CS 313: Networks

Advanced TCP
Exam 2 Review
3.5

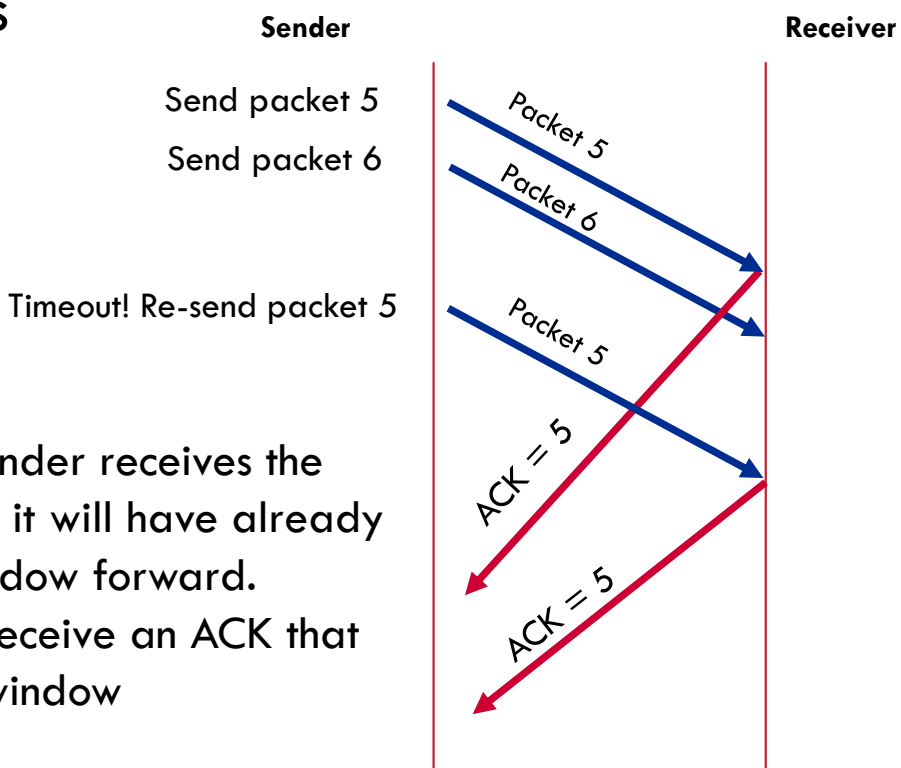
Scott Griffith | October 24 2019| Whitworth University

Homework

- Pg. 294. P2 Answer true or false
  - With SR, it is possible for the sender to receiver an ACK for a packet that falls outside of its current window.
  - With GBN?
  - The alternating-bit protocol is the same as SR protocol with a sender and receiver window size of 1
  - Alternating-bit is the same as GBN with window size of 1

Homework

- Pg. 294. P2 Answer true or false
  - With SR, it is possible for the sender to receiver an ACK for a packet that falls outside of its current window.
  - Yes



At the point sender receives the second ACK 5, it will have already moved the window forward.
Thus it would receive an ACK that is before the window

The only way to go beyond the current window is to possibly wrap around (if sequence numbers are too small)

Homework

- Pg. 294. P2 Answer true or false
  - With GBN, it is possible for the sender to receiver an ACK for a packet that falls outside of its current window.
    - Basically the same as previous, timeout likely causes double ACKs

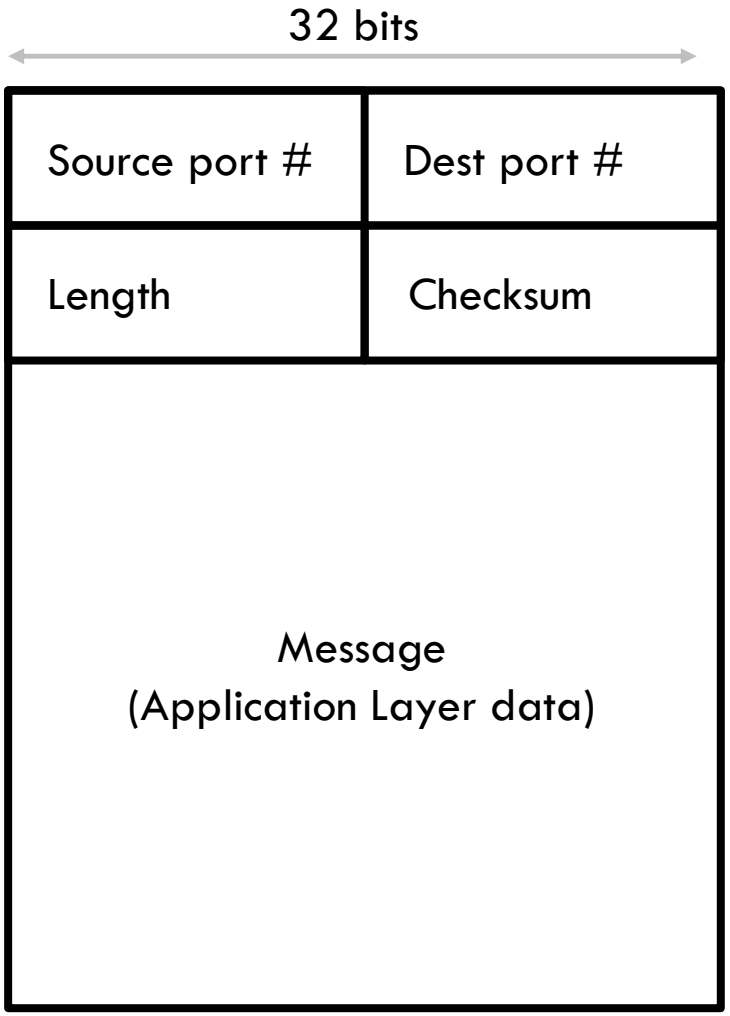
Homework

- Pg. 294. P2 Answer true or false
  - The alternating-bit protocol is the same as SR protocol with a sender and receiver window size of 1
  - Alternating-bit is the same as GBN with window size of 1
  - Both True, it is the same as having a window with a size of 1.

UDP segment format?

UDP segment format

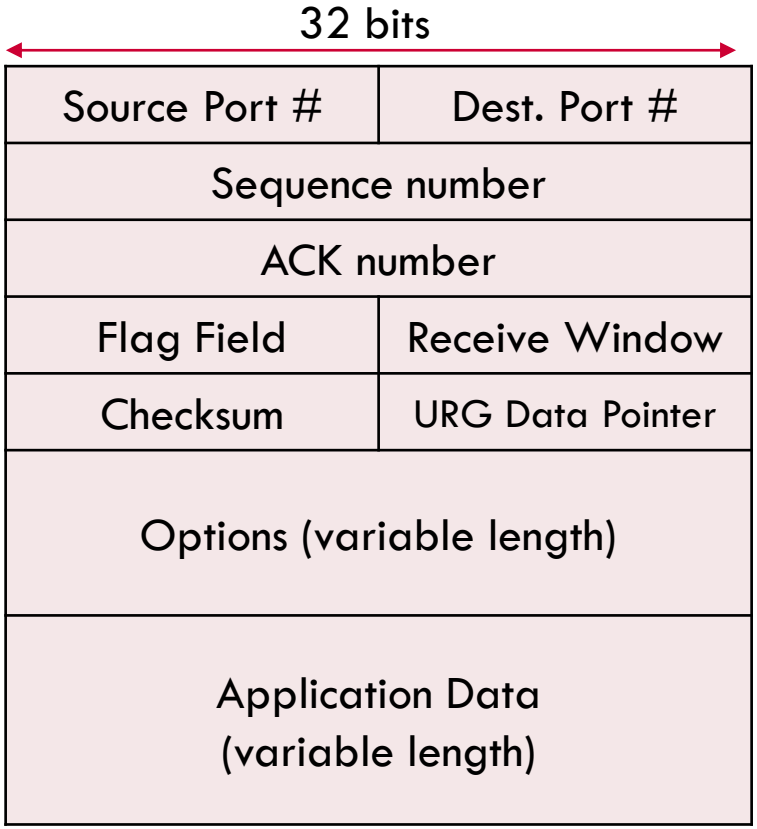
- Port Numbers
  - Used to de/multiplex between Application and Transport Layer
  - Between 0 and 65535
- Length
  - Total length in bytes of the whole segment
    - 4*(2 bytes) + message size
- Checksum
  - Error Check
- Message
  - What are we actually sending?



TCP segment format?

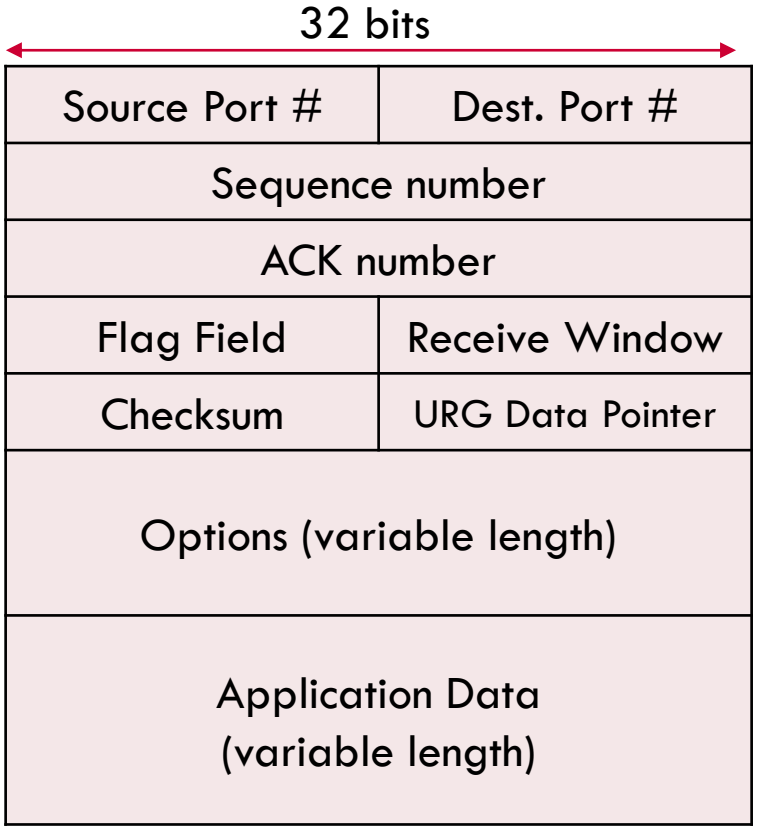
TCP Segment

- Source / Dest. Port #
  - Same as UDP
- Sequence / ACK Number
  - Used for RDT (more in a sec)
- Flag Field
  - TCP specific control flags
- Receive Window
  - Flow control, how many bytes the receiver will accept



TCP Segment

- Checksum
  - Same as UDP
- URG Data Pointer
  - Not really used, urgent data flag
- Options
  - RFC 854, RFC 1323
  - More TCP options
- Application Data
  - Data actually being sent by application layer



Concepts

- Pipelining
- Go-Back-N
- Selective Repeat

Concepts

- Pipelining
  - More than one message ‘in flight’
  - Requires more complex protocols
- Go-Back-N
- Selective Repeat

Concepts

- Pipelining
  - More than one message ‘in flight’
  - Requires more complex protocols
- Go-Back-N
  - Only Sender has buffer
  - Receiver sends back most ‘in order’ number in ACK
  - Sender keeps one timer for timeout
- Selective Repeat

Concepts

- Pipelining
  - More than one message ‘in flight’
  - Requires more complex protocols
- Go-Back-N
  - Only Sender has buffer
  - Receiver sends back most ‘in order’ number in ACK
  - Sender keeps one timer for timeout
- Selective Repeat
  - Sender and Receiver both have buffers
  - Receiver sends back ACK for every message
  - Sender keeps timer for each message

Outline

- TCP Details
- Exam 2 Review

TCP: Overview

- Reliable, in-order byte stream
- Pipelined protocol
- Sender and Receiver data is buffered
- Full duplex data
  - Data flow in both directions, not just commands

TCP: Overview

- Reliable, in-order byte stream
- Pipelined protocol
- Sender and Receiver data is buffered
- Full duplex data
  - Data flow in both directions, not just commands
- Connection Oriented
  - Three way handshake to initialize both sides
- Flow controlled – avoid congestion

TCP Sequence Numbers

- At the start of transmission
  - send_base = nextseqnum = random();
    - This defines starting window index

TCP Sequence Numbers

- At the start of transmission
  - send_base = nextseqnum = random();
    - This defines starting window index
  - To calculate next sequence number:
    - nextseqnum += data_bytes_in_segment

Example of TCP Sequence Numbers

C	S	3	1	3		i	s		A	w	e	s	o	m	e	!
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

- TCP sender picks random initial sequence number
  - send_base = nextseqnum = 51

Example of TCP Sequence Numbers

C	S	3	1	3		i	s		A	w	e	s	o	m	e	!
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

- TCP sender picks random initial sequence number
  - send_base = nextseqnum = 51
- Each segment is now assigned a number

nextseqnum: 51

C	S	3	1	3		i	s
51	52	53	54	55	56	57	58

nextseqnum: 59

	A	w	e	s	o	m	e
59	60	61	62	63	64	65	66

nextseqnum: 67

!							
67	68	69	70	71	72	73	74

Three Way Handshake

- Before any application data is sent, receiver and sender establish connection
  - This sets sequence numbers
  - Also sets up buffers

Three Way Handshake

- Before any application data is sent, receiver and sender establish connection
  - This sets sequence numbers
  - Also sets up buffers
- Sender sends SYN message with sequence number

Three Way Handshake

- Before any application data is sent, receiver and sender establish connection
  - This sets sequence numbers
  - Also sets up buffers
- Sender sends SYN message with sequence number
- Receiver sends SYN message with ACK of sender number and receiver's sequence number

Three Way Handshake

- Before any application data is sent, receiver and sender establish connection
  - This sets sequence numbers
  - Also sets up buffers
- Sender sends SYN message with sequence number
- Receiver sends SYN message with ACK of sender number and receiver's sequence number
- Sender sends back ACK with receiver's number to finish set up

TCP ACKs

- Cumulative Acknowledgments
  - ACKs – includes the sequence number of the next byte expected from sender

TCP ACKs

- Cumulative Acknowledgments
  - ACKs – includes the sequence number of the next byte expected from sender
- Timers:
  - Used only ONE countdown timer per connection

TCP ACKs

- Cumulative Acknowledgments
  - ACKs – includes the sequence number of the next byte expected from sender
- Timers:
  - Used only ONE countdown timer per connection
  - Retransmissions are triggered by:
    - Timeouts
    - Duplicate acknowledgements (fast retransmit)

TCP Order Note

- The TCP RFC does not require a specific action when data is received out of order

TCP Order Note

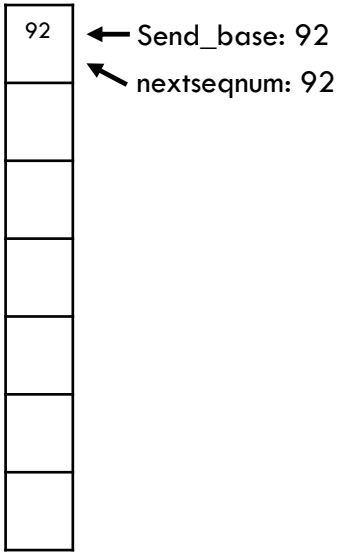
- The TCP RFC does not require a specific action when data is received out of order
  - Option 1: Receiver discards out of order data, only ACKs to data received in order
    - Simple

TCP Order Note

- The TCP RFC does not require a specific action when data is received out of order
  - Option 1: Receiver discards out of order data, only ACKs to data received in order
    - Simple
  - Option 2: Or buffers data, waits for missing data
    - Efficient

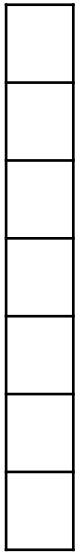
TCP: Fault Free Scenario

Assume send_base and nextseqnum are randomly set to 92



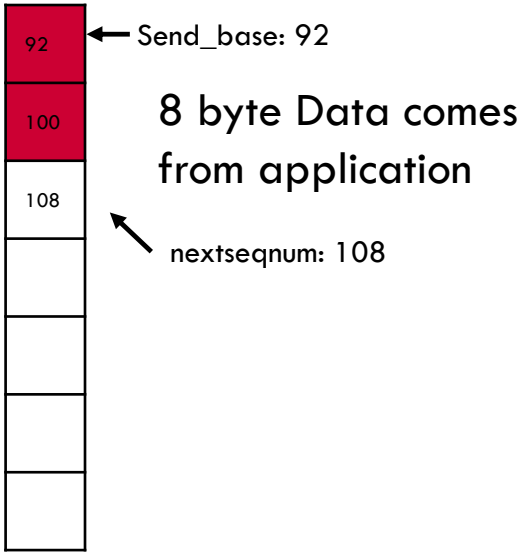
Sender

Receiver



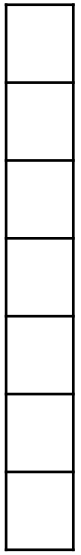
TCP: Fault Free Scenario

Assume send_base and nextseqnum are randomly set to 92



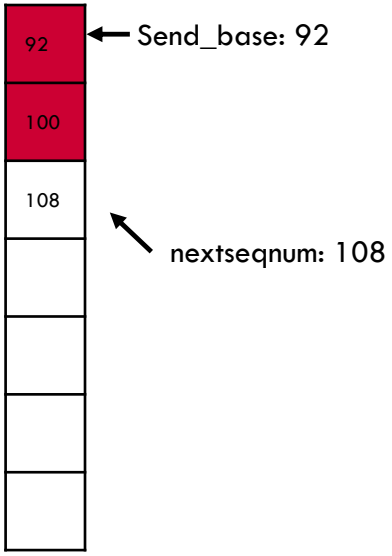
Sender

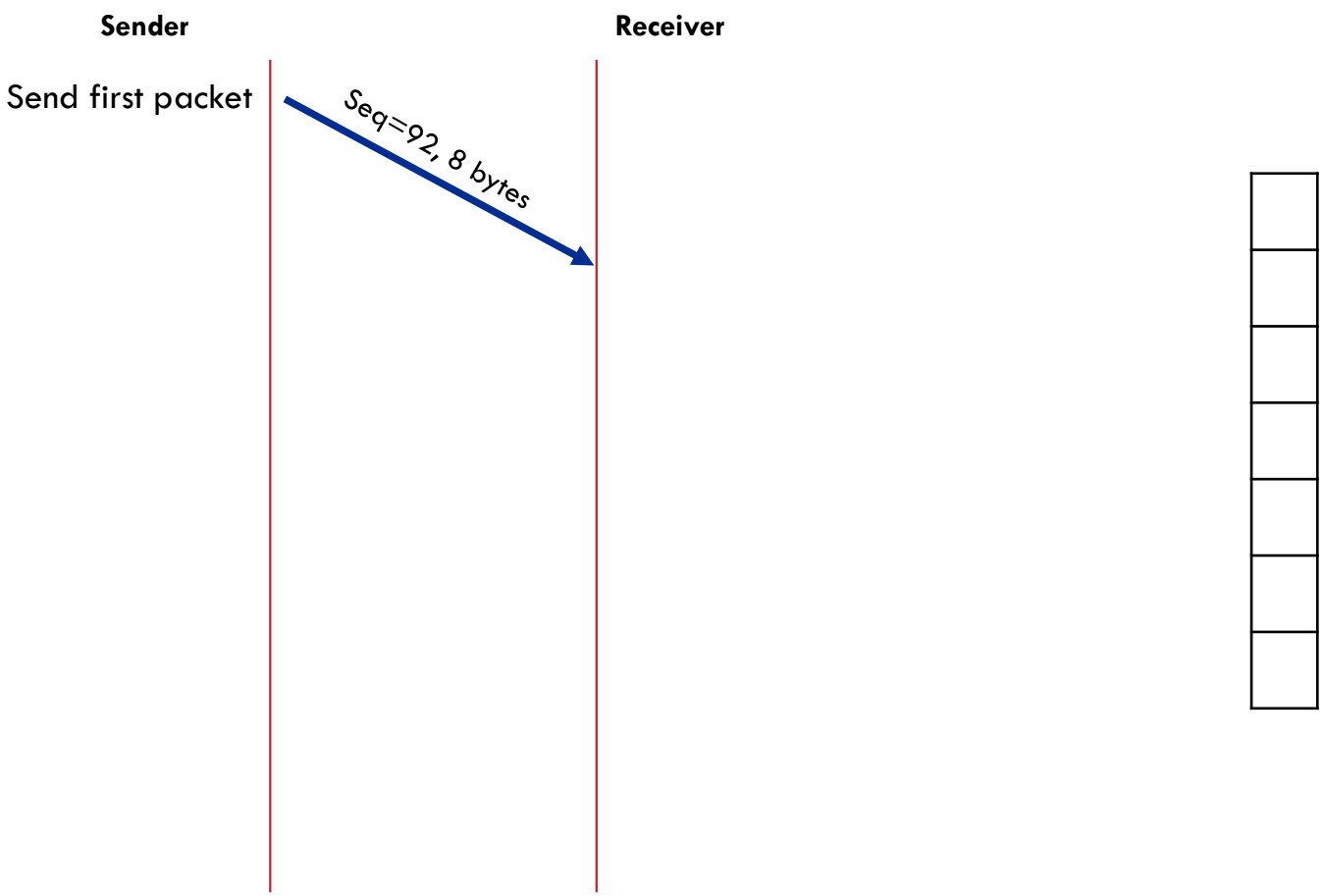
Receiver



TCP: Fault Free Scenario

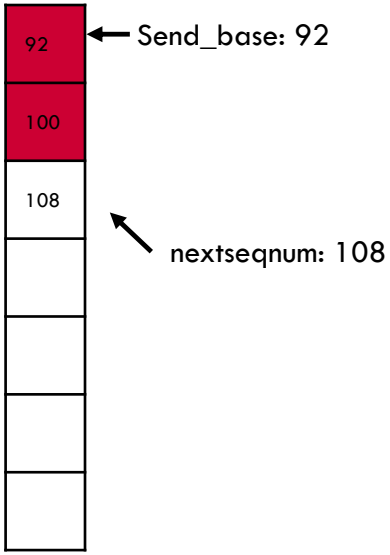
Assume send_base and nextseqnum are randomly set to 92

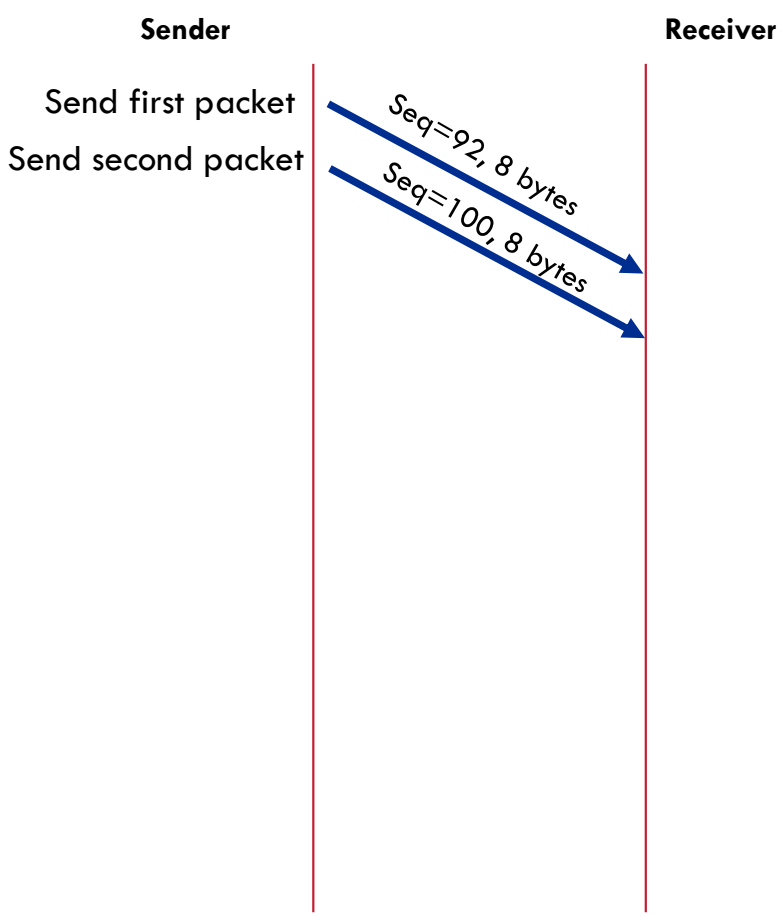


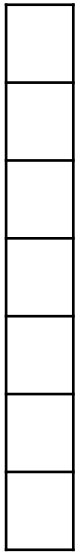


TCP: Fault Free Scenario

Assume send_base and nextseqnum are randomly set to 92

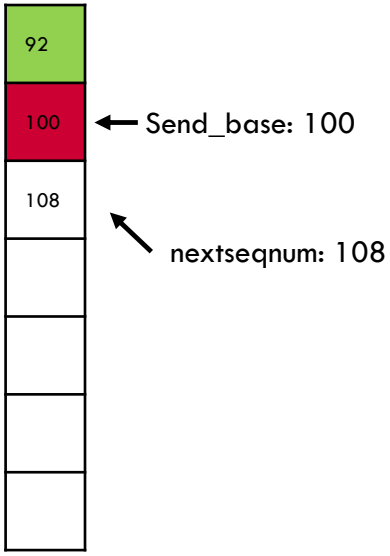


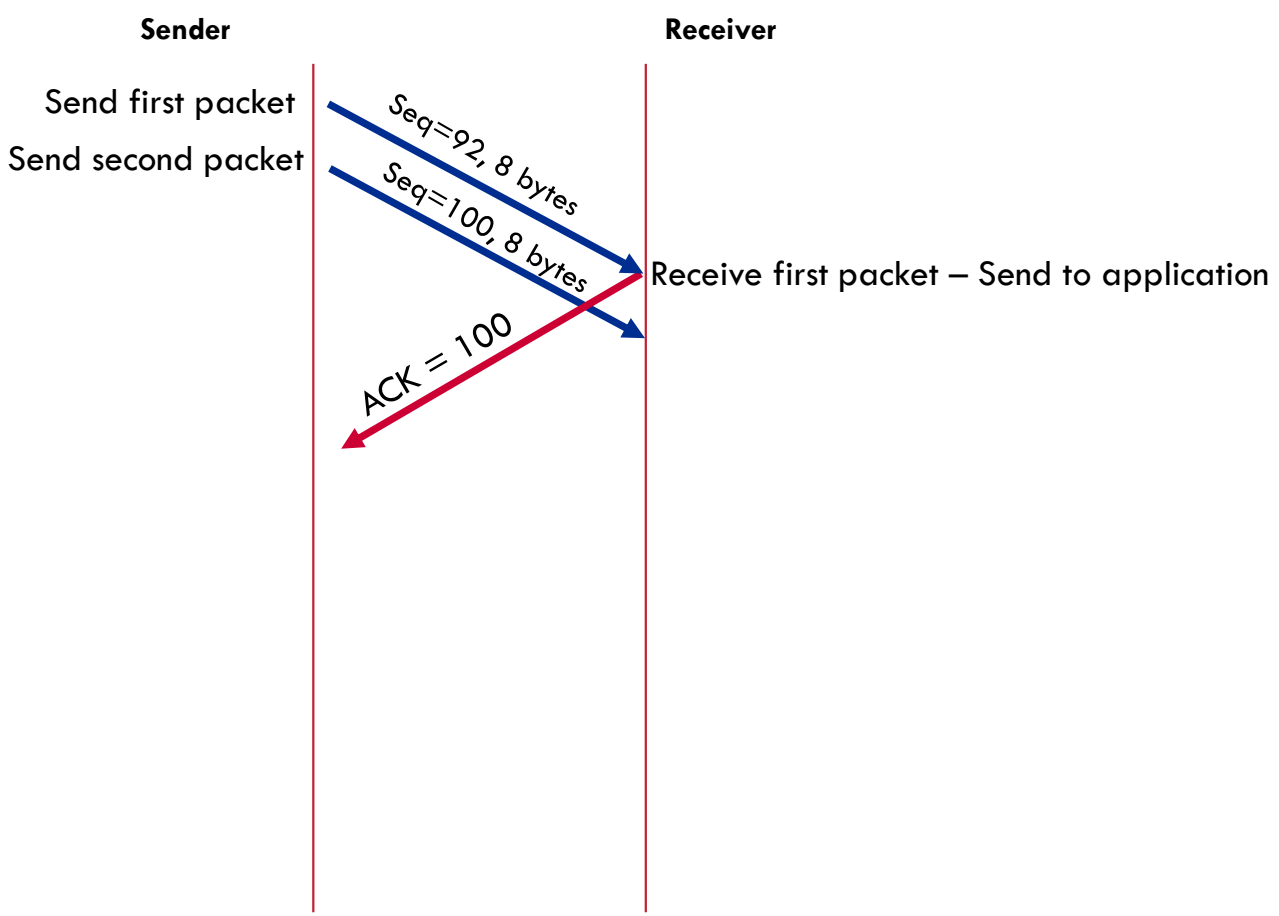




TCP: Fault Free Scenario

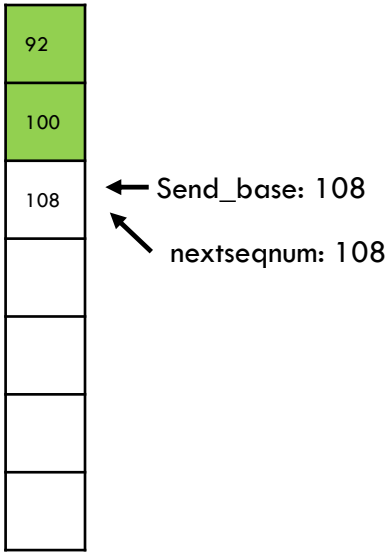
Assume send_base and nextseqnum are randomly set to 92

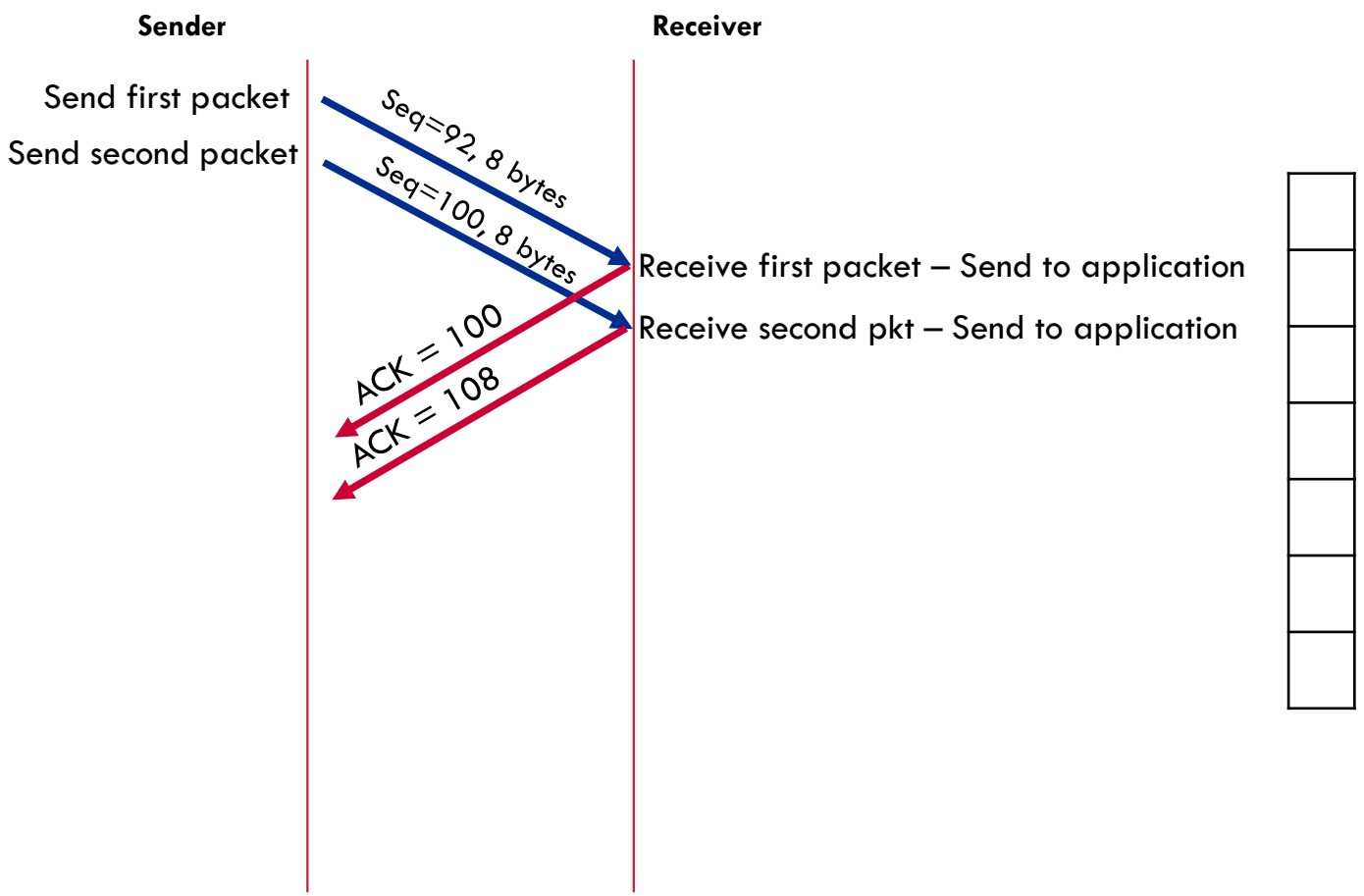




TCP: Fault Free Scenario

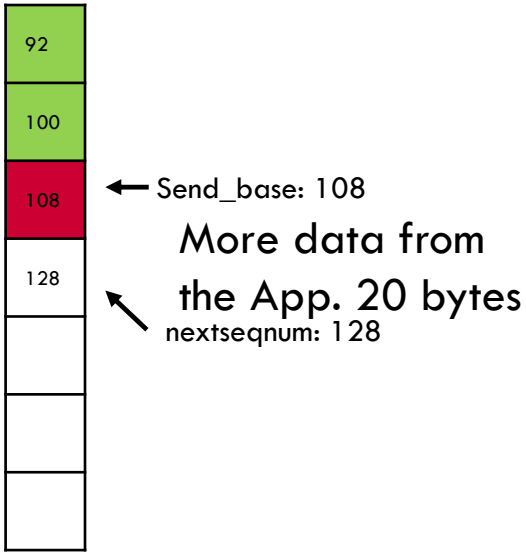
Assume send_base and nextseqnum are randomly set to 92

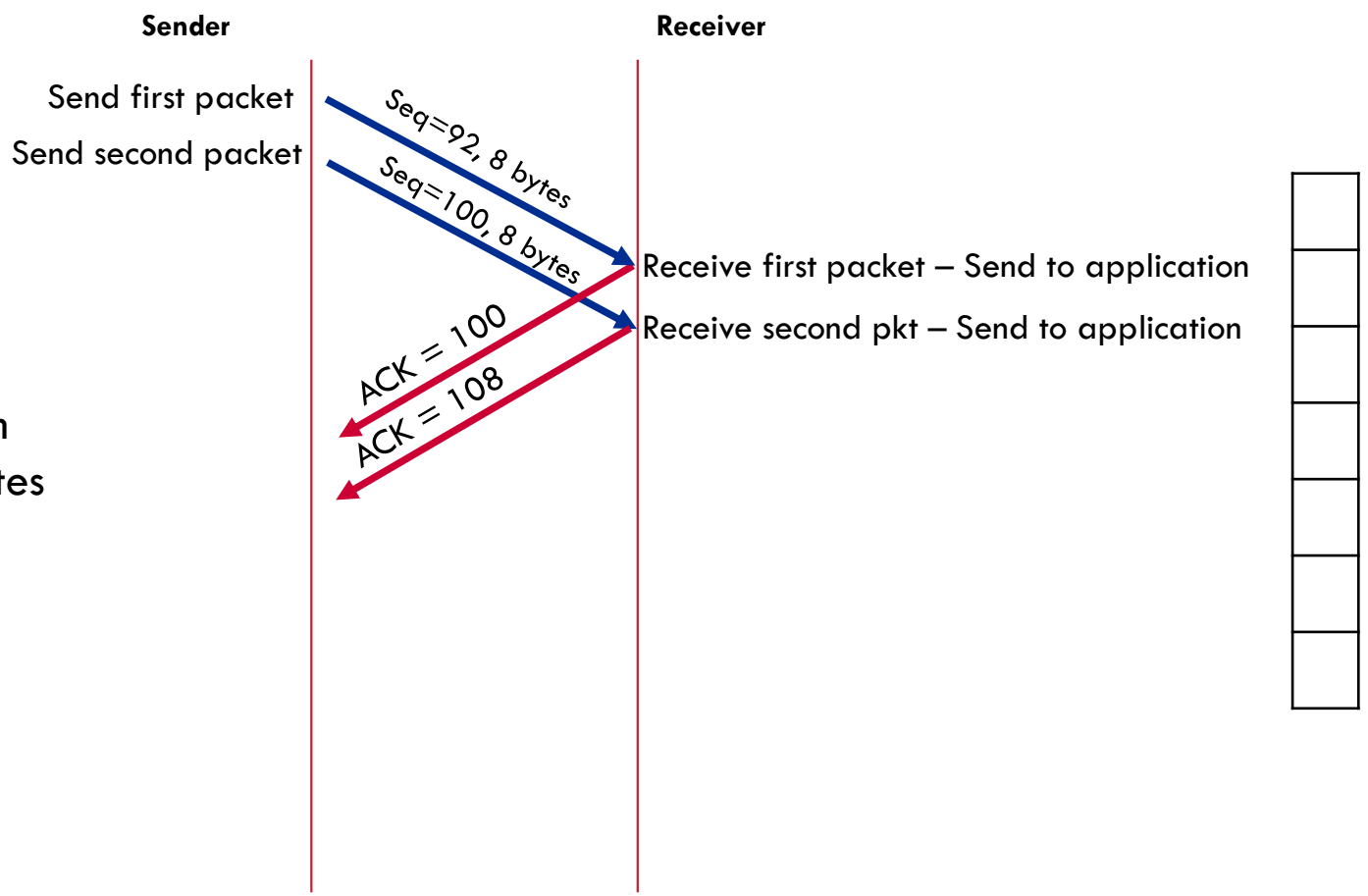




TCP: Fault Free Scenario

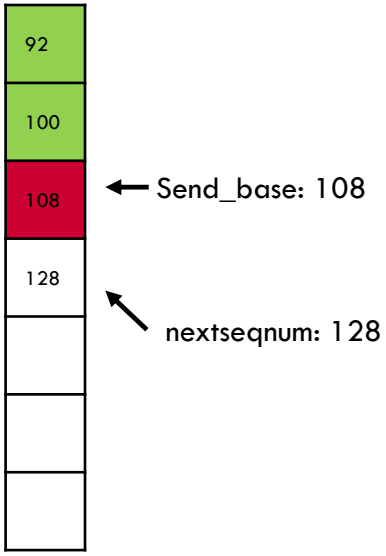
Assume send_base and nextseqnum are randomly set to 92

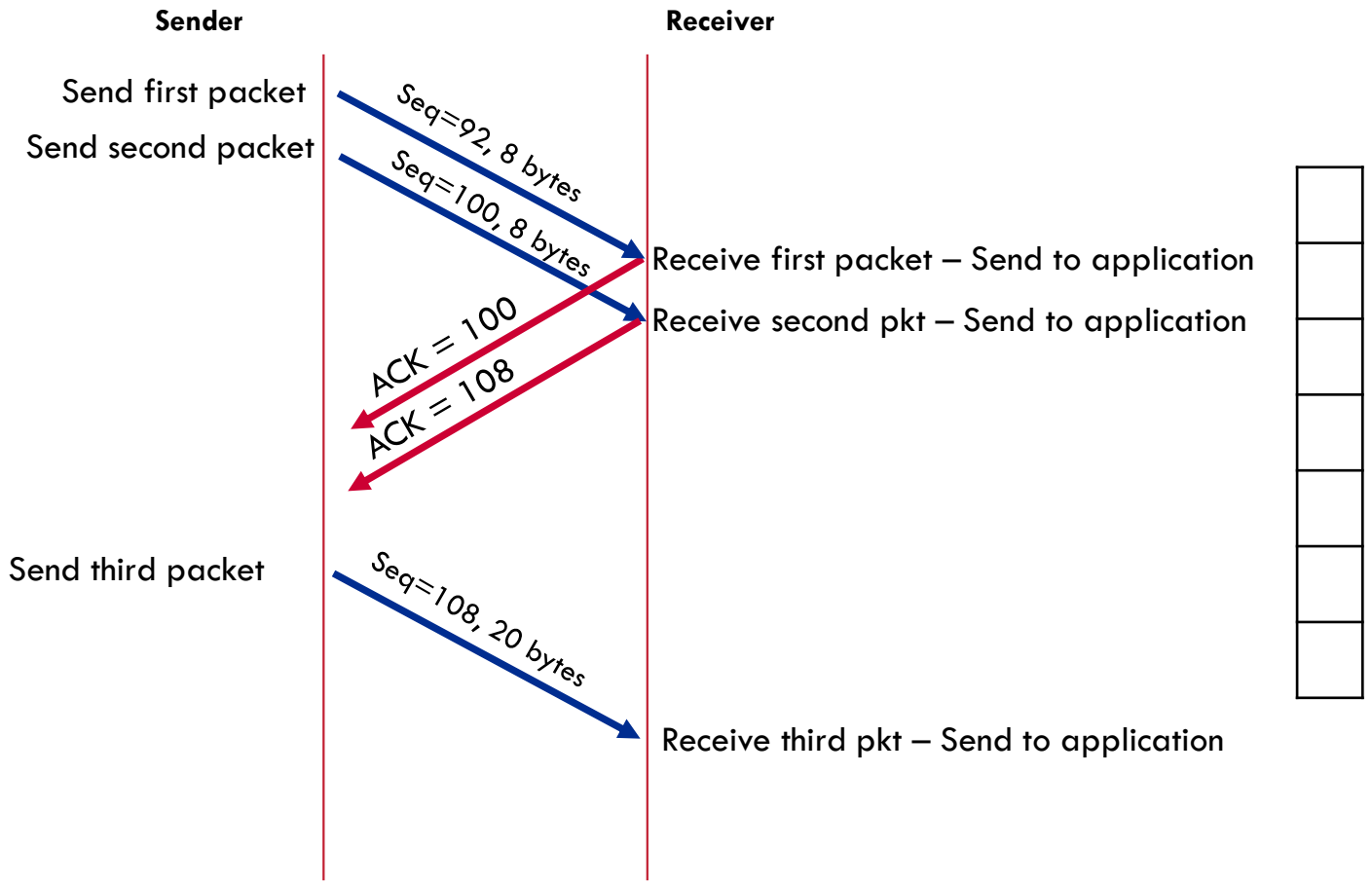




TCP: Fault Free Scenario

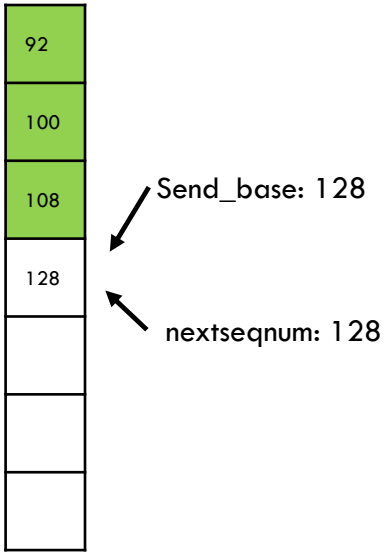
Assume send_base and nextseqnum are randomly set to 92

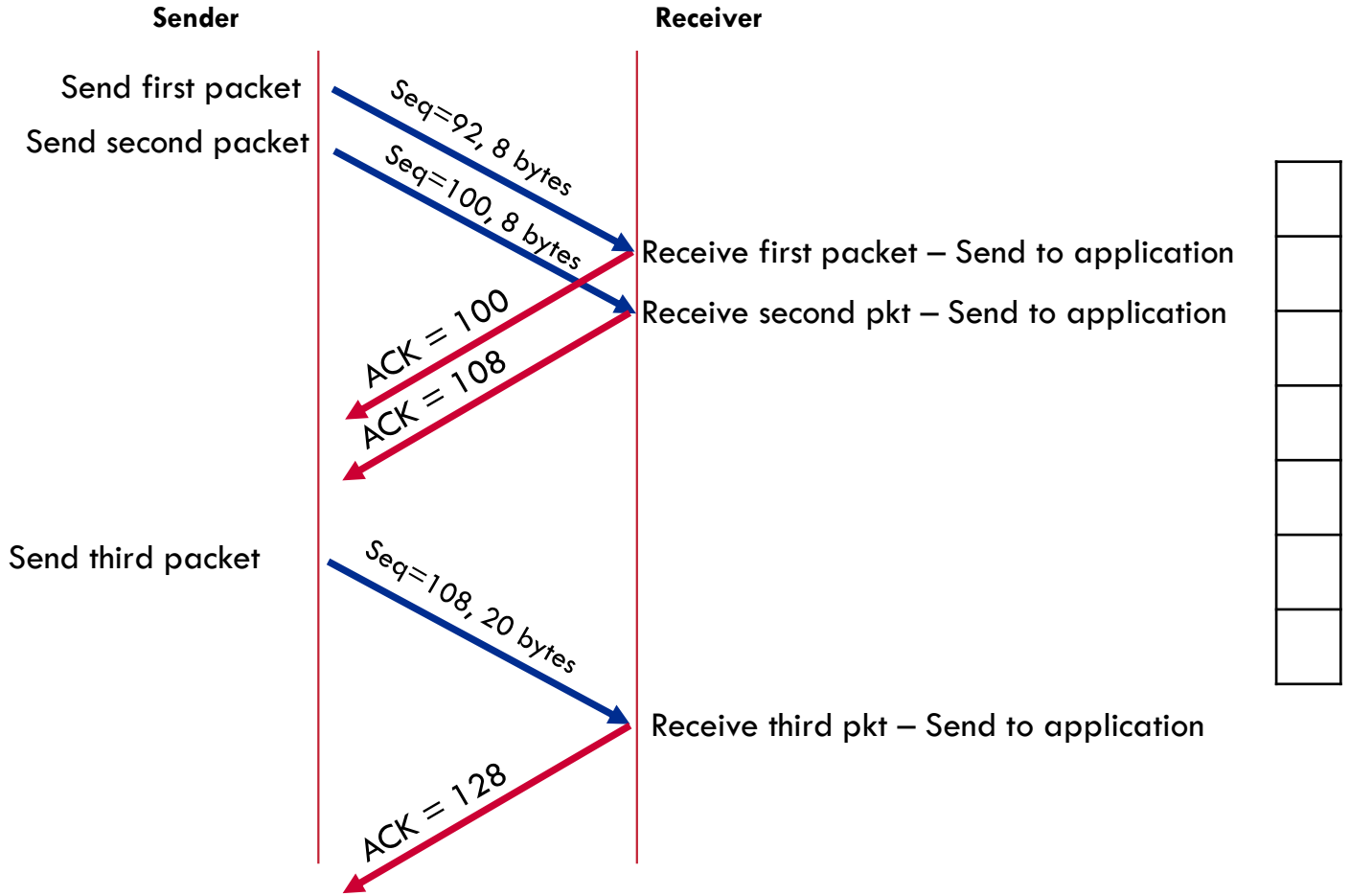




TCP: Fault Free Scenario

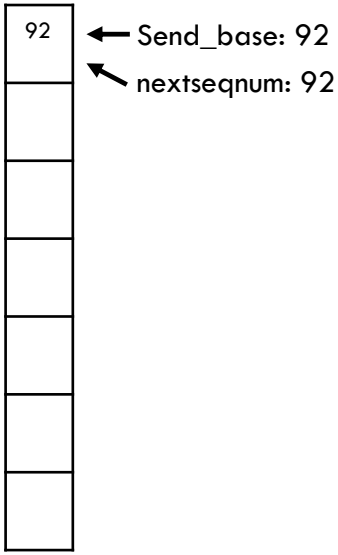
Assume send_base and nextseqnum are randomly set to 92





TCP: Lost Segment Scenario

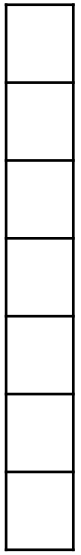
Assume send_base and nextseqnum are randomly set to 92



Sender

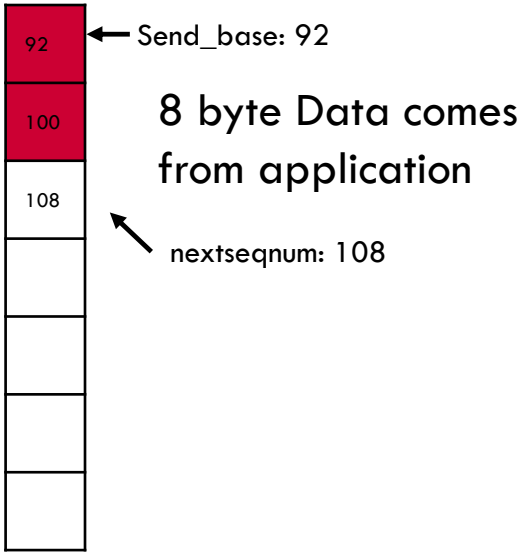
Receiver

During set up: Receiver knows that the next number is 92



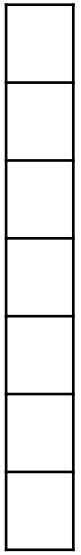
TCP: Lost Segment Scenario

Assume send_base and nextseqnum are randomly set to 92



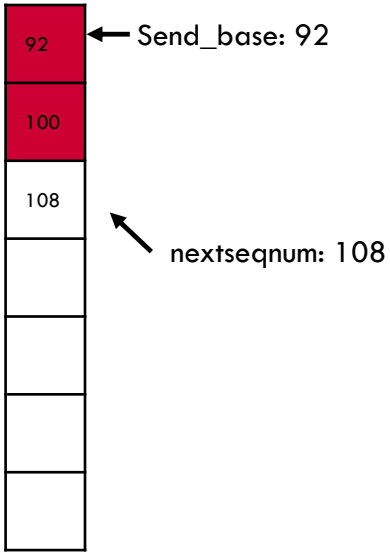
Sender **Receiver**

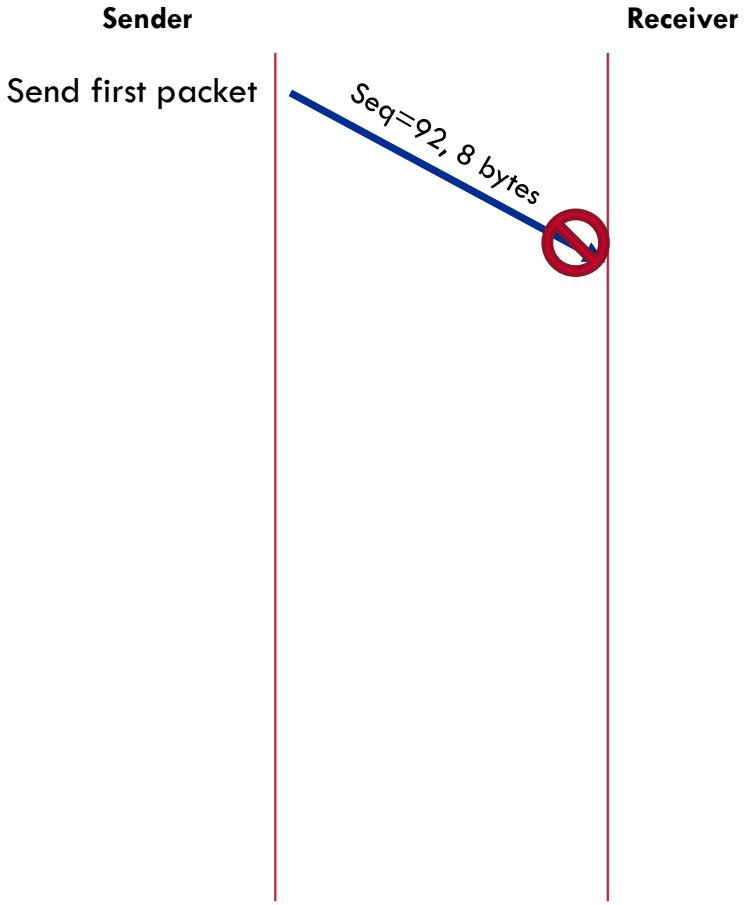
During set up: Receiver knows that the next number is 92



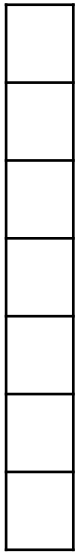
TCP: Lost Segment Scenario

Assume send_base and nextseqnum are randomly set to 92



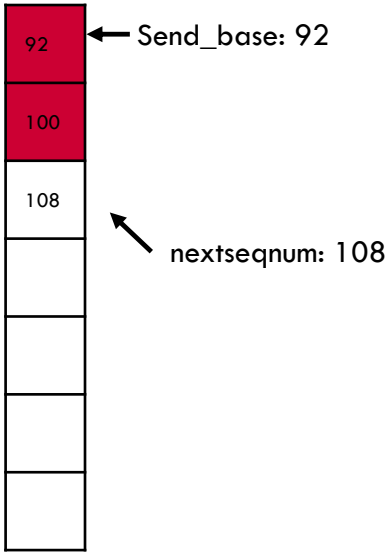


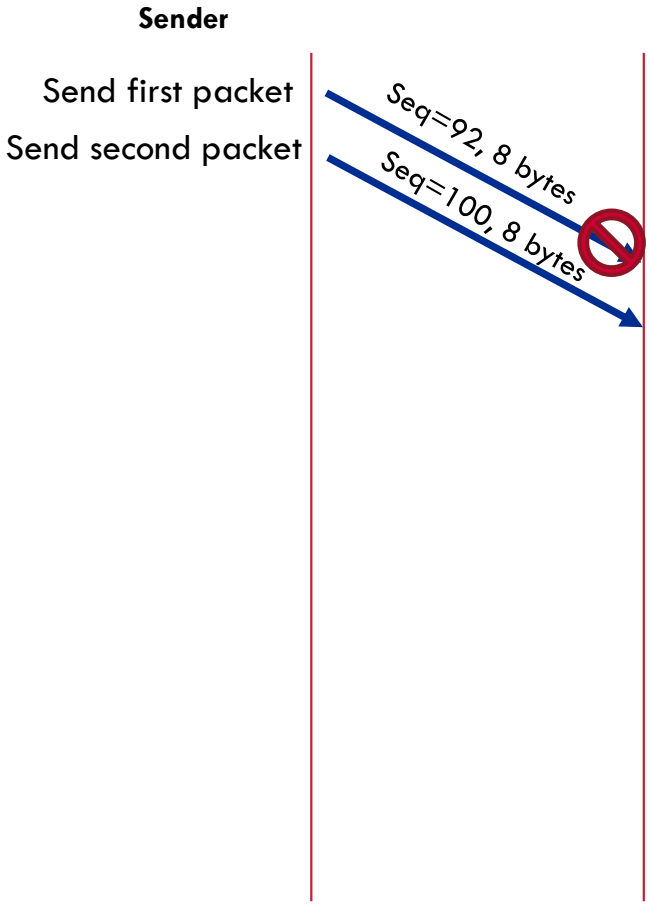
During set up: Receiver knows that the next number is 92



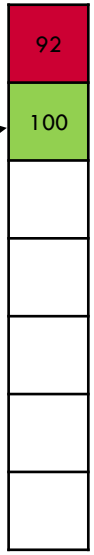
TCP: Lost Segment Scenario

Assume send_base and nextseqnum are randomly set to 92



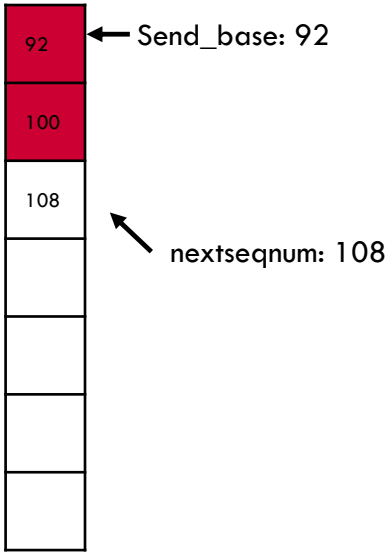


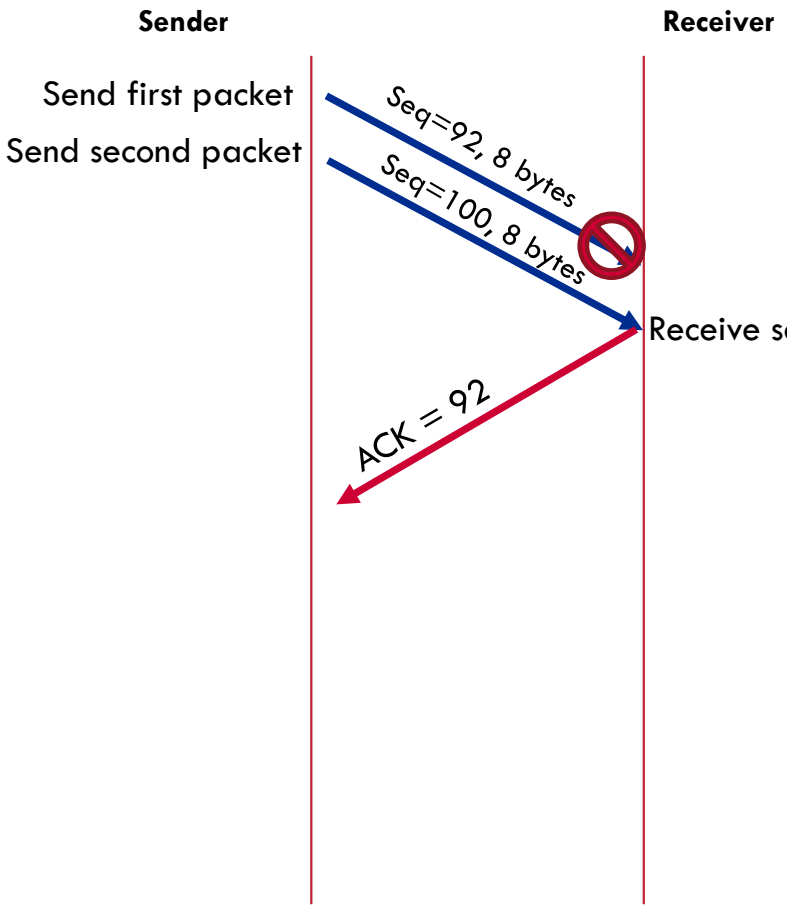
During set up: Receiver knows that the next number is 92



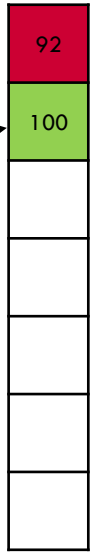
TCP: Lost Segment Scenario

Assume send_base and nextseqnum are randomly set to 92



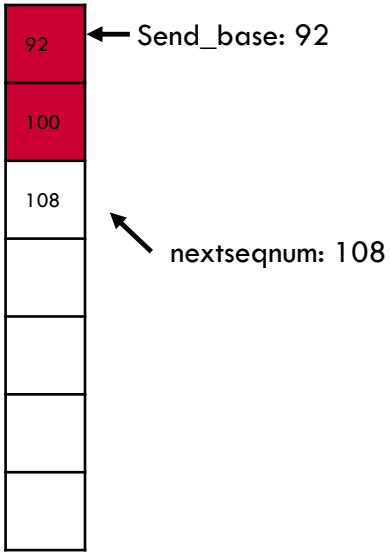


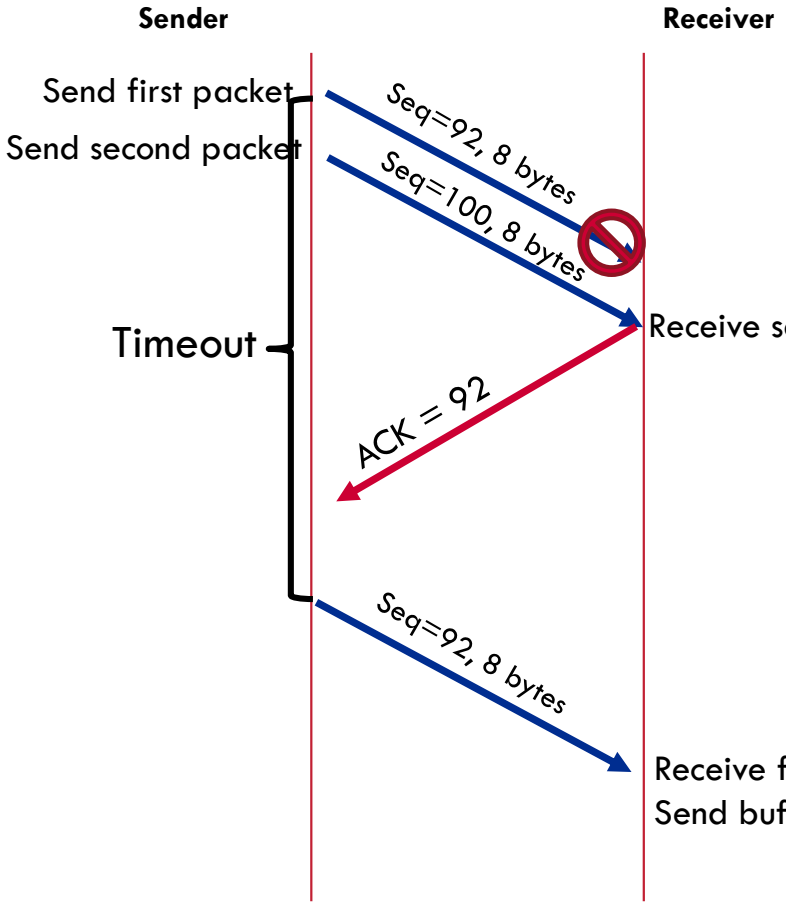
During set up: Receiver knows that the next number is 92



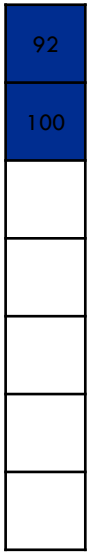
TCP: Lost Segment Scenario

Assume send_base and nextseqnum are randomly set to 92



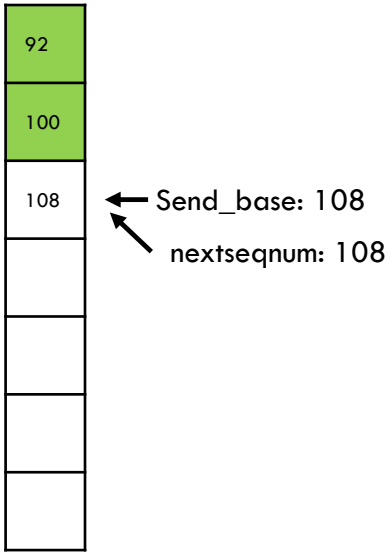


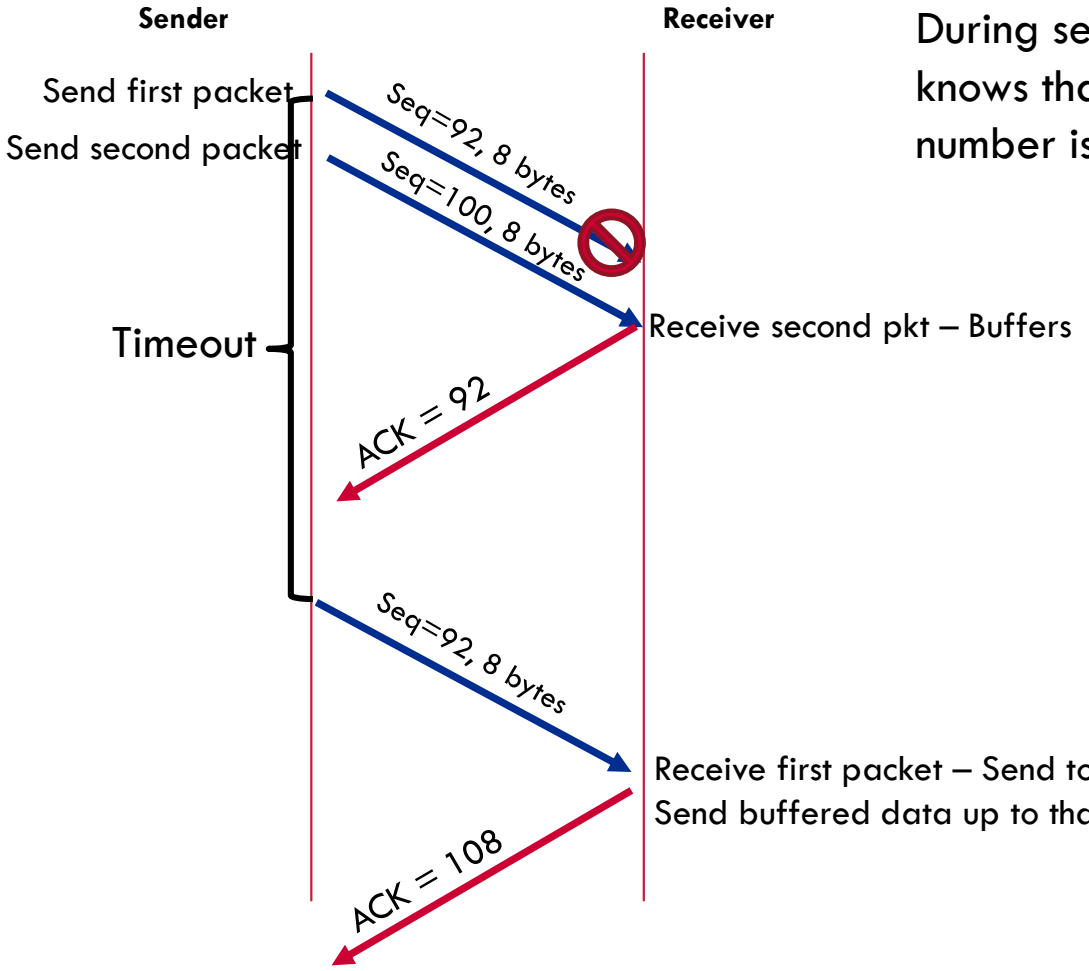
During set up: Receiver knows that the next number is 92



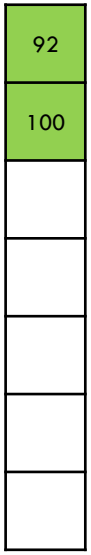
TCP: Lost Segment Scenario

Assume send_base and nextseqnum are randomly set to 92



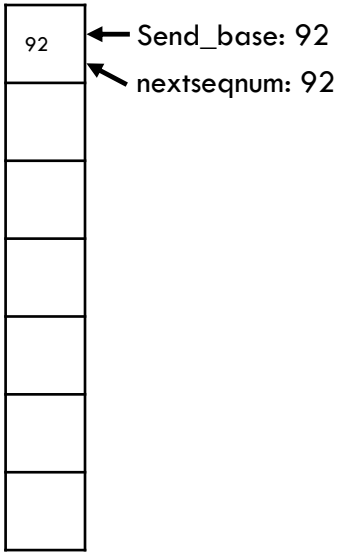


During set up: Receiver knows that the next number is 92



TCP: Lost ACK Scenario

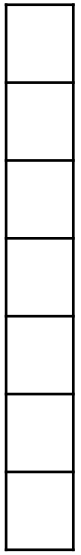
Assume send_base and nextseqnum are randomly set to 92



Sender

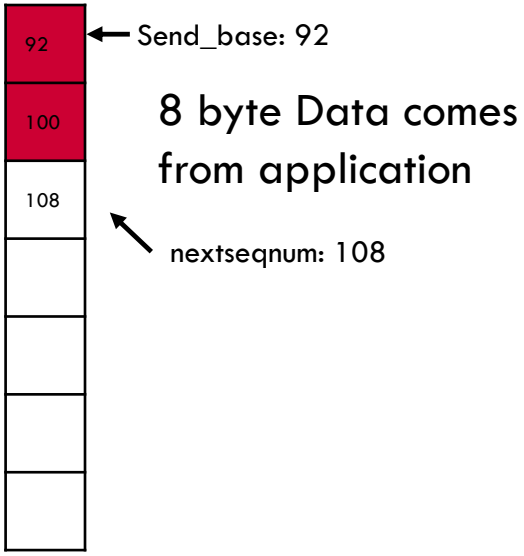
Receiver

rcv_base: 92



TCP: Lost ACK Scenario

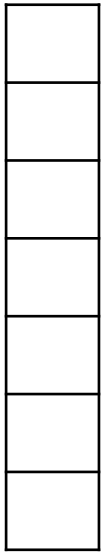
Assume send_base and nextseqnum are randomly set to 92



Sender

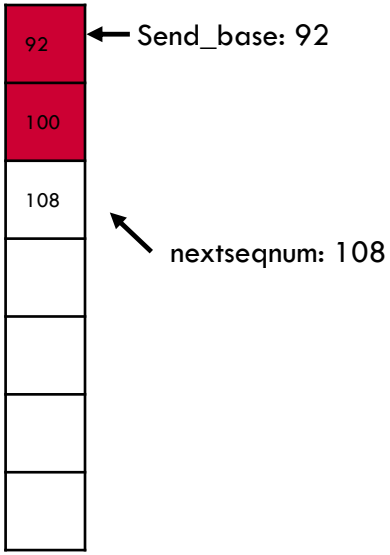
Receiver

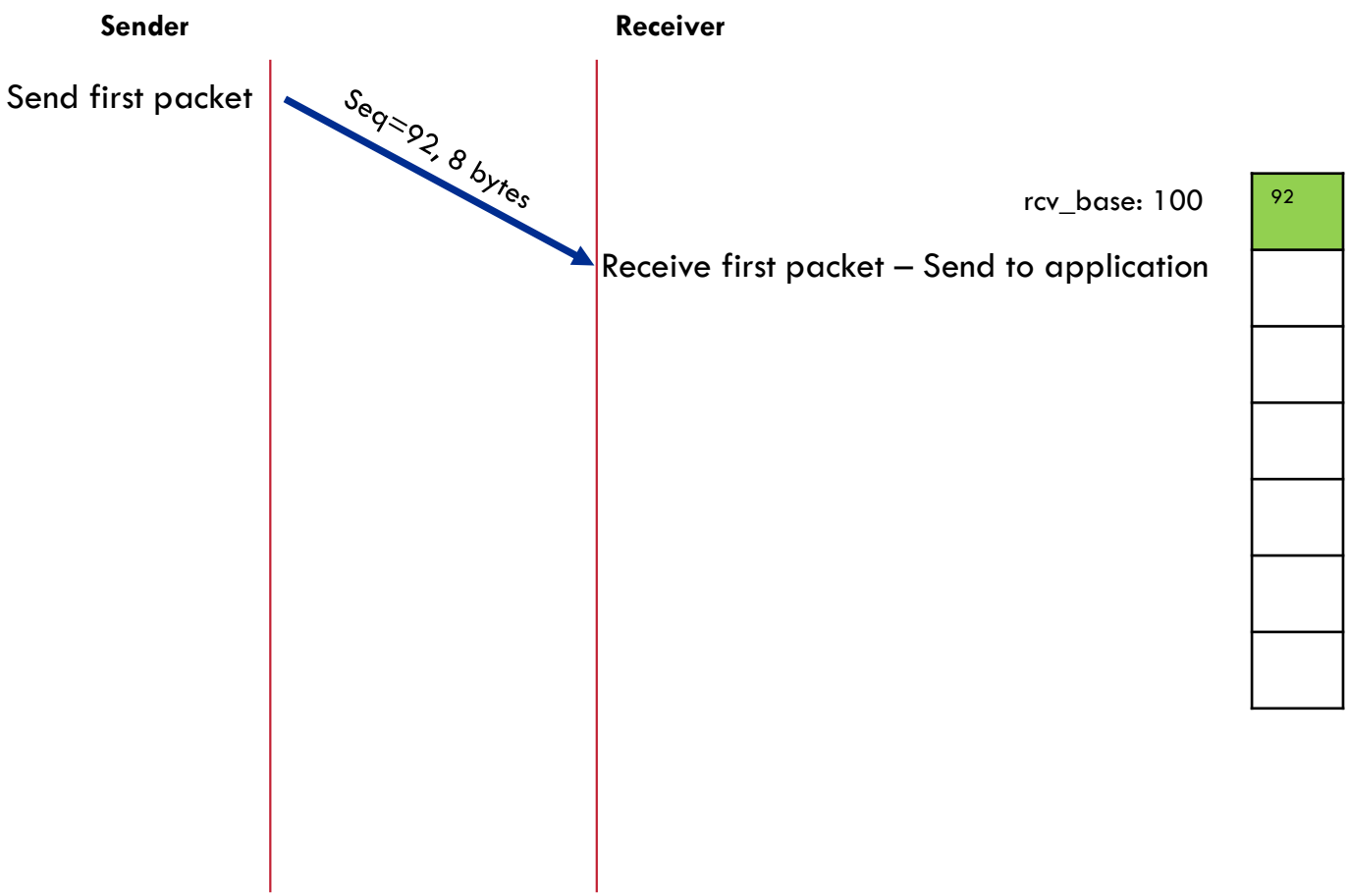
rcv_base: 92



TCP: Lost ACK Scenario

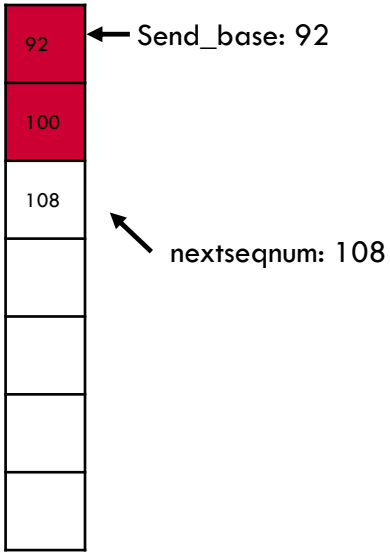
Assume send_base and nextseqnum are randomly set to 92

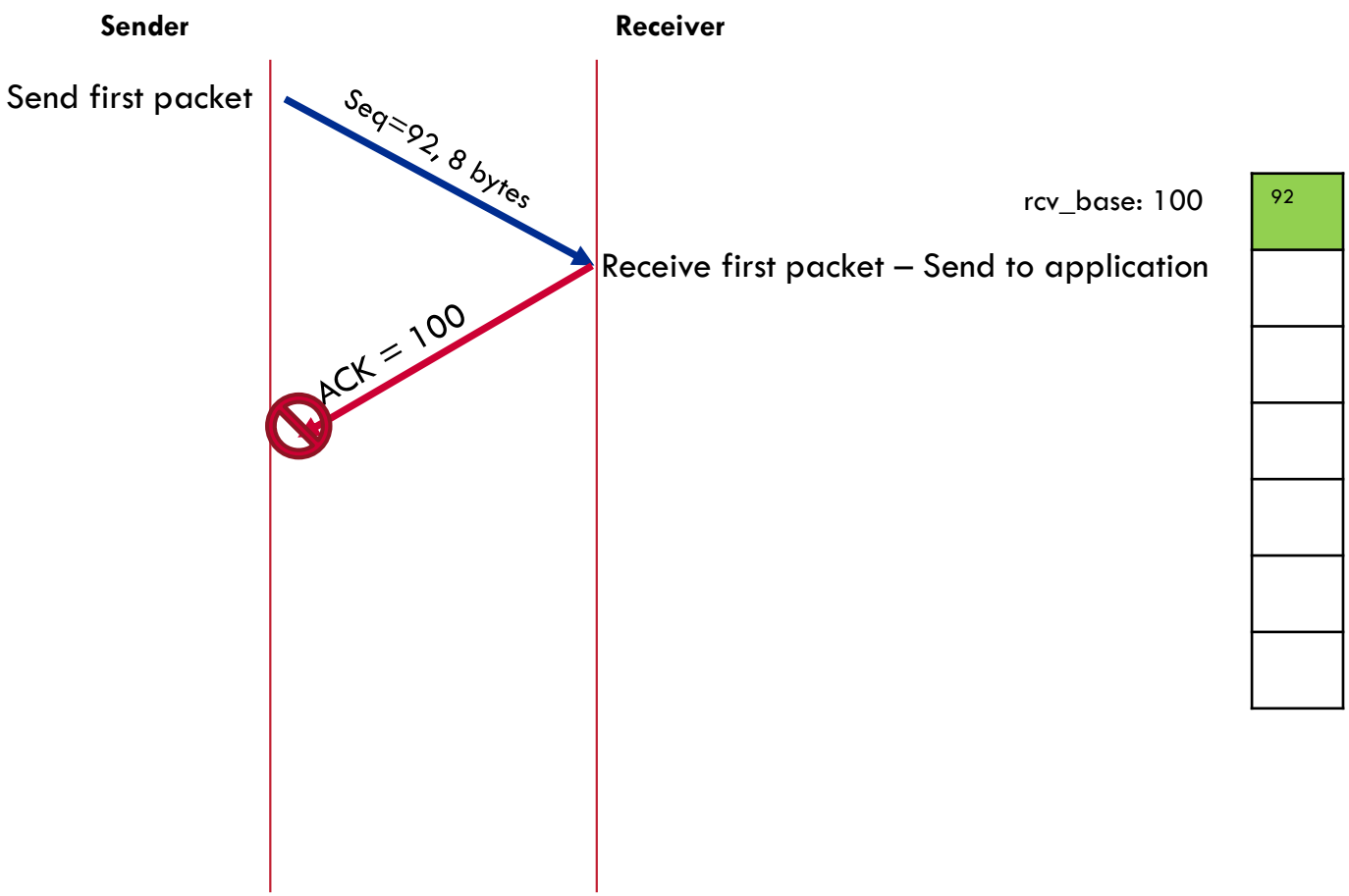




TCP: Lost ACK Scenario

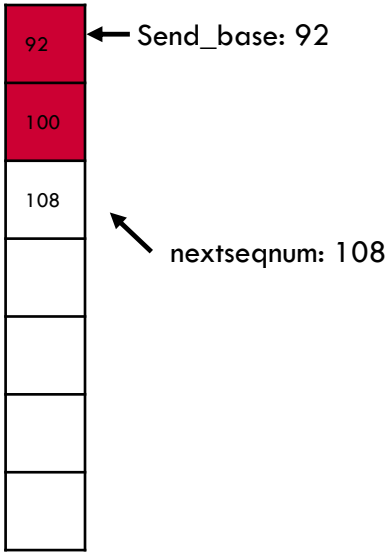
Assume send_base and nextseqnum are randomly set to 92

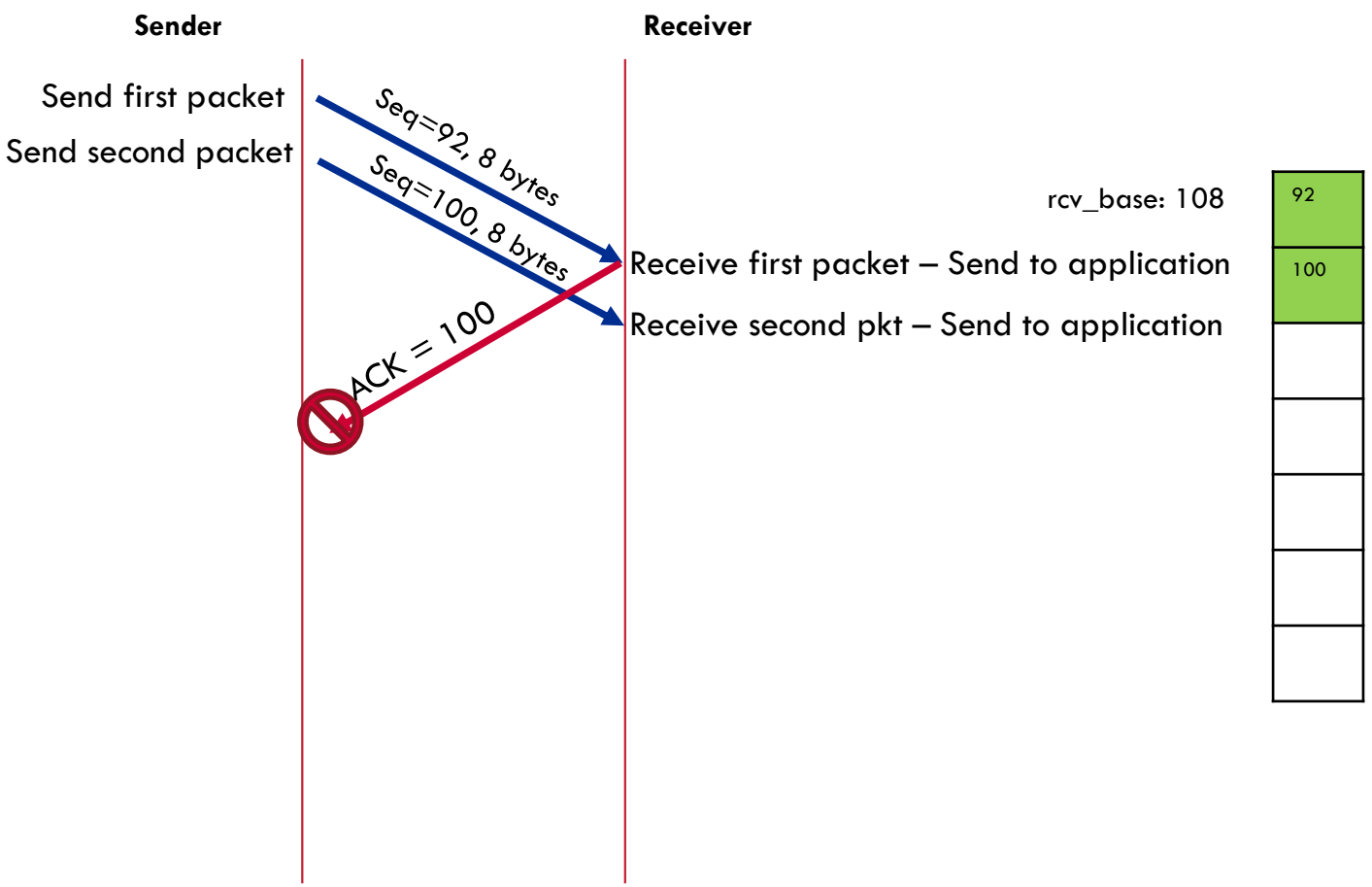




TCP: Lost ACK Scenario

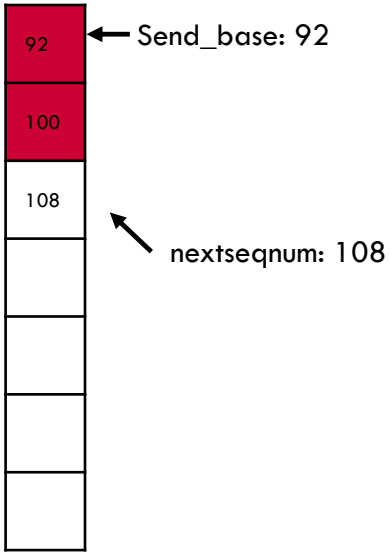
Assume send_base and nextseqnum are randomly set to 92

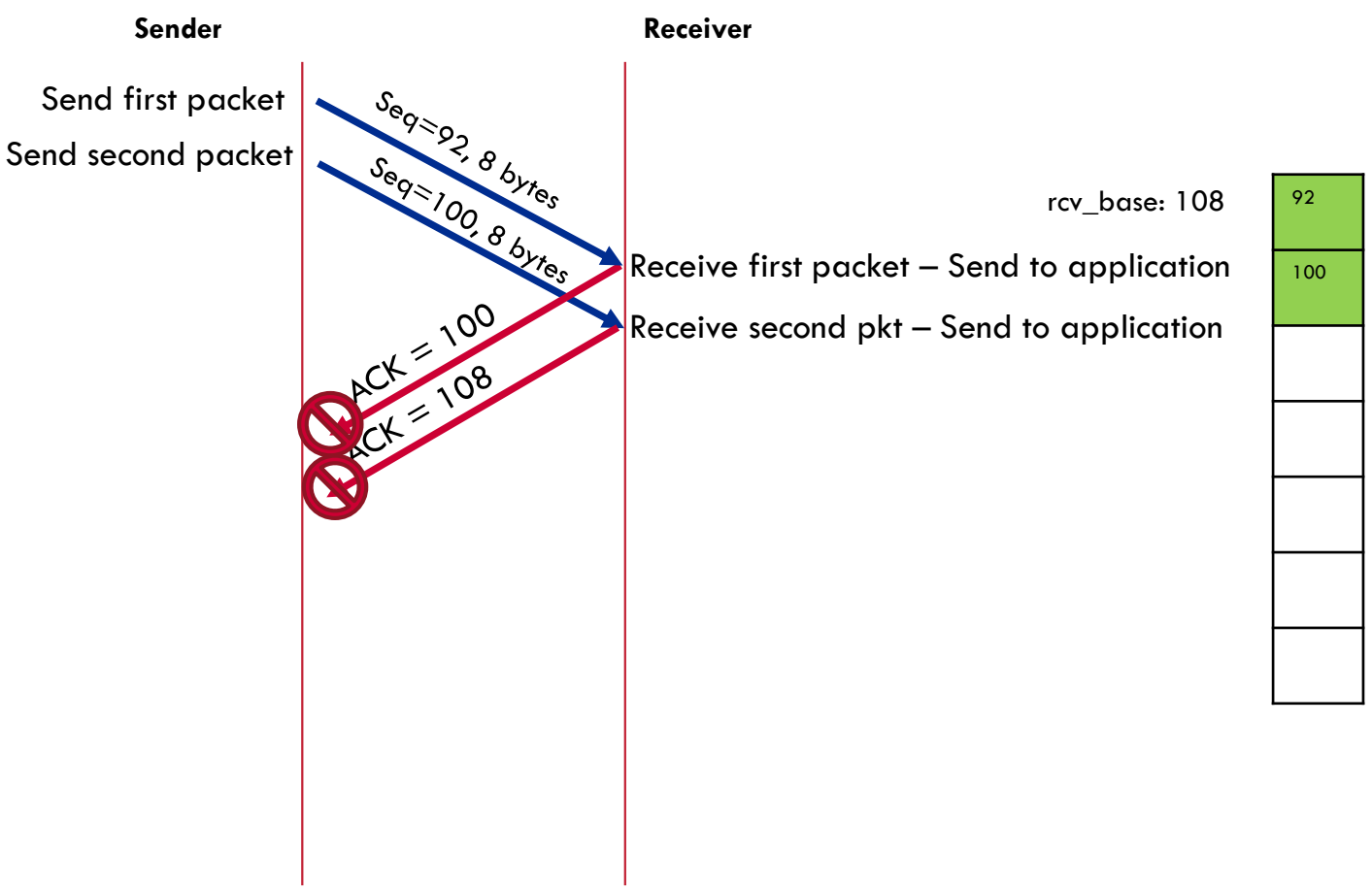




TCP: Lost ACK Scenario

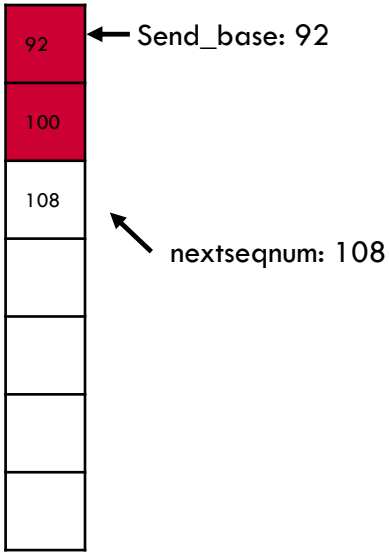
Assume send_base and nextseqnum are randomly set to 92

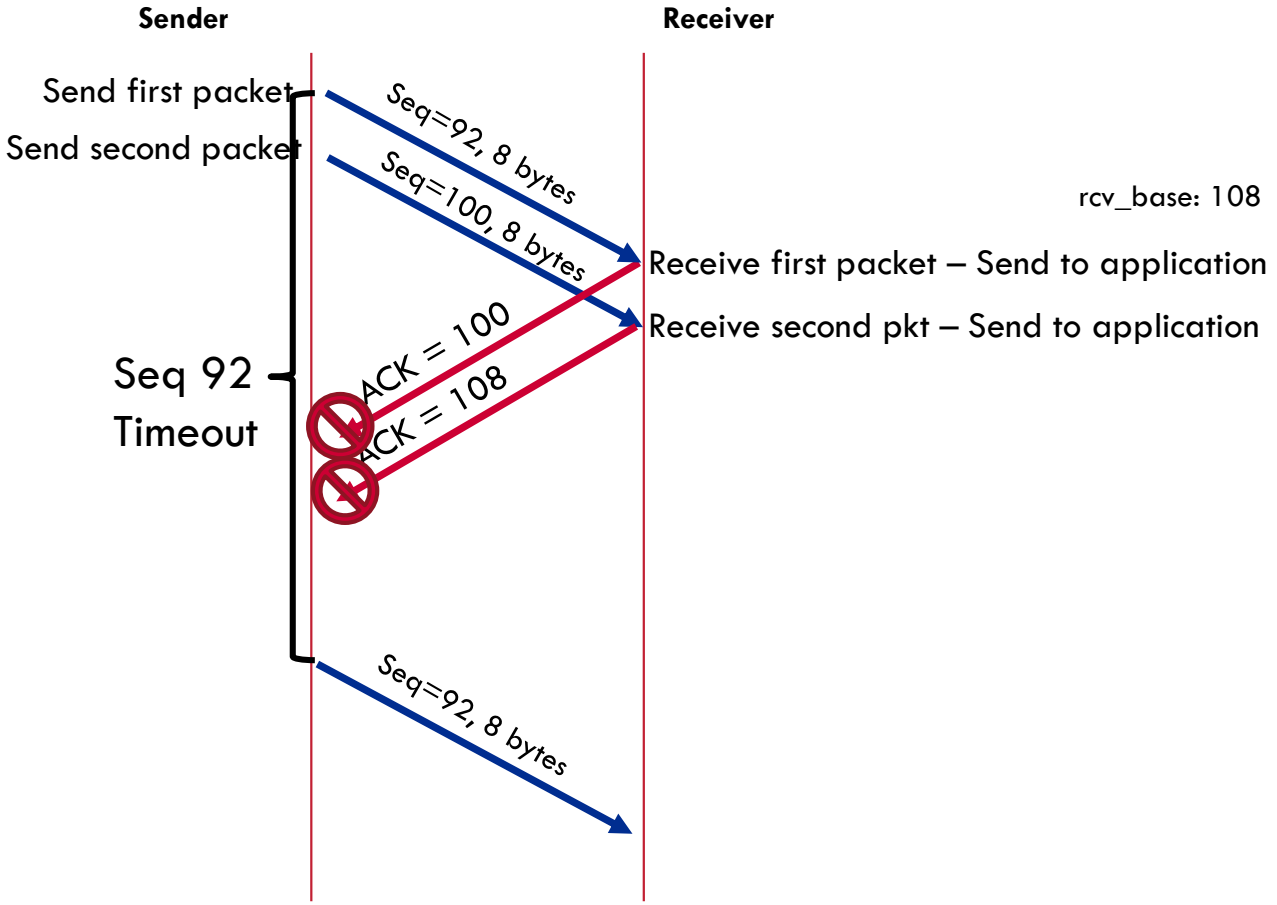




TCP: Lost ACK Scenario

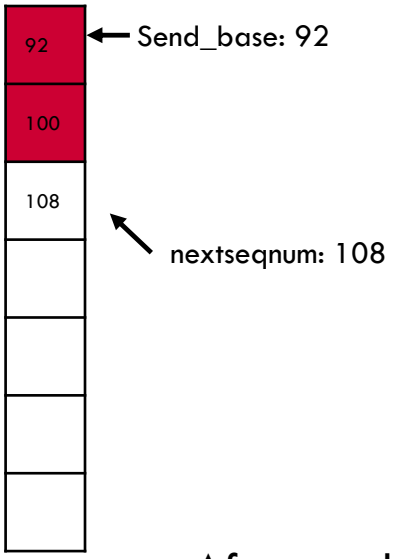
Assume send_base and nextseqnum are randomly set to 92

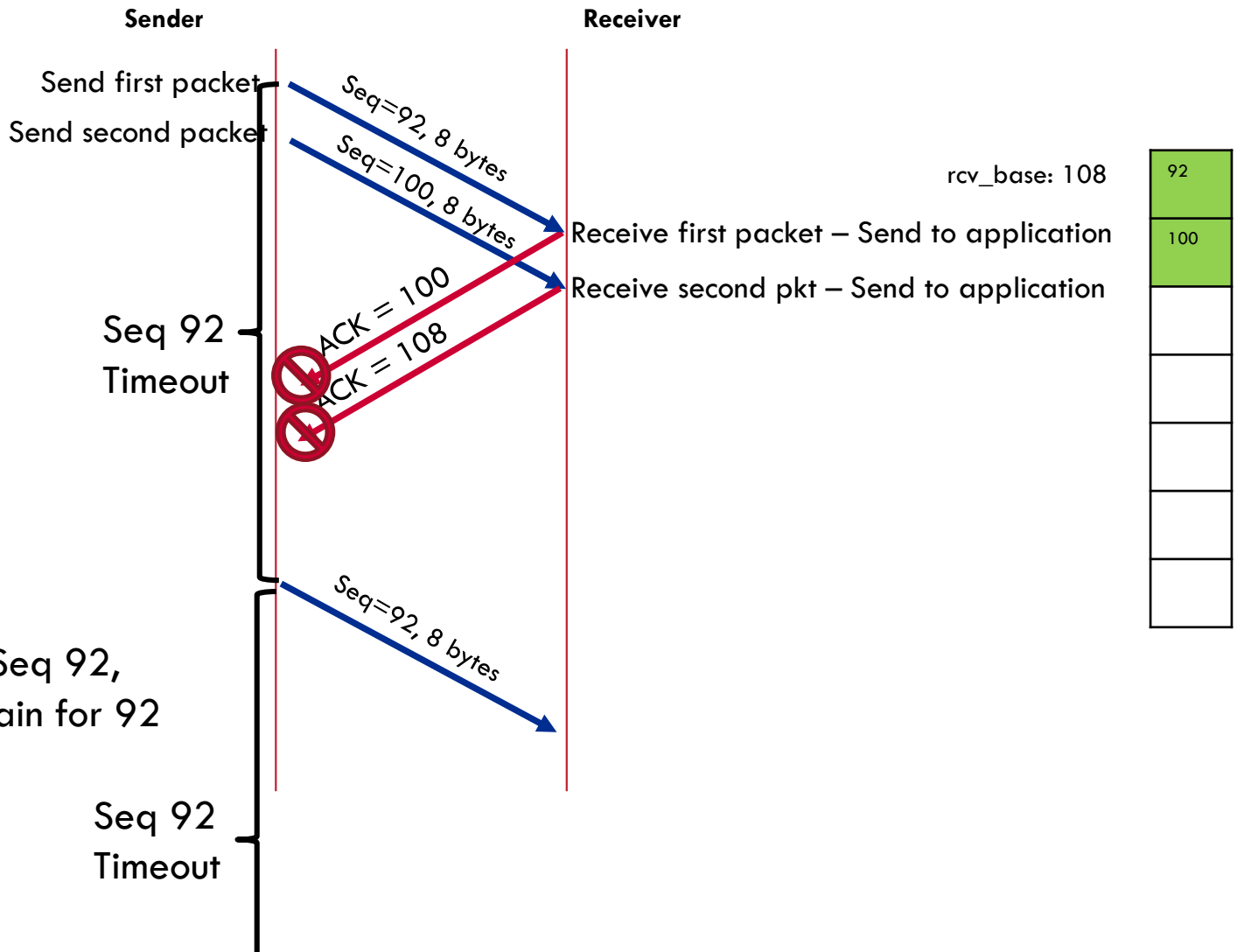




TCP: Lost ACK Scenario

Assume send_base and nextseqnum are randomly set to 92

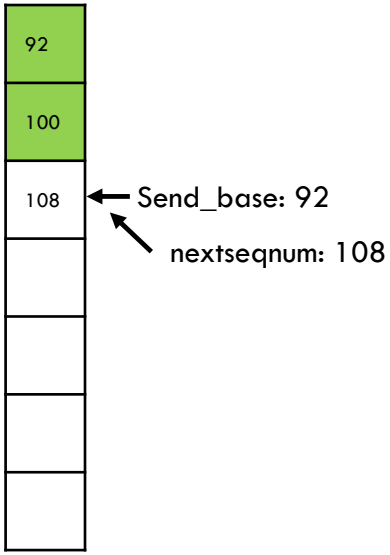


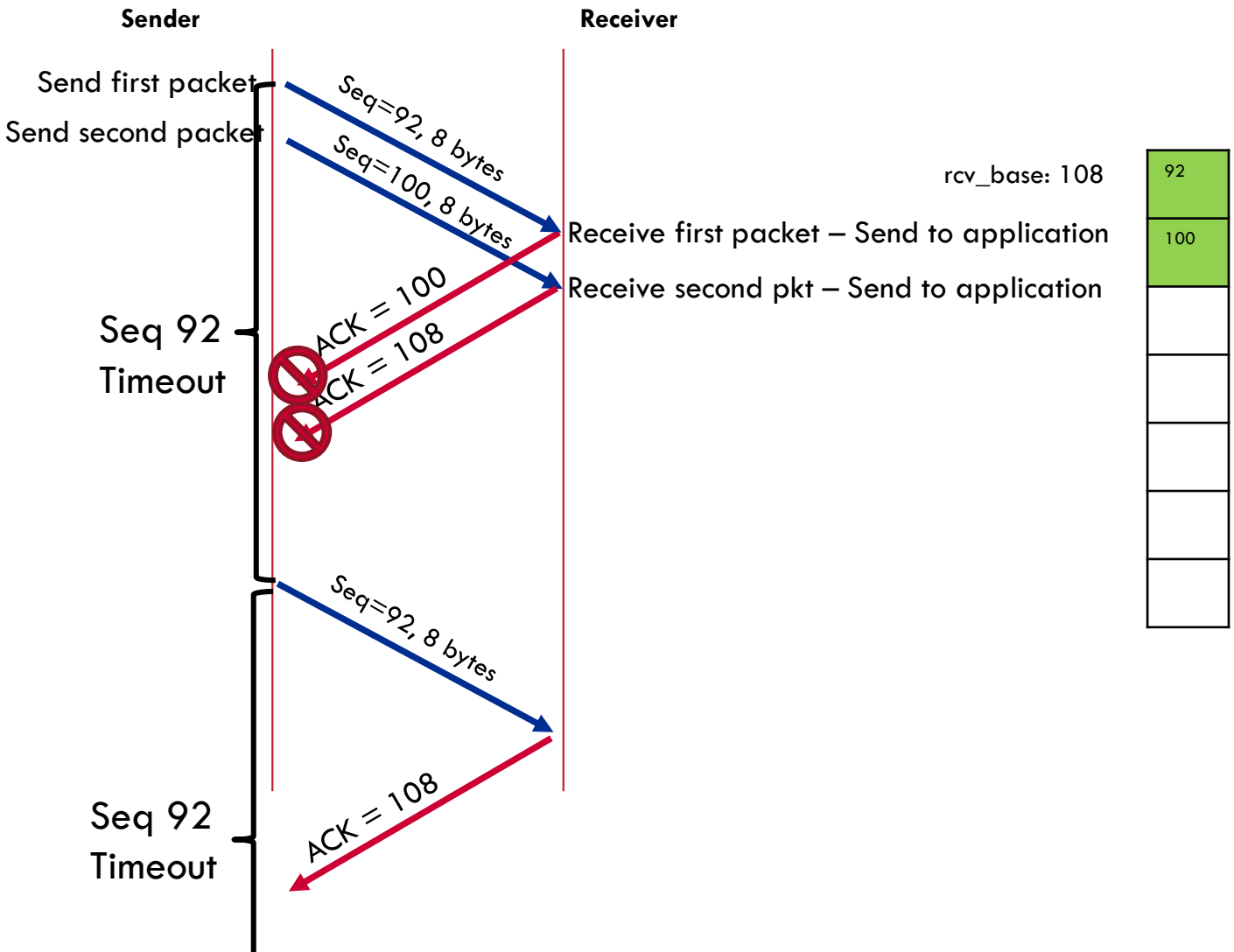


After sending Seq 92, timer starts again for 92

TCP: Lost ACK Scenario

Assume send_base and nextseqnum are randomly set to 92

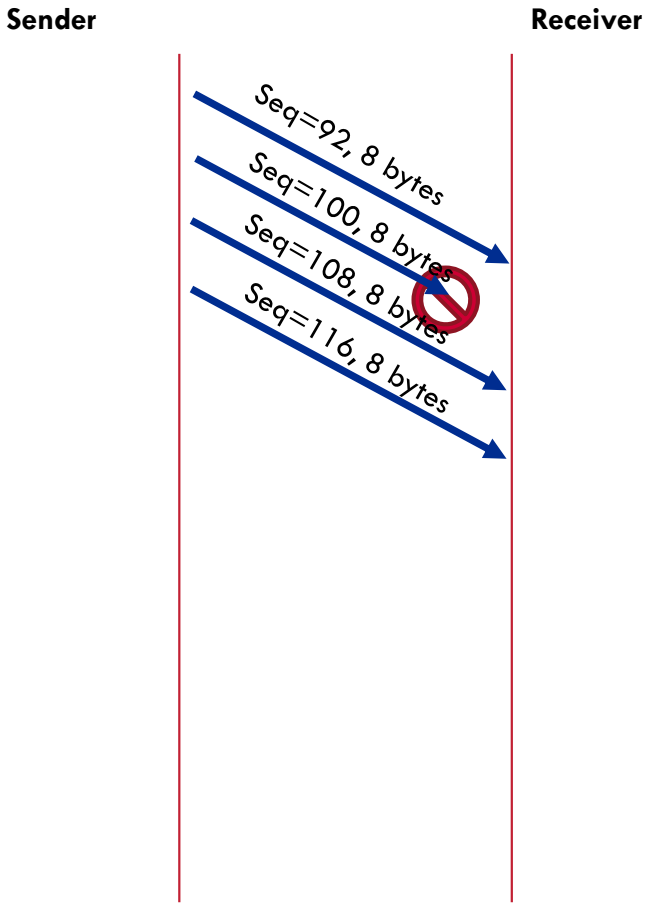




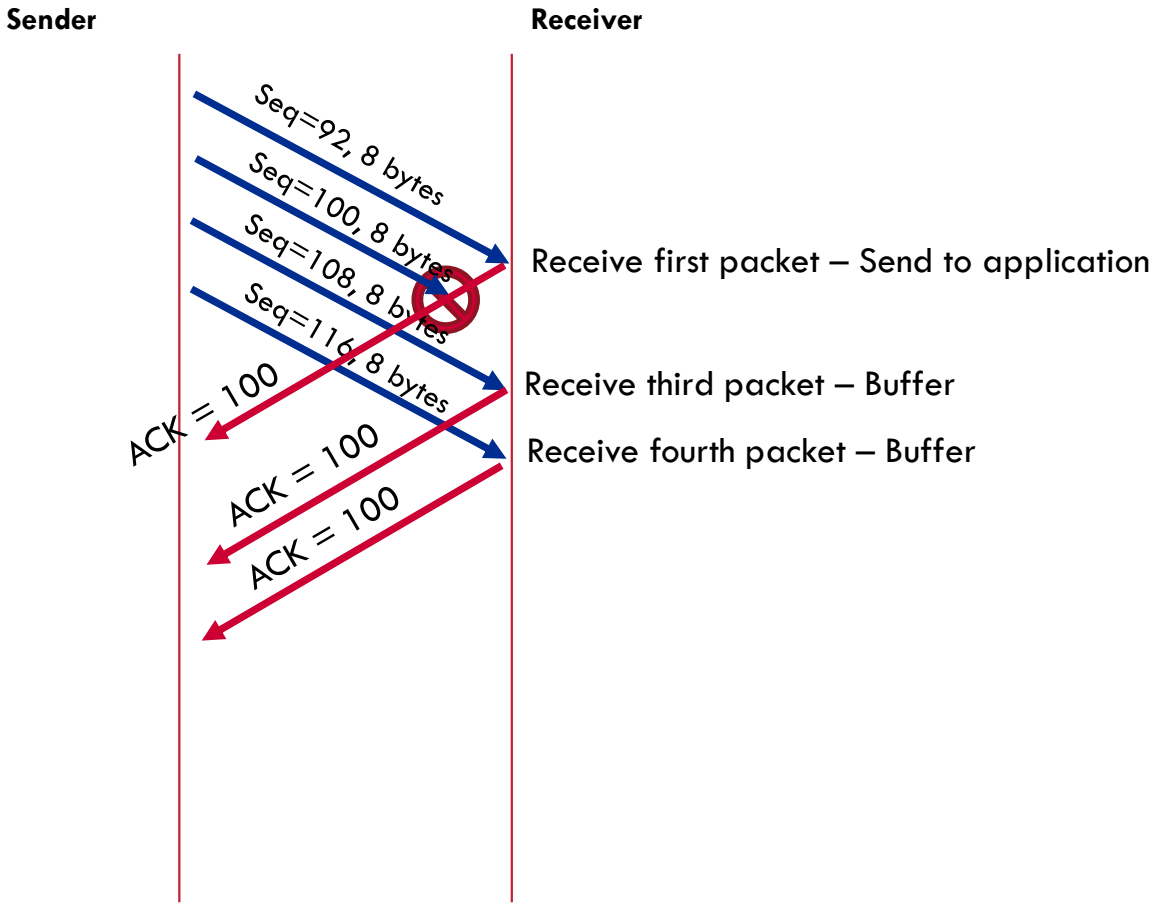
TCP Fast Retransmit

- Timeouts need to be necessarily long (propagation around the earth)
- Fast retransmit: immediately resend segment if 3 duplicate ACKs are received
  - Normally resends are only based on timeout, not ACK

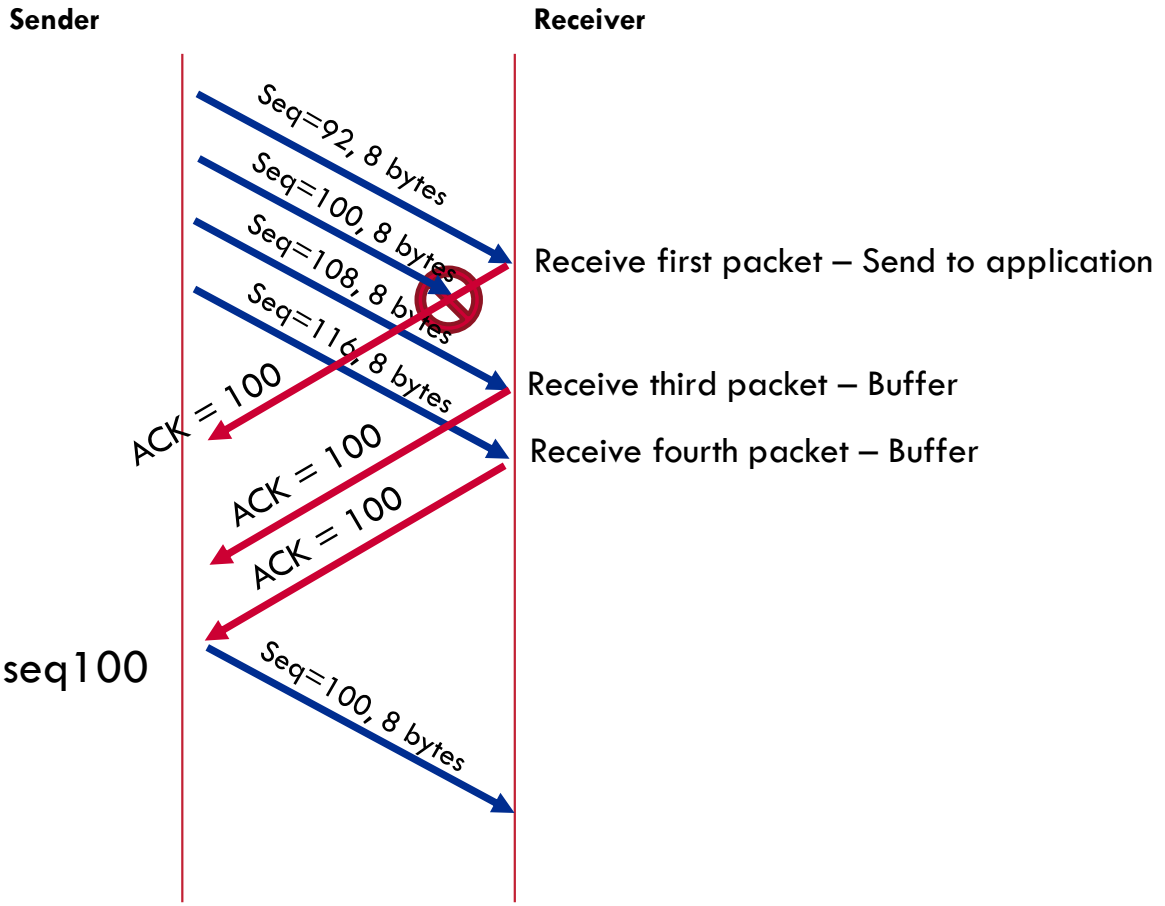
TCP: Triple ACK Scenario



TCP: Triple ACK Scenario

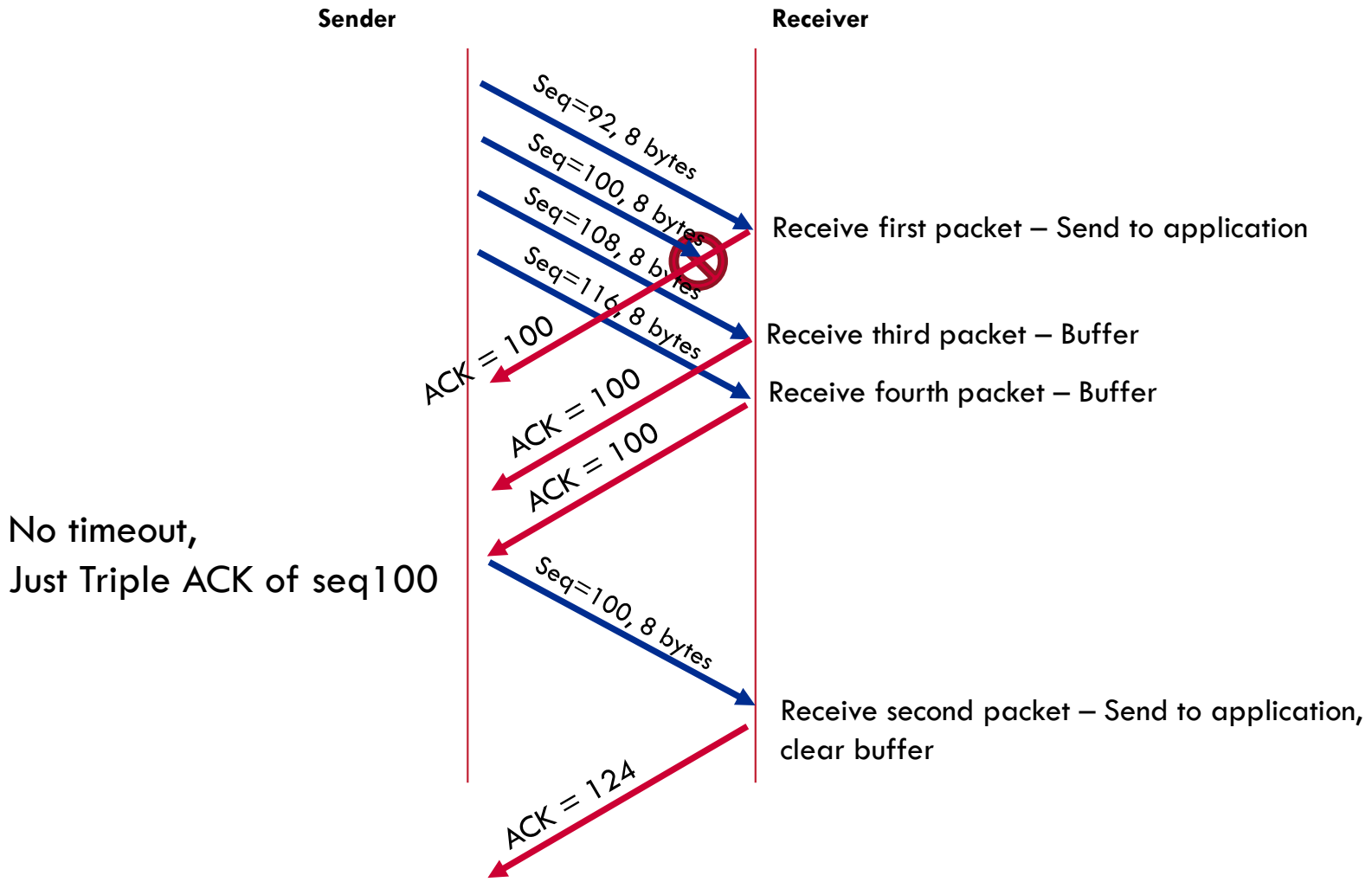


TCP: Triple ACK Scenario



No timeout,
Just Triple ACK of seq100

TCP: Triple ACK Scenario

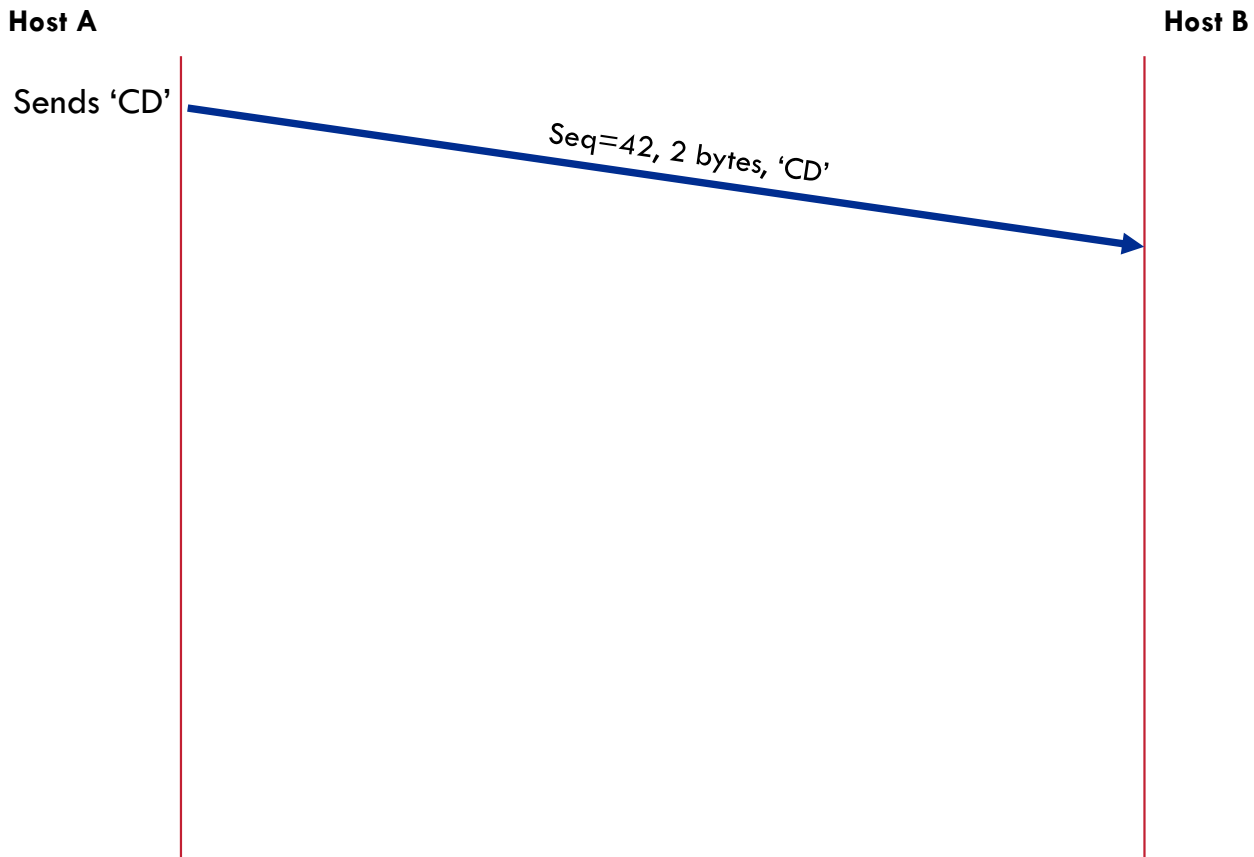


TCP Full Duplex

- TCP Segments include sequence number of data being carried and expected sequence as ACK

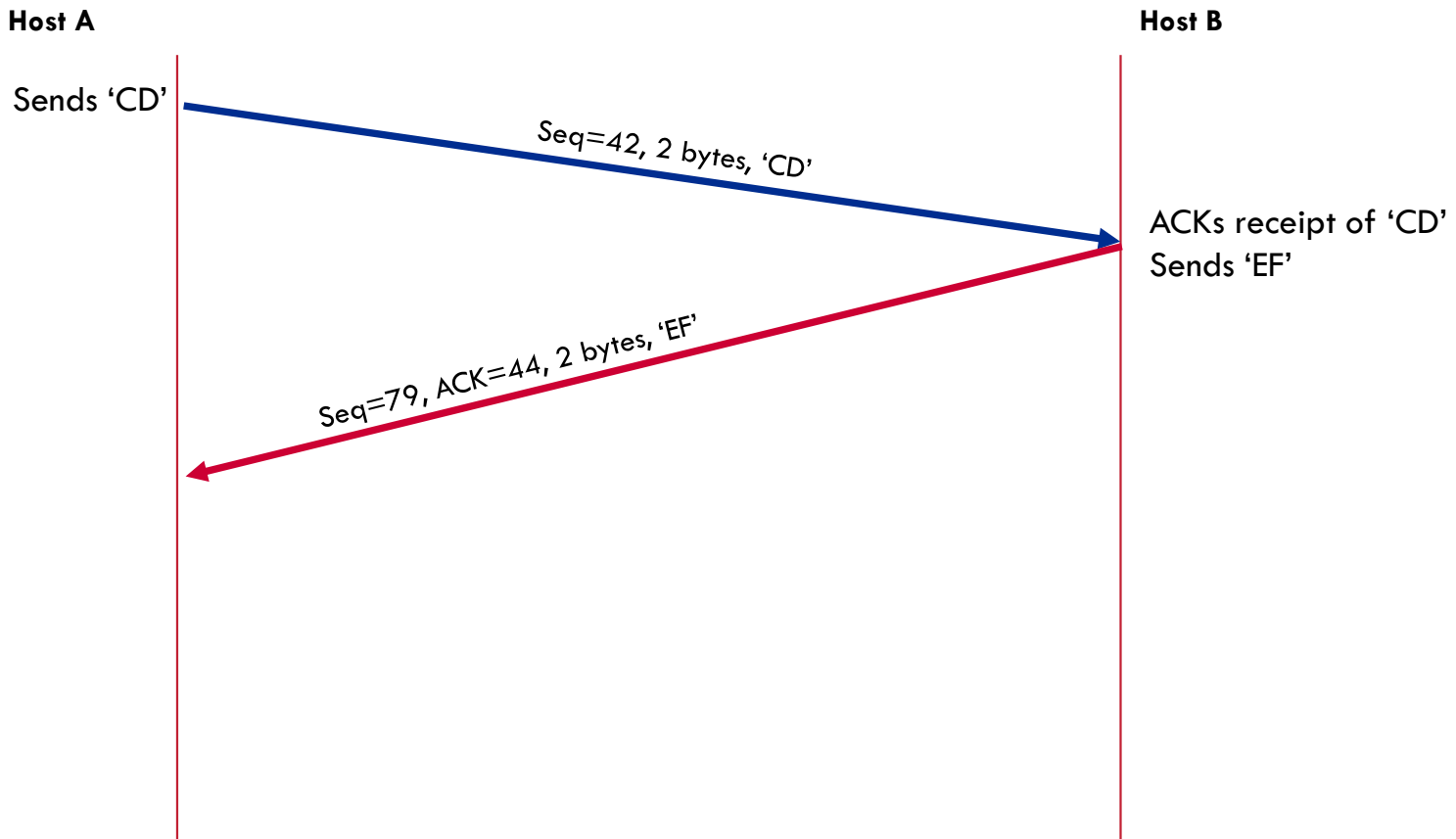
TCP Full Duplex

- TCP Segments include sequence number of data being carried and expected sequence as ACK



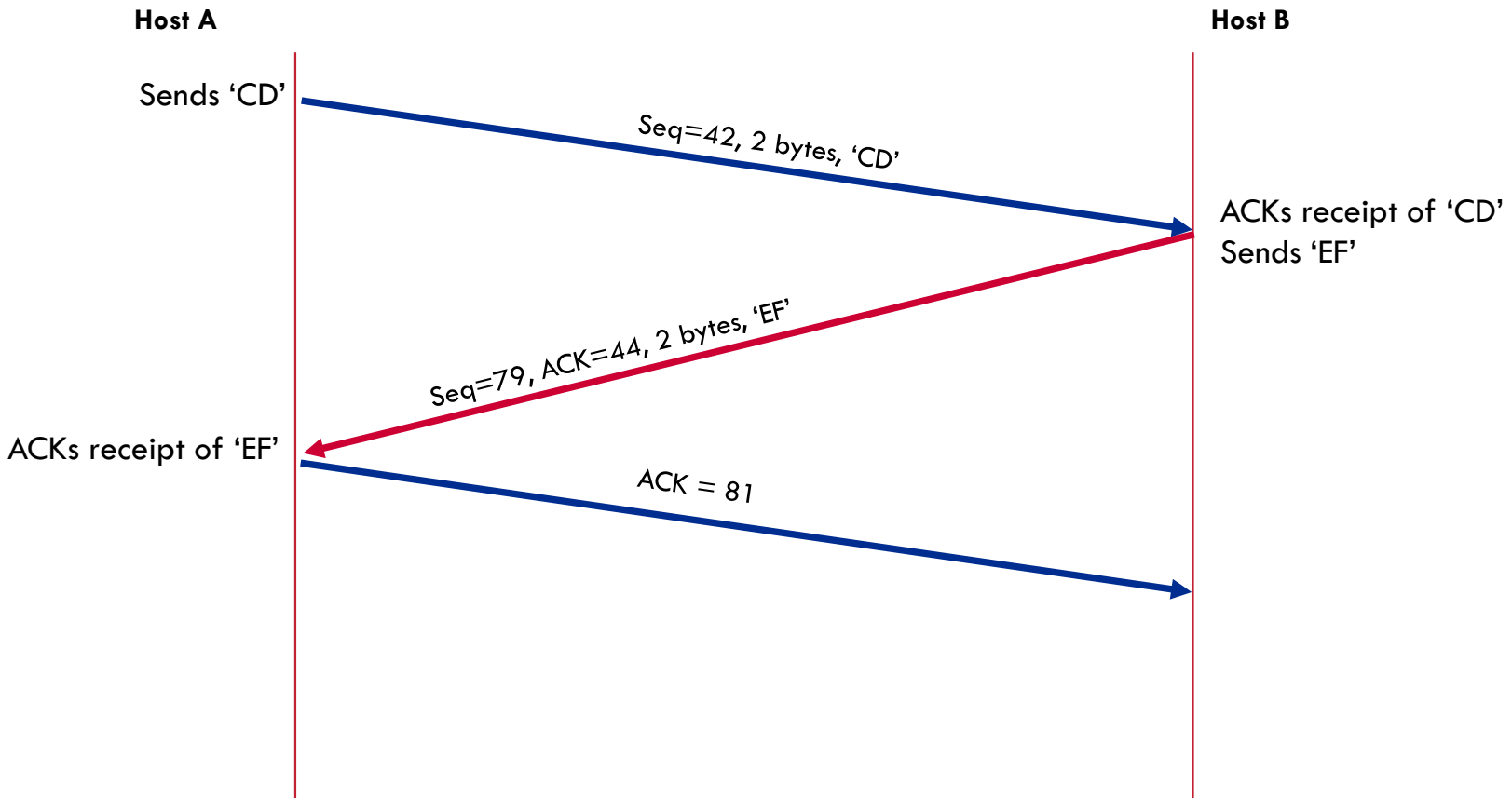
TCP Full Duplex

- TCP Segments include sequence number of data being carried and expected sequence as ACK



TCP Full Duplex

- TCP Segments include sequence number of data being carried and expected sequence as ACK



TCP Timeout

- Timeout is calculated dynamically
  - RFC 2988 & RFC 6298
  - Needs to be twice as long as RTT
  - If too short:
    - ?

TCP Timeout

- Timeout is calculated dynamically
  - RFC 2988 & RFC 6298
  - Needs to be twice as long as RTT
  - If too short:
    - Premature timeout
    - Unnecessary retransmission
  - If too long:
    - ?

TCP Timeout

- Timeout is calculated dynamically
  - RFC 2988 & RFC 6298
  - Needs to be twice as long as RTT
  - If too short:
    - Premature timeout
    - Unnecessary retransmission
  - If too long:
    - Slow reaction to loss

TCP Timeout Factors

- Because timeout is based on RTT dynamically, where do we get RTT?

TCP Timeout Factors

- Because timeout is based on RTT dynamically, where do we get RTT?
  - SampleRTT = time from segment transmission to ACK

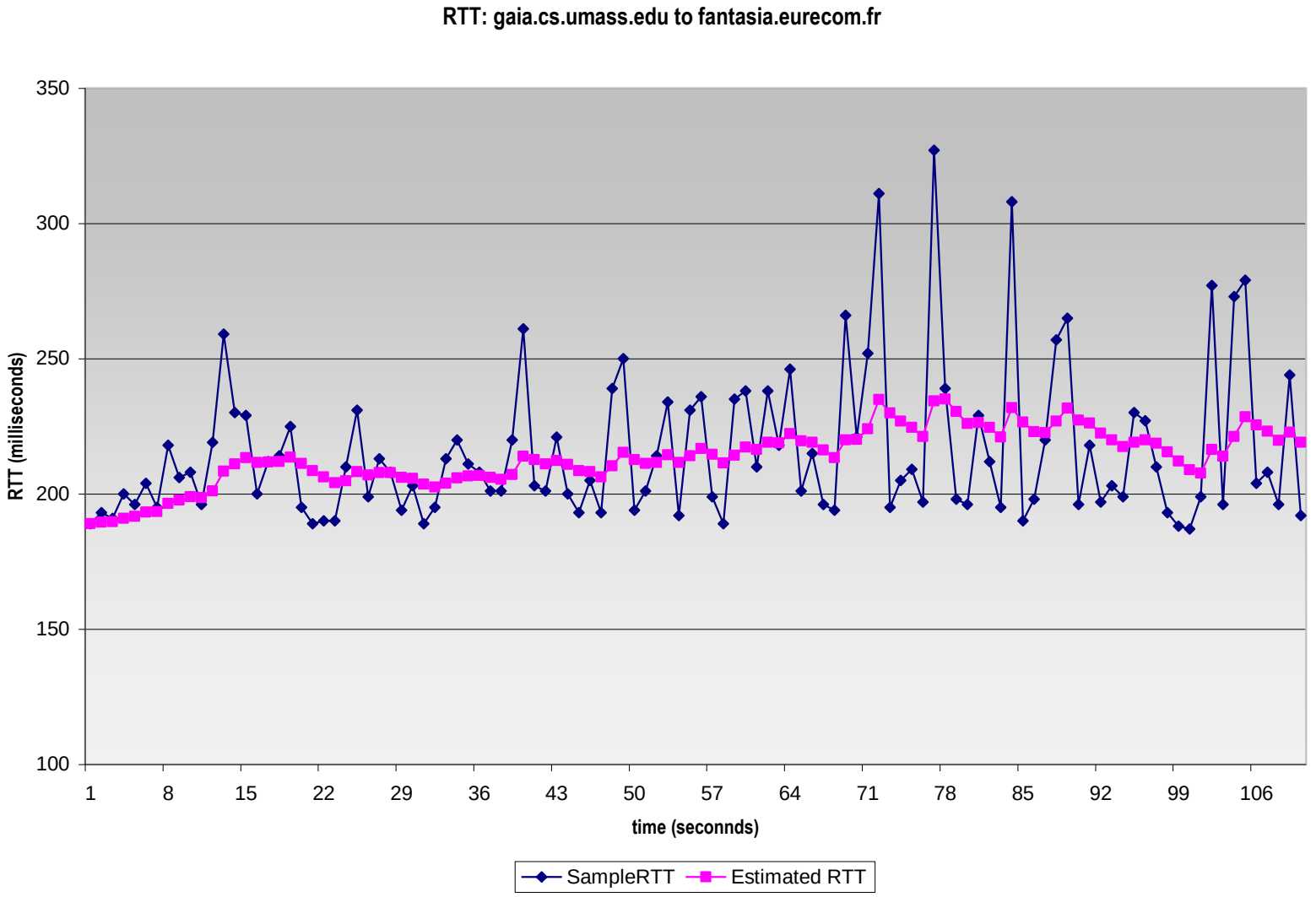
TCP Timeout Factors

- Because timeout is based on RTT dynamically, where do we get RTT?
  - SampleRTT = time from segment transmission to ACK
- Because individual RTTs may vary, this is smoothed in to EstimatedRTT:
  EstimatedRTT = (1-α)*EstimatedRTT + α *SampleRTT

TCP Timeout Factors

- Because timeout is based on RTT dynamically, where do we get RTT?
  - SampleRTT = time from segment transmission to ACK
- Because individual RTTs may vary, this is smoothed in to EstimatedRTT:
  EstimatedRTT = (1-α)*EstimatedRTT + α *SampleRTT
  - Weighted average, iteratively updated
  - Common α value: 0.125

RTT Estimation



More TCP Timeout Factors

- Estimation is not good enough if connection is inconsistent

More TCP Timeout Factors

- Estimation is not good enough if connection is inconsistent
- Deviation calculation can be helpful

$$DevRTT = (1-\beta)*DevRTT + \beta * |SampleRTT - EstimatedRTT|$$

More TCP Timeout Factors

- Estimation is not good enough if connection is inconsistent
- Deviation calculation can be helpful

  $DevRTT = (1 - \beta) * DevRTT + \beta * |SampleRTT - EstimatedRTT|$

  - Often β is 0.25
  - Also weighted average
  - Calculation of how 'off' SampleRTT is from EstimateRTT
    - PID loops anyone? Kind of like the Derivative value.

Timeout Interval Calculation

- Given EstimatedRTT and DevRTT, what should the timeout be?

Timeout Interval Calculation

- Given EstimatedRTT and DevRTT, what should the timeout be?
  - It has to be greater than EstimatedRTT
    - Or unnecessary retransmissions are likely
  - Can't be too big
    - Laggy response times are likely

Timeout Interval Calculation

- Given EstimatedRTT and DevRTT, what should the timeout be?
  - It has to be greater than EstimatedRTT
    - Or unnecessary retransmissions are likely
  - Can't be too big
    - Laggy response times are likely
  - Can be small if link is stable
  - Probably should be larger is link is not stable

Timeout Interval Calculation

- Given EstimatedRTT and DevRTT, what should the timeout be?
  - It has to be greater than EstimatedRTT
    - Or unnecessary retransmissions are likely
  - Can't be too big
    - Laggy response times are likely
  - Can be small if link is stable
  - Probably should be larger is link is not stable

TimeoutInterval =EstimatedRTT + 4*DevRTT

Timeout in Practice

- TimeoutInterval =EstimatedRTT + 4*DevRTT
- Initially TimeoutInterval is set to 1 second

Timeout in Practice

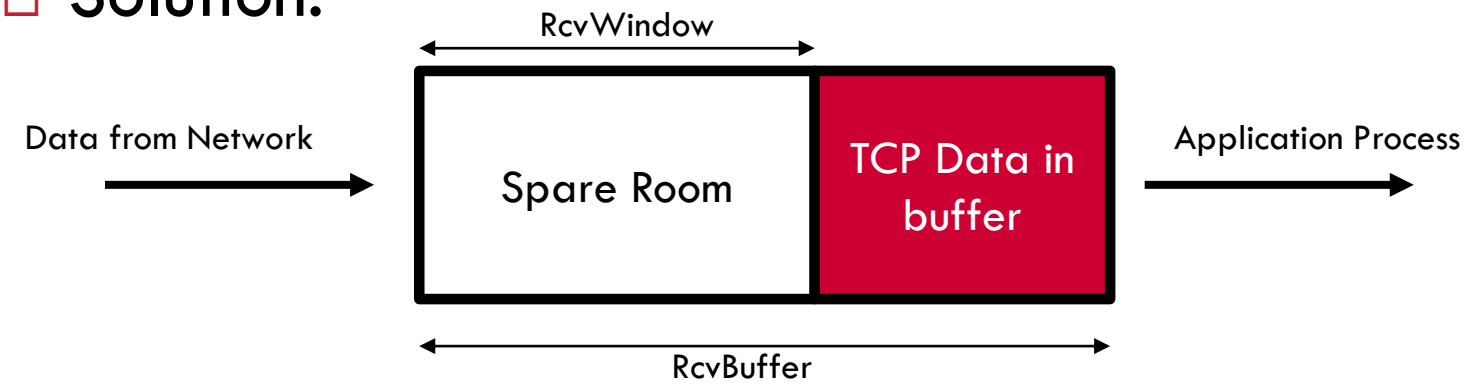
- TimeoutInterval =EstimatedRTT + 4*DevRTT
- Initially TimeoutInterval is set to 1 second
- If timeout occurs, TimeoutInterval is doubled
  - When segment is ACK'ed, TimeoutInterval goes back to be calculated normally

Flow Control

- Issue: Sender can push too much data through network to overload receiver buffer
- Solution?

Flow Control

- Issue: Sender can push too much data through network to overload receiver buffer
- Solution:



- Receiver sends back RcvWindow size in segment header of ACKs
- Sender adjusts window to match RcvWindow

TCP: Overview

- Reliable, in-order byte stream
- Pipelined protocol
- Sender and Receiver data is buffered
- Full duplex data
  - Data flow in both directions, not just commands
- Connection Oriented
  - Three way handshake to initialize both sides
- Flow controlled – avoid congestion

Outline

- TCP Details
- Exam 2 Review

Exam Review

- Possible question on Peer-to-Peer
- Exam 2 is all about Transport Layer
- Some concepts still carry over from Exam 1
  - This is all cumulative

Exam 2

- Peer-to-Peer
  - BitTorrent
- Transport Layer
  - Overview – Service
  - De/Multiplexing
  - Ports
  - Reliable Data Transfer (rdt model)
    - Diagrams! Lots of them!
  - Pipelining
    - Go-Back-N
    - Selective Repeat
  - Checksum
  - UDP
    - Segment Format
  - TCP
    - Segment Format
    - Sequence Numbers
    - ACK and error control
    - Timeout

Looking Forward

- Next Tuesday: Exam
- Next Thursday: Network Layer
- LM5: Tuesday Nov. 5th