTT is 30 ms
  - Link speed is 1Gbps
  - Packet size is 1000 bytes, including overhead
- How much of the bandwidth are we actually using?
  - Only one packet is sent at a time
  - Transmission delay, one packet: ${}^{8000}/_{10^9} = 0.008\ ms$
  - Utilization: ${}^{0.008}/_{30+0.008} = 0.00027$ (or 0.027%)
- How can we solve this?

SENDING MULTIPLE PACKETS

- Sender can send multiple packets
- Don’t wait for each acknowledgement
  - Assume most packets are successful
- Sender needs to save sent packets to potentially resend
  - Size is limited

SENDER'S WINDOW

- Sender's window: range of packets that are stored for potential resend
- Window only moves when first packet in window is acknowledged
  - What is other ACKs are received?
- New packets are sent only when they "fit" in the window

RECEIVER'S WINDOW

- What if receiver receives packets out of order?
- Receiver's window: store packets received out of order
- Once missing packets arrive, window is processed
- Packets received beyond the window limit will be discarded

PROBLEMS TO CONSIDER

- Sender
  - How does the sender know that data got lost?
  - Can lost data be distinguished from a lost ACK?
  - If we send more than one packet, how many packets can we remember?
- Receiver
  - How can you tell if data is out-of-order or missing?
  - What should be ACKed?

GO-BACK-N STRATEGY

- Receiver:
  - When packet is received, send ACK for last packet received in order
  - Discard arriving packet if out of order (no receiver window)
- Sender:
  - Can have a specific number of outstanding (unacknowledged) packets in memory: ***sender's window***
  - Start timer on first packet sent
  - On timeout go to last unack'ed packet and resend everything (restart timer)
  - Received ACKs may be cumulative (restart timer on receipt)

SELECTIVE-REPEAT STRATEGY

- Receiver:
  - Each packet is ack'ed individually
  - Out of order packet is stored for later: ***receiver's window***
- Sender:
  - Can have a specific number of outstanding (unacknowledged) packets in memory: ***sender's window***
  - Each packet has its own timer
  - Each packet is individually resent if timeout is reached
  - ACKs received in order move the sender's window

SEQUENCE NUMBER RANGE

- The range of possible sequence numbers is limited by the number of bits used for it
  - Example: 3 bits gets numbers 0-7, 8 bits gets numbers 0-255
  - Sequences return to zero (e.g., in range 0-7, after 7 comes another 0)
- What is the maximum sender window size for the range 0-255?
  - Maybe easier to compute: what about range 0-3?
  - Can the receiver distinguish a new 0 from a resent old 0?
  - Does the answer change for selective repeat vs go-back-N?

SEQUENCE NUMBER RANGE

- Rule: sender's window size + receiver's window size <= sequence number range
- For a range with n numbers (0 to $n - 1$):
  - Go-back-N:
    - receiver's window size is 1
    - sender's maximum window size is $n - 1$
  - Selective Repeat:
    - any values for window sizes that add up to n is fine
    - for same size on both sides, use $\lfloor {}^{n}/_{2} \rfloor$
- Why?

FLOW CONTROL

- Should we always send the full window size?
- What if the receiving application is slow accepting new data?
  - Packets will accumulate in the receiver's buffer
  - Eventually buffer will be full, packets will be dropped
  - Immediately resending this data does not resolve the problem
- Receiver will notify the sender how much data it can handle
  - This information is usually included in the ACK
  - Sender adjusts its window size based on this information

CONGESTION CONTROL

- What if the network can't handle a full window's worth of data?
  - Packets and ACKs will be dropped by the routers
- Missing ACKs are a sign that there is congestion somewhere (in either direction)
- Sender can reduce sending window once congestion is detected
  - Example: if some number of ACKs are missing in a period of time
- If all packets are ACKed, we can increase the window again

EFFECT ON THROUGHPUT

- Throughput is affected by
  - Bandwidth of sender's direct network connection
  - Receiver's specified window size (flow control)
  - Sender's adjusted window size (congestion control)
- If sender's direct connection is not the bottleneck:
  - Another router will experience congestion elsewhere
  - Congestion control will reduce transmission speed

TCP IMPLEMENTATION

- Both sides act as sender and receiver
  - All packets have both a sequence number and an ACK
  - Sequence numbers in one direction correspond to ACK numbers in opposite direction
  - Sequence numbers are incremented by payload size
- Retransmission strategy:
  - ACKs correspond to first sequence number not yet received (similar to Go-Back-N)
  - Receiver stores packets in its own window (like Selective-Repeat)
  - Three or more ACKs with same number trigger a retransmission without a timeout (fast retransmission)

TCP SEGMENT FORMAT

- For information only, you will not be tested on the format of the header

<table>
<tr><td colspan="10">Source port #</td><td>Destination port #</td></tr>
<tr><td colspan="11">Sequence Number</td></tr>
<tr><td colspan="11">Acknowledgement Number</td></tr>
<tr><td>Hdr Len</td><td>Not used</td><td>CWR</td><td>ECE</td><td>URG</td><td>ACK</td><td>PSH</td><td>RST</td><td>SYN</td><td>FIN</td><td>Receiver Window</td></tr>
<tr><td colspan="10">Checksum</td><td>Urgent Data Pointer</td></tr>
<tr><td colspan="11">Options (optional, variable length)</td></tr>
<tr><td colspan="11">Application payload (optional, variable length)</td></tr>
</table>

TCP CONNECTION ESTABLISHMENT

- TCP uses "three-way handshake" to establish connection
- Client sends initial SYN message
  - Initial sequence number for client→server is specified
- Server responds with SYN/ACK message
  - Client→server sequence number is confirmed in ACK
  - Server→client initial sequence number is specified
- Client sends an ACK message
  - Server→client sequence number is confirmed in ACK

TCP CONNECTION TERMINATION

- Side that wants to terminate sends FIN message
  - Other side responds with ACK
- Other side will also send a FIN message
  - It may not send it immediately, since it may have more data to send
  - Also responded with an ACK
- Alternative: connection abortion (RST message)
  - Usually used if other side misbehaves or if disconnection or too many timeouts are detected