Computer Networks

COMP 3670

Transport Layer

Sherif Saad

School of Computer Science | University of Windsor

June 14, 2021

Table of Contents

Pipelined Reliable Data Transfer Protocols

- RDF v3.0 is a reliable data transfer protocol. However, it is an inefficient protocol. This is because it uses a stop-and-wait pattern.
- RDF 3.0 is sometimes known as the alternating-bit protocol. Because packet sequence numbers alternate between 0 and 1.
- A better solution is to use a pipelining pattern instead of the stop-and-wait.
- With pipelining, the sender is allowed to send multiple packets without waiting for acknowledgments.

What is needed to enable pipelining?

Pipelined RDT

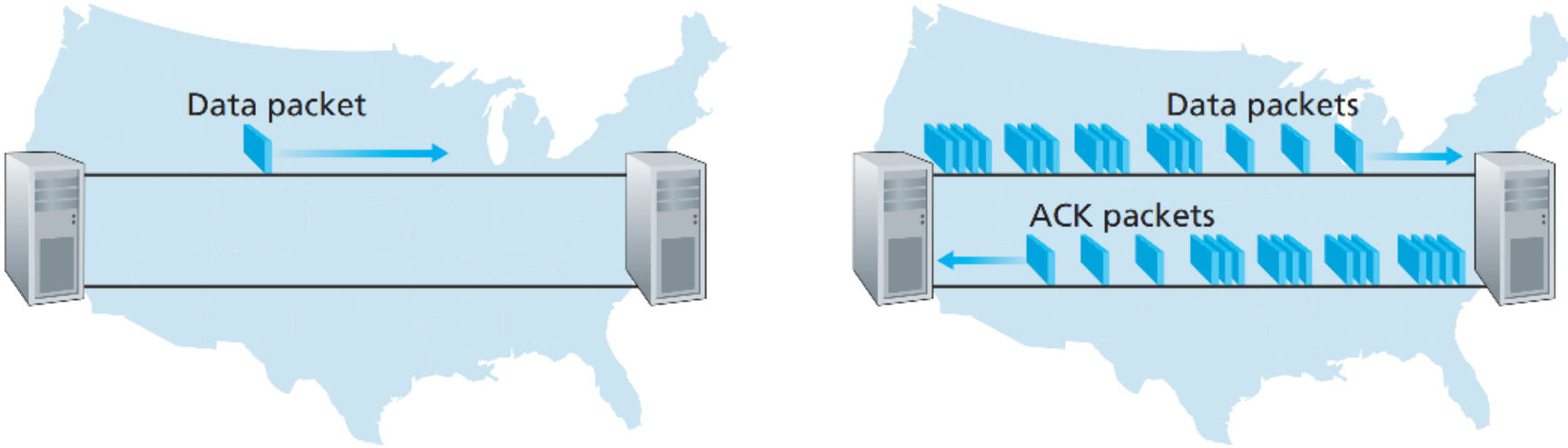


Figure: Stop-and-wait versus pipelined protocol

Enabling Pipelining

What is needed to enable pipelining?

- The range of sequence numbers must be increased.
- The sender and receiver sides of the protocols may have to buffer more than one packet.

What is the range of sequence number and what is the buffering requirements?

What is the primary impact of enabling pipelining on reliability of data transfer?

Stop-and-Wait Protocol

- A stop-and-wait protocol approach with timer, acknowledgment, error detection, sequence number, and buffering enable building a reliable data transfer protocol.
- Using stop-and-wait is inefficient.
- By using a stop-and-wait, we can not fully utilize the available transmission rate.
- While the link is available to push data into it, we can not push any data until we receive the feedback from the host on the other end of the logical path.

Stop-and-wait operation

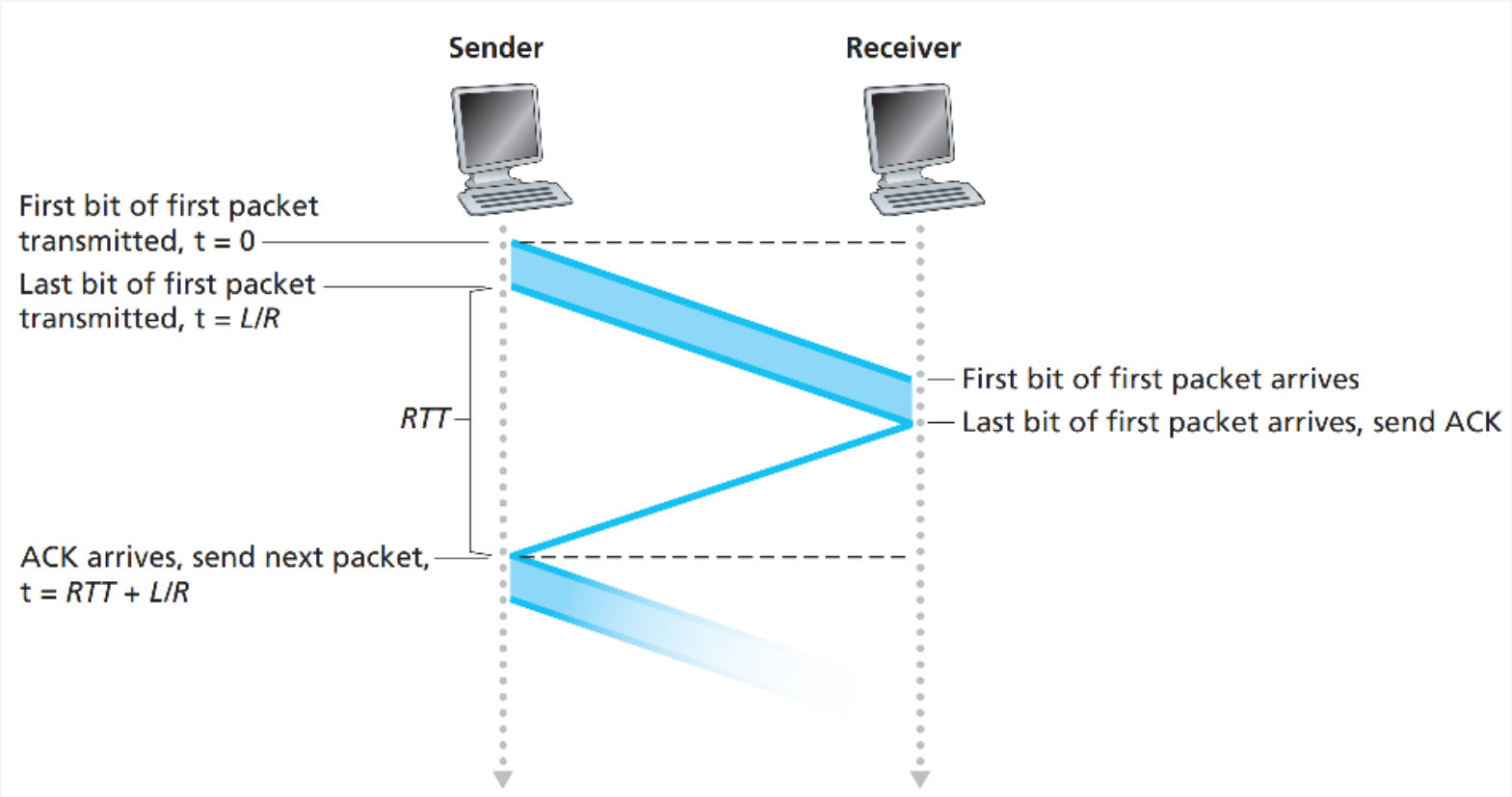


Figure: Stop-and-wait operation

Pipelined operation

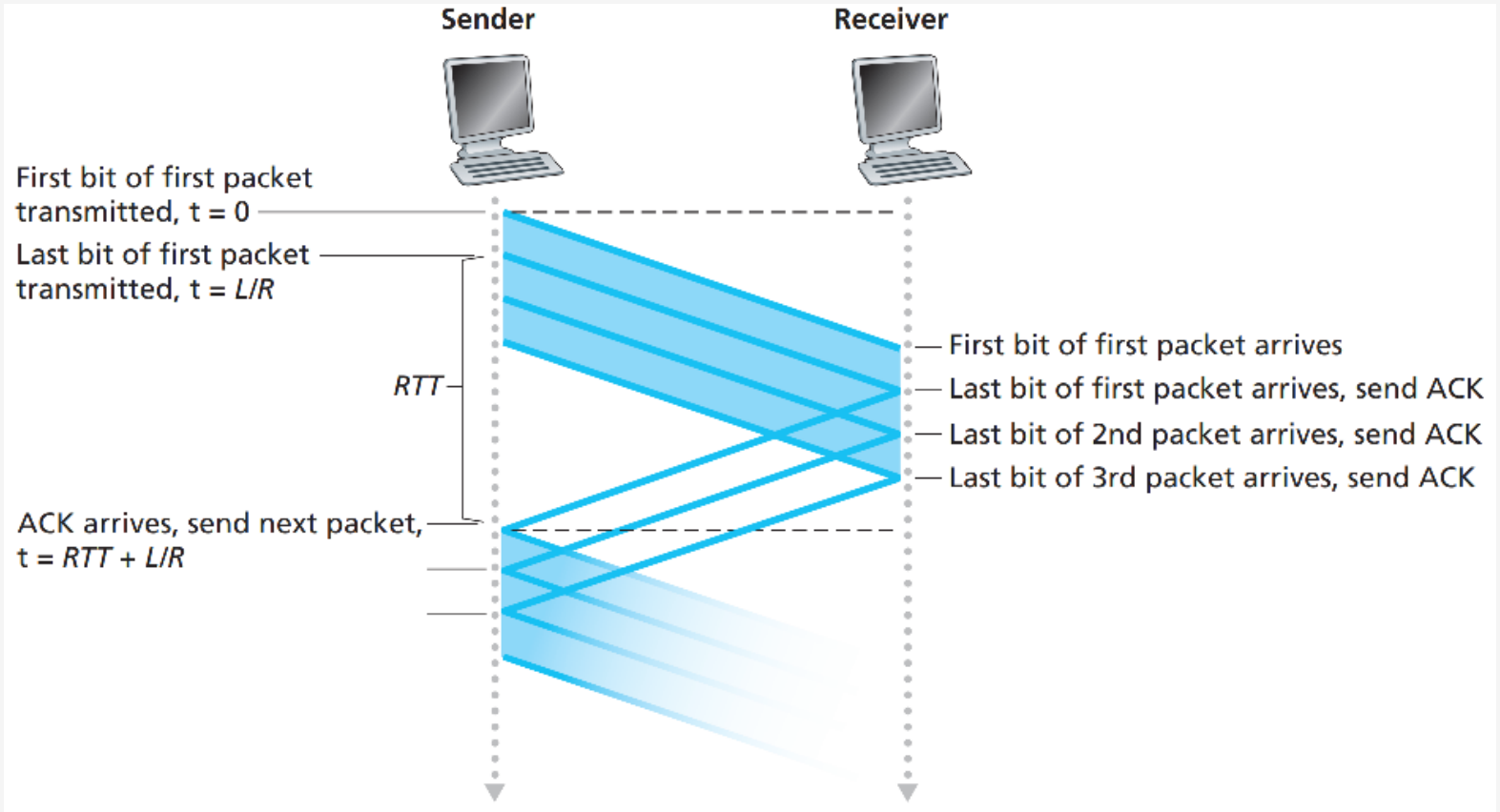


Figure: Pipelined operation

Stop and Wait Example I

- Assume we are sending data from host ***A*** to host ***B***.
- The propagation delay between host ***A*** and host ***B*** is 3 ms
- The access link of host A has a transmission rate of 1 Gbps (10^9 bits per second)
- Host ***A*** is sending packets of equal length, where each packet is of length ***L*** and ***L*** $= 1000$ bytes (8000 bits)
- The transmission delay of Host ***A*** is $\frac{8000}{10^9} = 8$ microseconds or 0.008 m

Stop and Wait Example II

- Assume the size of the ***ACK*** packet from ***B*** to ***A*** is very small, the time it takes is neglectable.
- The time to get the feedback at ***A*** $= RTT + \frac{L}{R} = 30.008$ ms

What is the sender utilization time?

- The utilization time of the sender
  $sender_{utiliz} = \frac{\frac{L}{R}}{RTT + \frac{L}{R}} = \frac{0.008}{30.008} = 0.000027$
- This means for sending 8000 bits we need 30.008 ms, which mean the access link for ***A*** operate at a transmission rate of $267 kbps$ and not $1 Gbps$.

Pipelining instead of Stop-and-Wait

- The solution to solve the poor utilization is to use pipelining instead of stop-and-wait.
- In pipelining, the sender can send packets without waiting for acknowledgment from the receiver.
- This could result in packets arrive out of order at the receiver side.
- Now, the range of sequence number needs to increase, can not use any more alternating-bit protocol.
- The sender and the receiver will most likely need to buffer more than one packet.

Implementing Pipelining

Approaches

The size of the **sequence number range** and the **buffer size** on the sender and receiver sides are determined based on the approach we will use to implement the pipelining.
Generally, there are two approaches to implement pipelining

- Go-Back N (GBN)
- Selective Repeat (SR)

Go-Back-N

- The sender can send up to N number of packets (if any available) without waiting for acknowledgment from the receiver.
- This approach can only tolerate up to N unacknowledged packets in the pipeline
- At any point in time there are U unacknowledged packets in the pipeline where $U <= N$, whenever a packet is acknowledged $U = U - M$ *where* $1 <= M <= N$

How Go-Back-N Works?

- Let **base** be the sequence number of the oldest unacknowledged packet.
- Let **nextseqnum** be the smallest unused sequence number out of the range of available sequence numbers.
- Let [0, base-1] a set of sequence numbers corresponding to acknowledged packets
- Let [base, nextseqnum-1] a set of sequence numbers corresponding to unacknowledged packets that have been sent
- Let [nextseqnum, base+N-1] a set of sequence numbers that could be used to send new packets before reaching the stop point (threshold)

Go-Back N

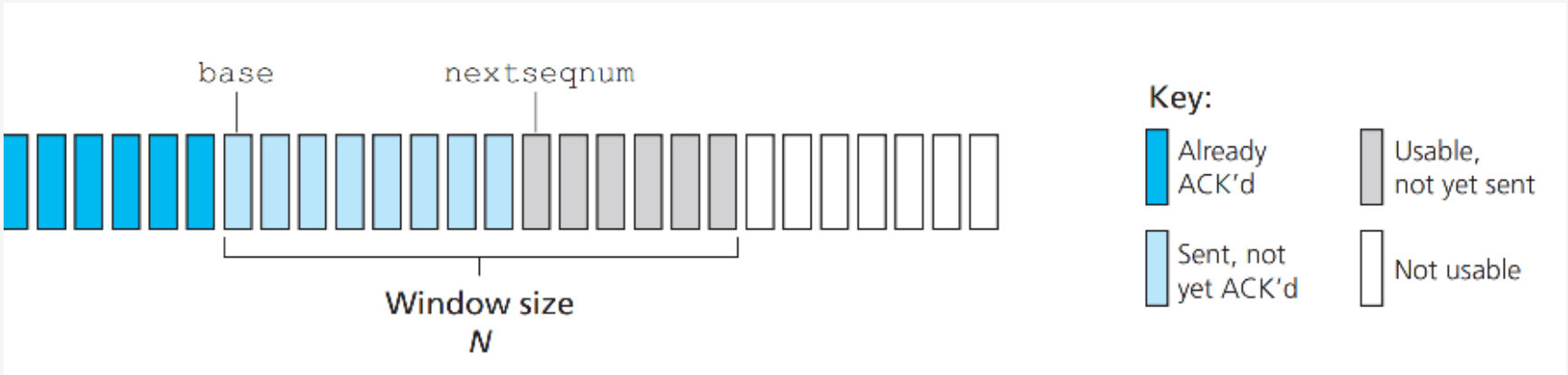


Figure: Sender's view of sequence numbers in Go-Back-N

How Go-Back-N Works?

- Every time an acknowledgment received by the sender, the different sets change (slide over the range of sequence numbers), by size N
- N is often referred to as the window size, and the GBN approach is known to be a sliding window protocol.
- If we have k bits to represent the sequence number than the range of sequence number is $[0, 2^k - 1]$
- In GBN the acknowledgment of a packet with sequence number K, indicating that the receiver has received all the packets between base and K.

How Go-Back-N Works?

- The acknowledgment in GBN is a cumulative acknowledgment.
- A protocol using GBN approach will use a timer; when the timer expires, the protocol will send all the packets that have not been acknowledged in the pipeline.

Do we need to buffer at the sender side and the receiver side?

GBN Sender FSM

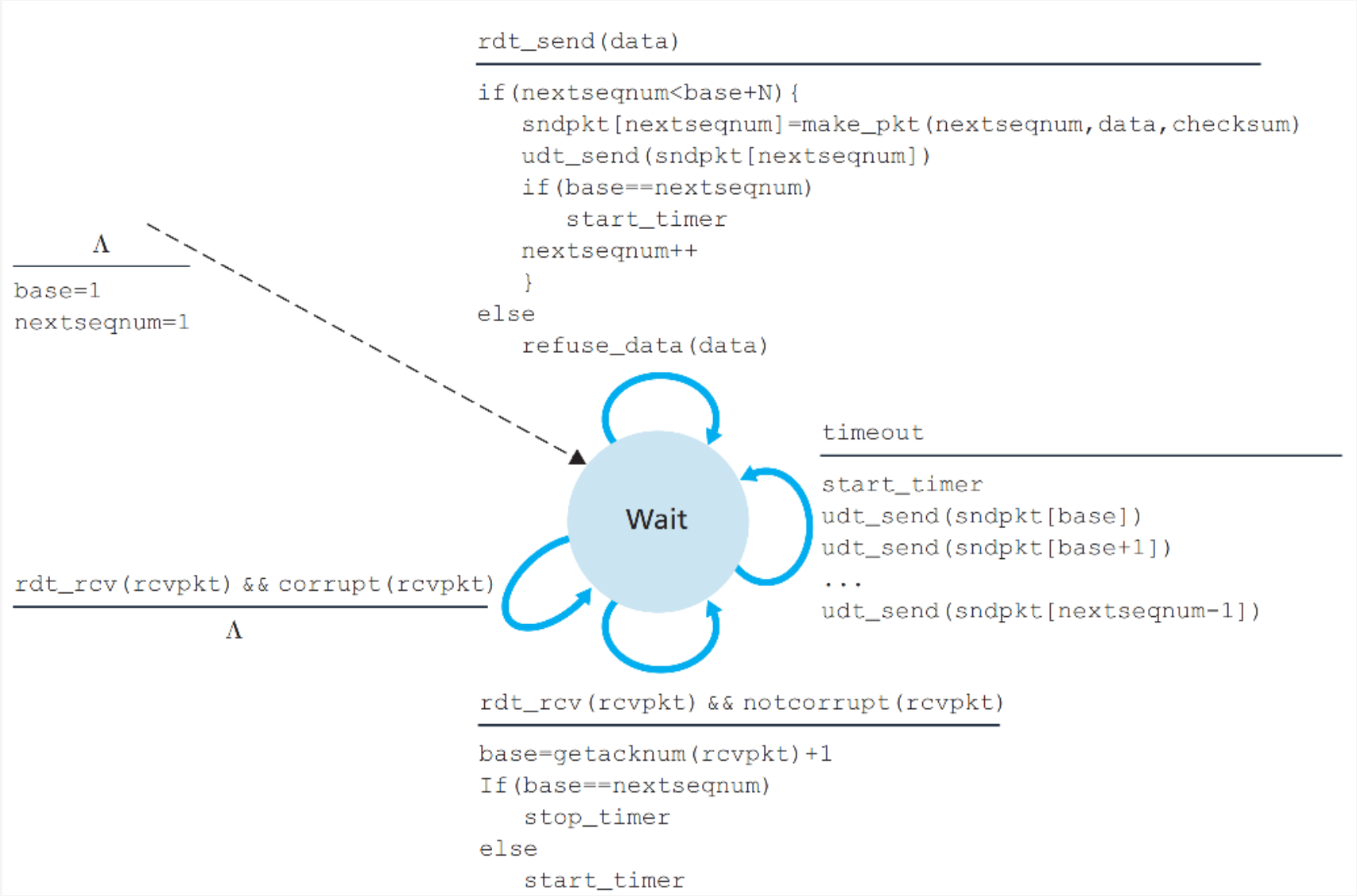


Figure: FSM description of the GBN sender

GBN Receiver FSM

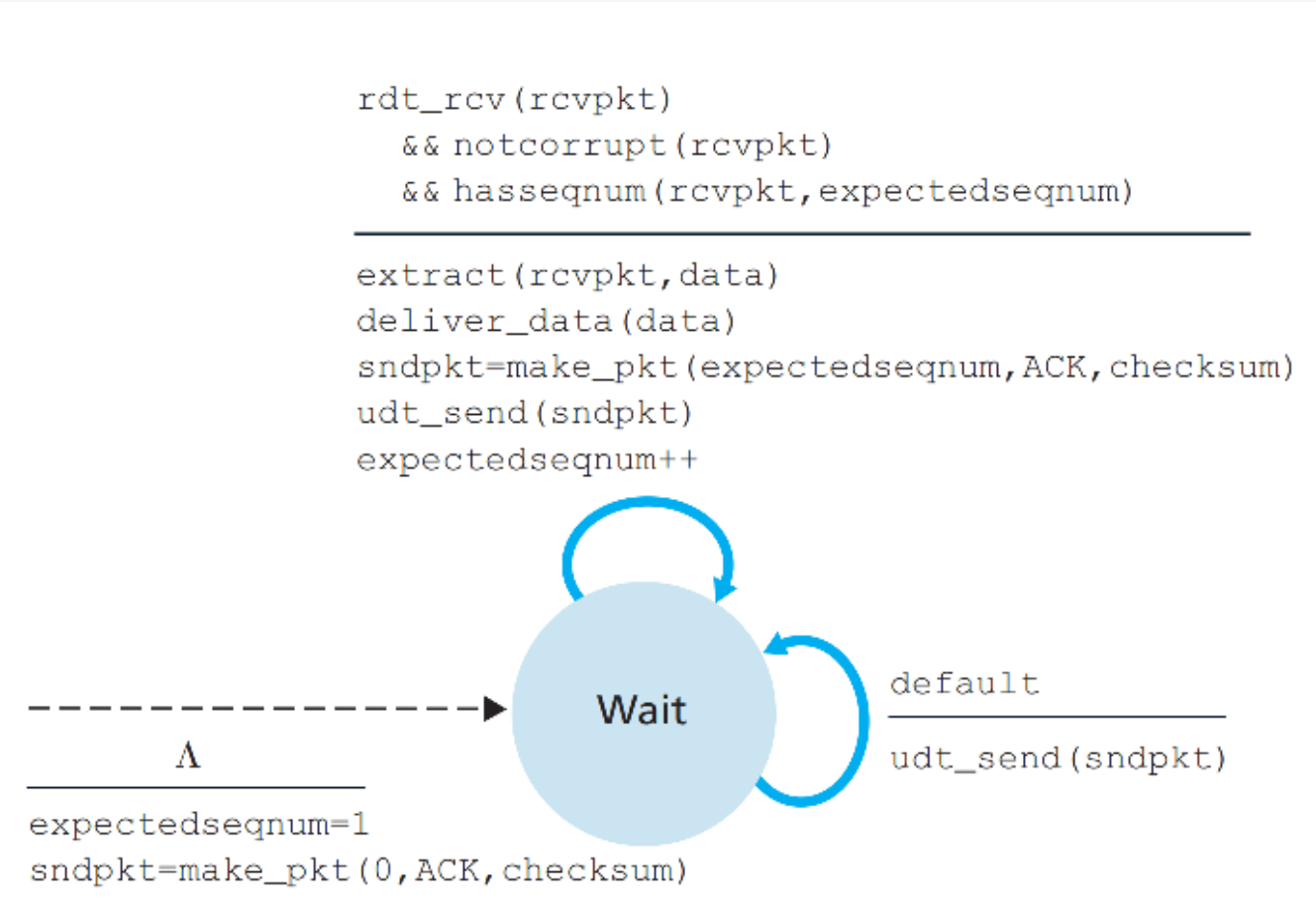


Figure: FSM description of the GBN receiver

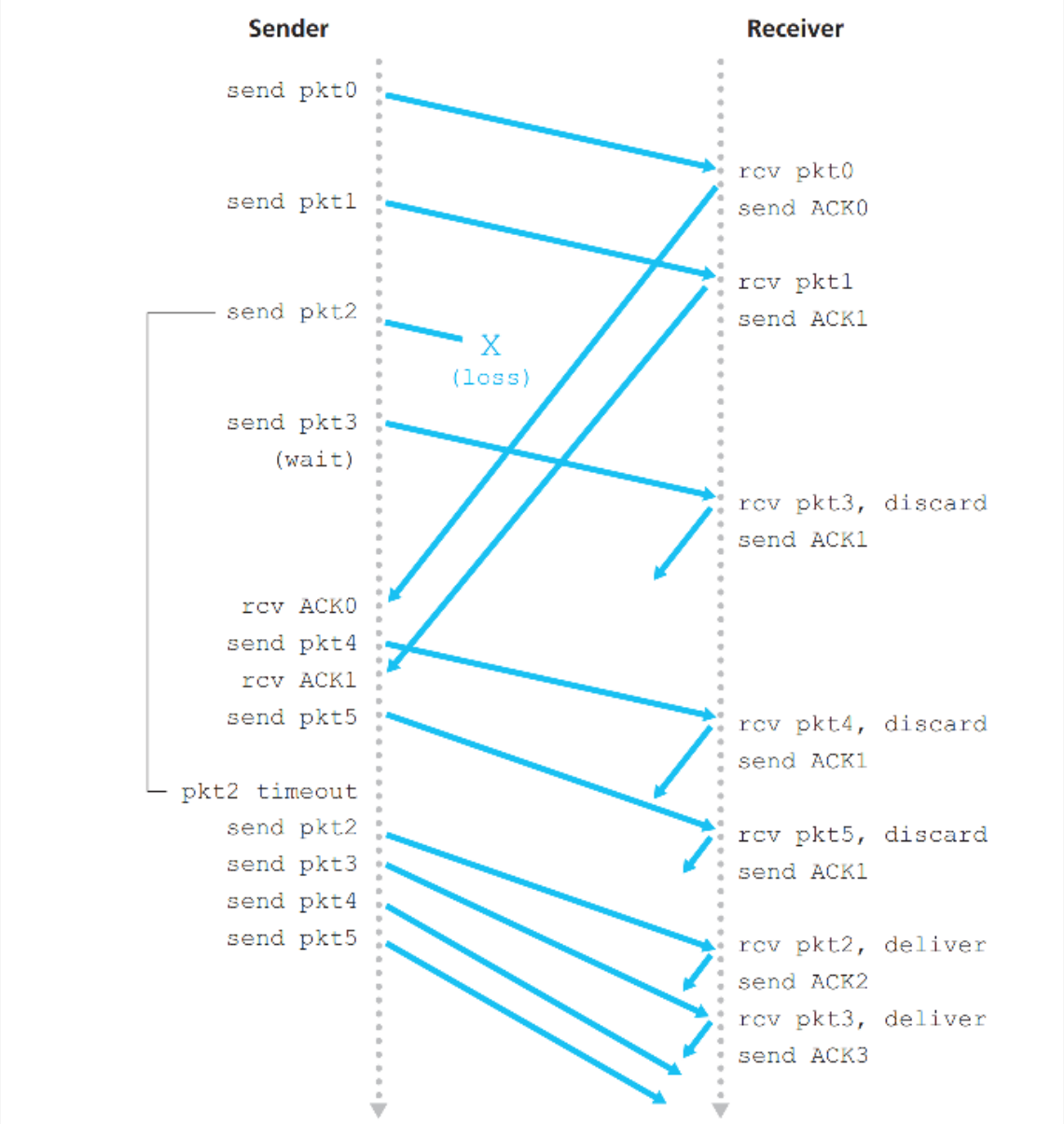


Figure: Go-Back-N in operation

Selective Repeat (SR)

- When packets arrive out of order GBN protocols will ignore them, resulting in the sender retransmitting those packets again.
- One approach that enables pipelining without the need for unnecessary retransmission is a selective repeat (SR).
- The sender can acknowledge individual packets as they arrive regardless their arriving order (in-order or out-of-order)
- In this approach, the receiver will have a buffer of size N; even if the packet arrives out of order, they are stored in the buffer at the receiver's side.

Selective-repeat (SR) Operations

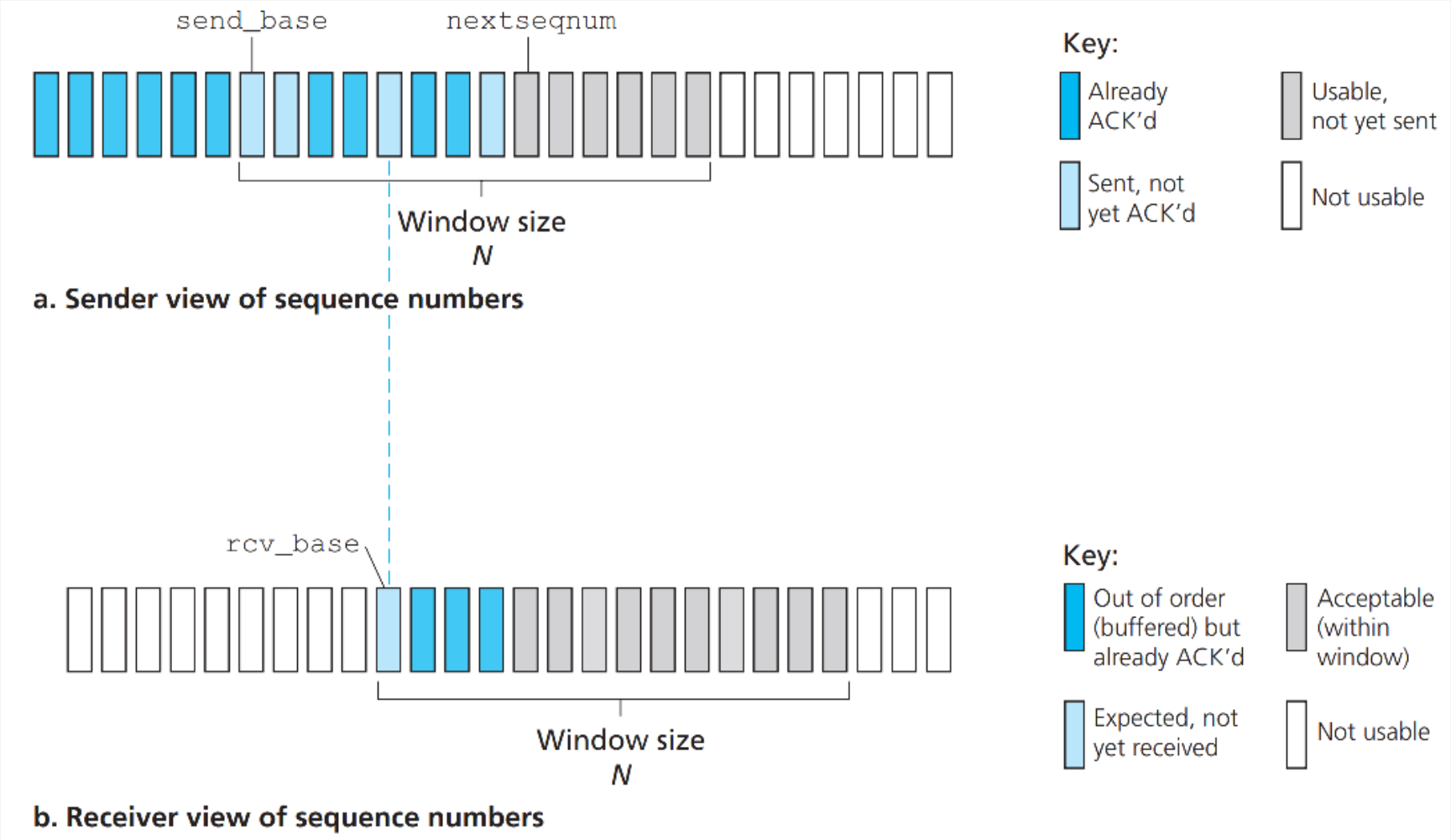


Figure: Selective-repeat (SR) sender and receiver views of sequence-number space

How Selective Repeat Works? I

- The sender uses a timer to detect potential packet loss. However, each packet has a timer.
- When the timer of a given packet expires, the sender retransmits this packet and resets the timer for that packet.
- When an acknowledgment received; it only used as an indication of receiving a specific packet (no cumulative acknowledgment)
- On the receiver side, when a packet arrives within the allowed window size, the packet is stored in a local buffer, and an acknowledgment is sent to the sender.

How Selective Repeat Works? II

- As soon as the packet with the sequence number corresponding to the **rcv_base** sequence number arrives at the receiver side, this packet and all consecutively numbered packets in the receiver buffer are delivered to the application layer.
- The **rcv_base** then moves to the smallest unreceived sequence number within the window size.

Questions

What happens if we receive a packet with a sequence number between [rcv_base-N, rcv_base-1]?

What happens if we receive a packet with a sequence number that is not in the range of [rcv_base, rcv_base+N-1] or [rcv_base-N, rcv_base-1]?

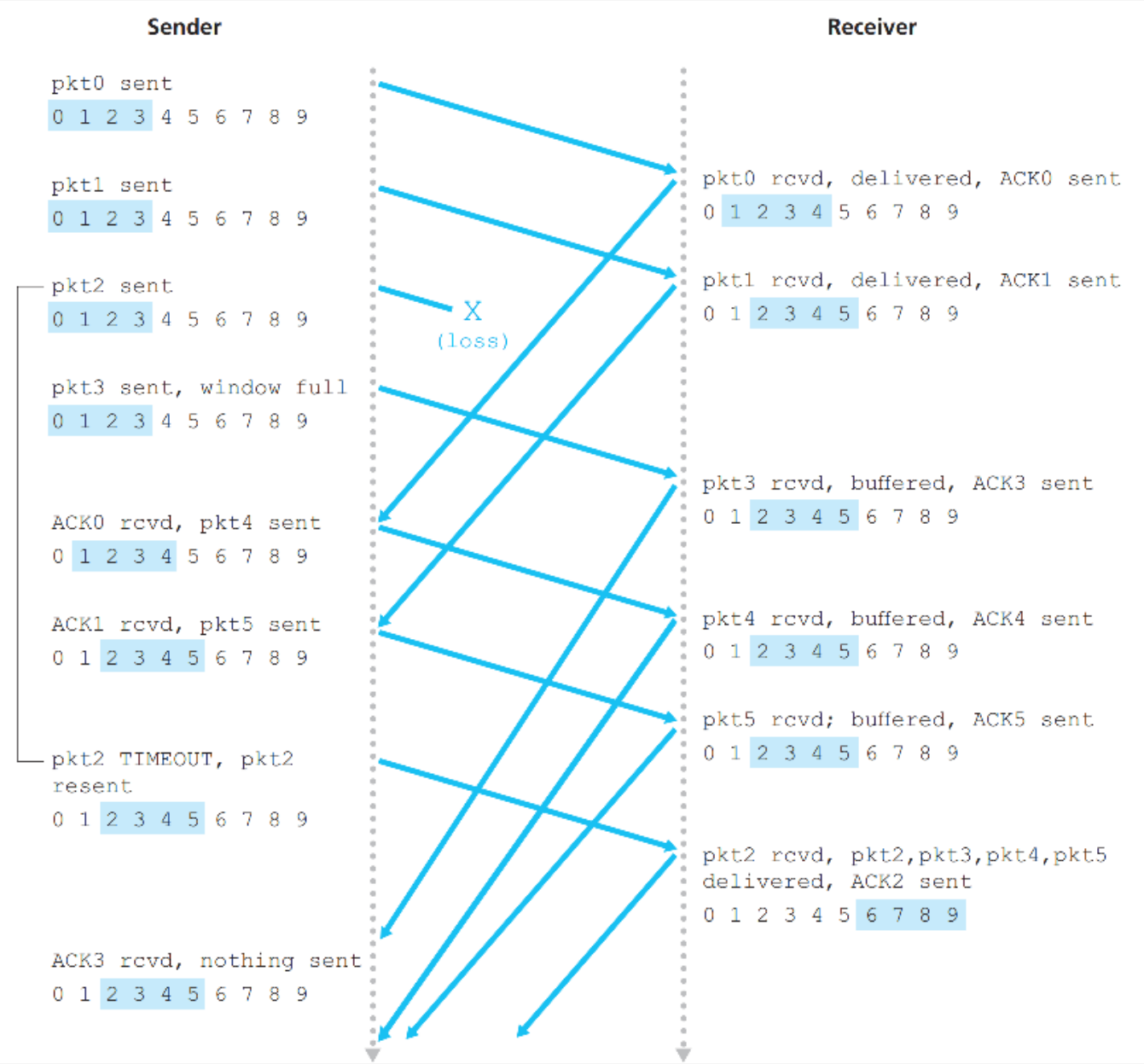


Figure: SR in Operation

Selecting Window Size

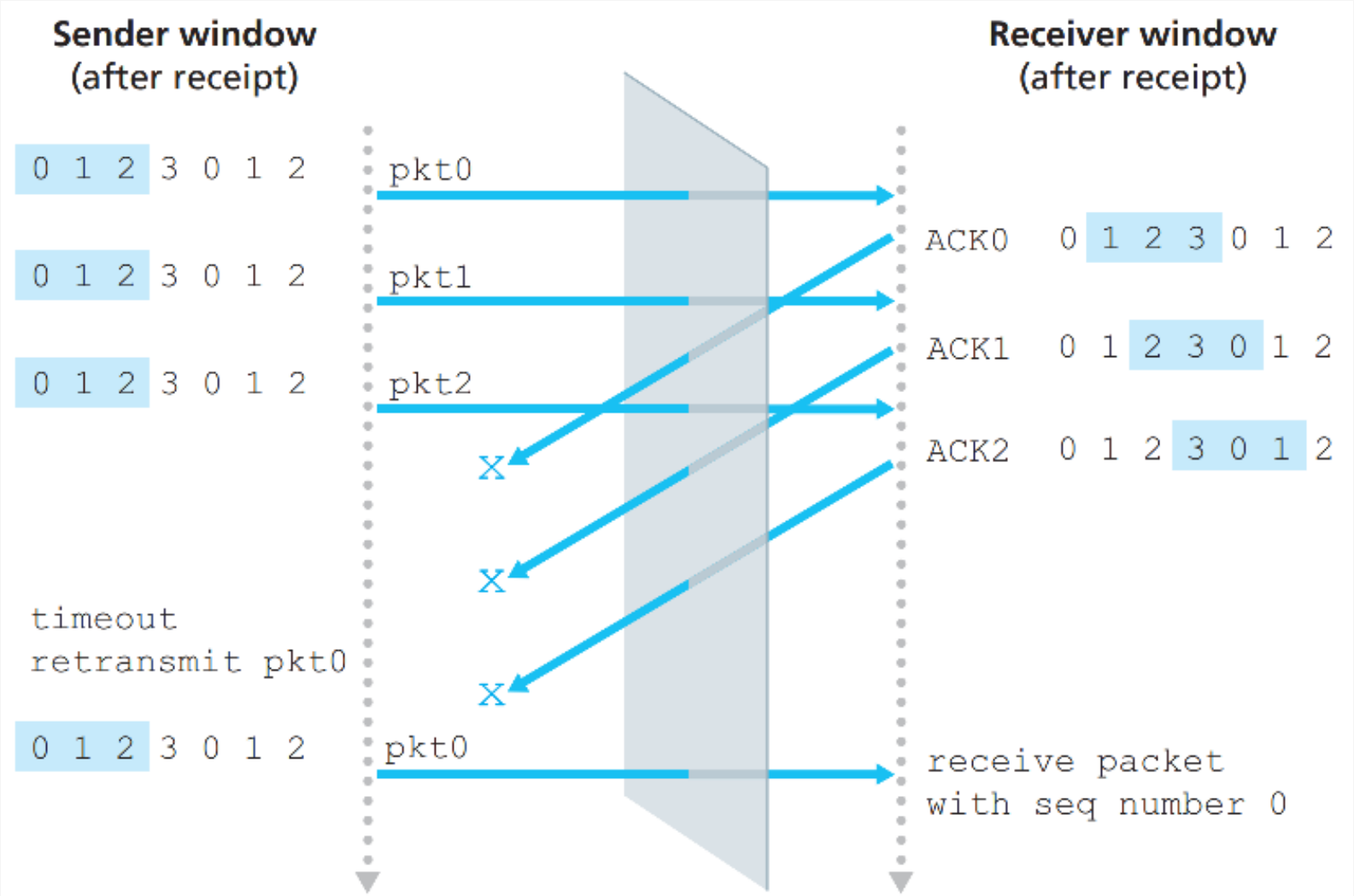


Figure: SR receiver dilemma with too-large windows: A new packet or a retransmission?

Selecting Window Size

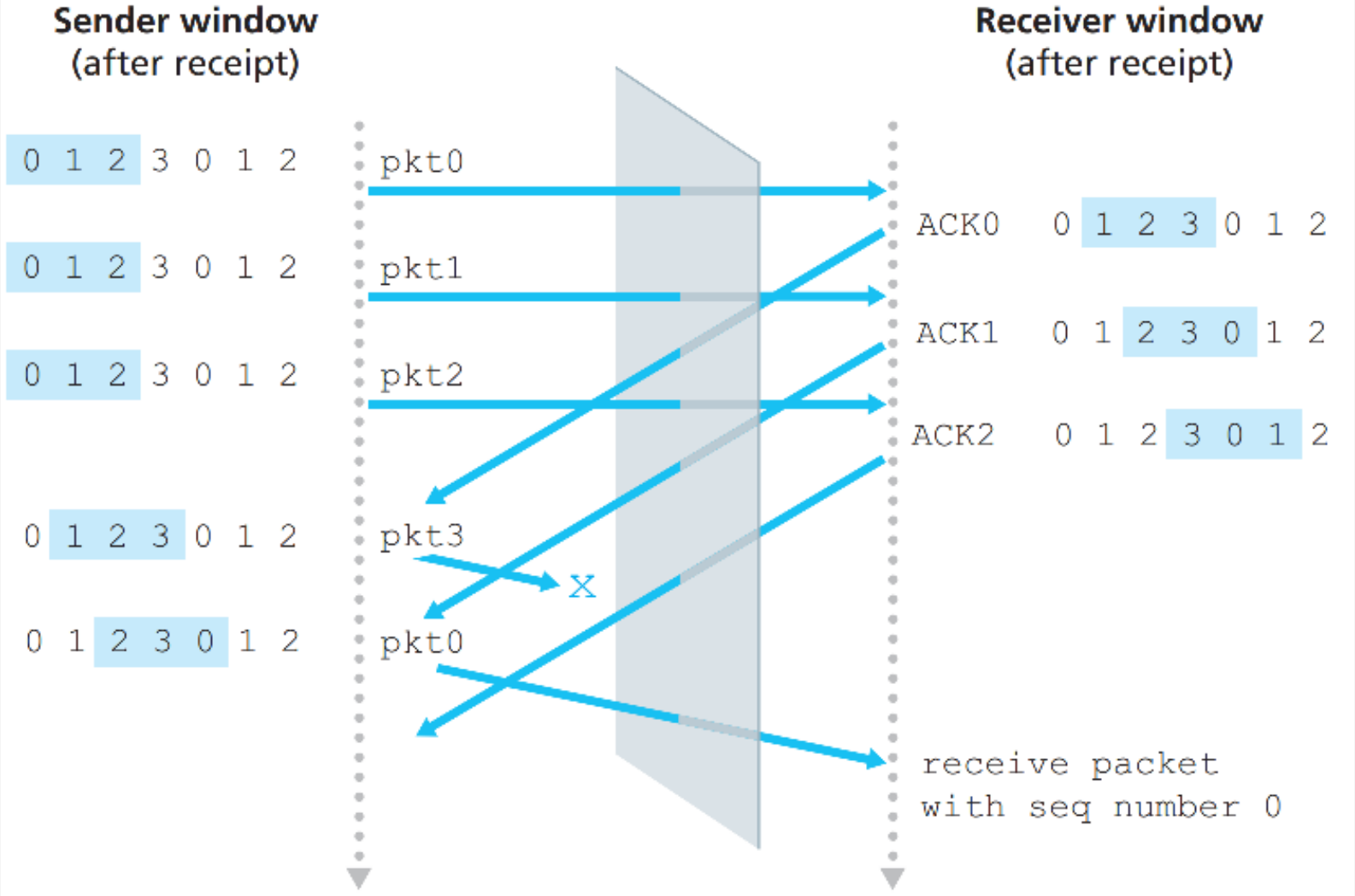


Figure: SR receiver dilemma with too-large windows: A new packet or a retransmission?

Mechanisms for Designing Reliable Data Transfer Protocol

1. Checksum
2. Timer
3. Sequence Number
4. Acknowledgment
5. Pipelining

Connection-Oriented Transport Protocol: TCP

- TCP is a connection-oriented protocol
- TCP is a reliable protocol
- TCP is an end-to-end or point-to-point communication protocol
- TCP establishes a logical connection between processes on two different hosts.
- TCP is a full-duplex protocol.
- TCP does not support one-to-many communication or multicasting connections.
- The process of establishing the TCP connection (connection-oriented) is known as a three-way handshake.

TCP Send and Receive Buffers

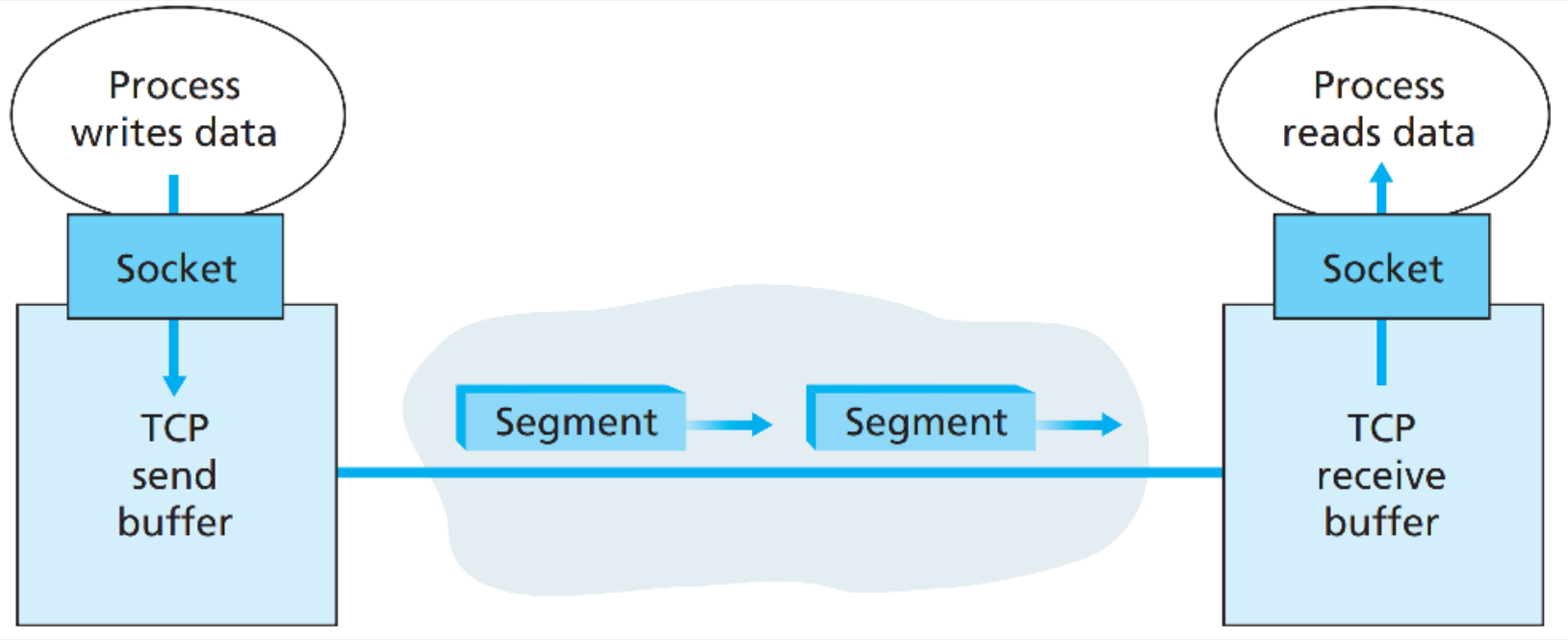


Figure: TCP send and receive buffers

TCP Send and Receive Buffers

- The outgoing application data are stored in the send buffer.
- The TCP will grab chunks of data from the send buffer and pass it to the network layer for transmission. (How often?, How much?)
- Each TCP segment is limited in size; this limit is referred to as the maximum segment size (MSS)
- The MSS is defined based on the maximum transmission unit (MTU)
- The MTU measures the maximum data frame we could send over link-layer (network link)
- The TCP segment and header, plus the IP header, should be less than or equal to the MTU (why?)

TCP Segment Structure

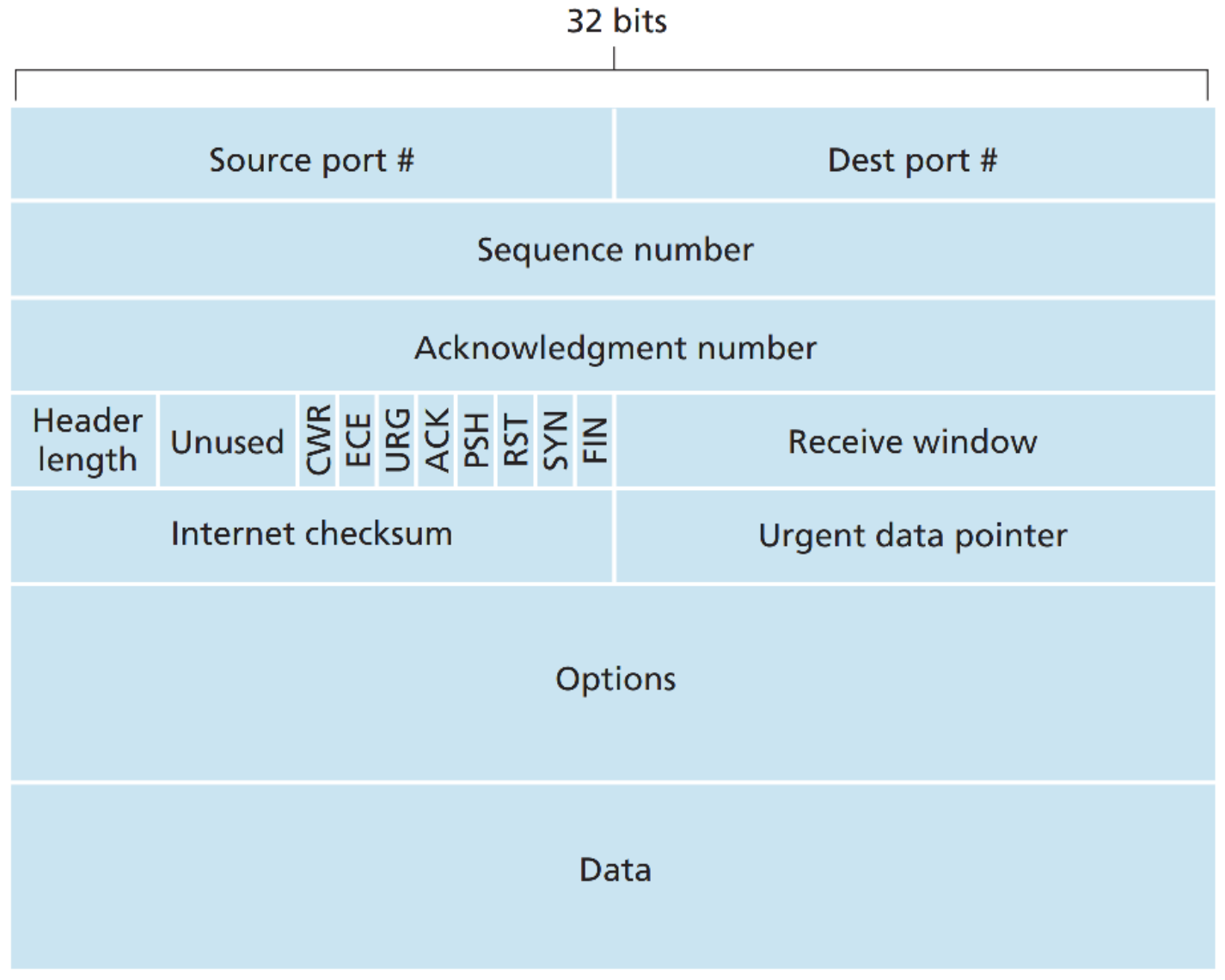


Figure: TCP segment structure details

Sequence Number and Acknowledgment

- TCP views data as an unstructured but ordered stream of bytes.
- The sequence number for a segment is the byte-stream number of the first byte in the segment.
- The acknowledgment number sent by the receiver to the sender is the sequence number of the next bytethe receiver expects from the sender.
- TCP uses a cumulative acknowledgment approach.
- If the receiver has data to send back to the sender, it is possible to send it with the acknowledgment packet (piggybacked ack)

How Sequence Numbers Work?

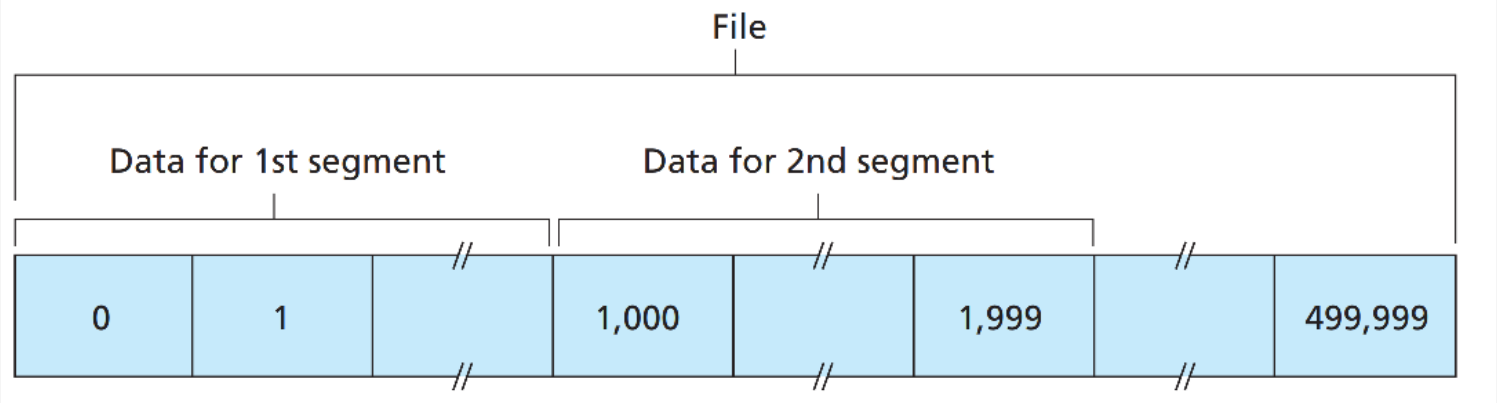


Figure: Setting Sequence Number based on Data

Sequence and Acknowledgment Numbers: Telnet Example

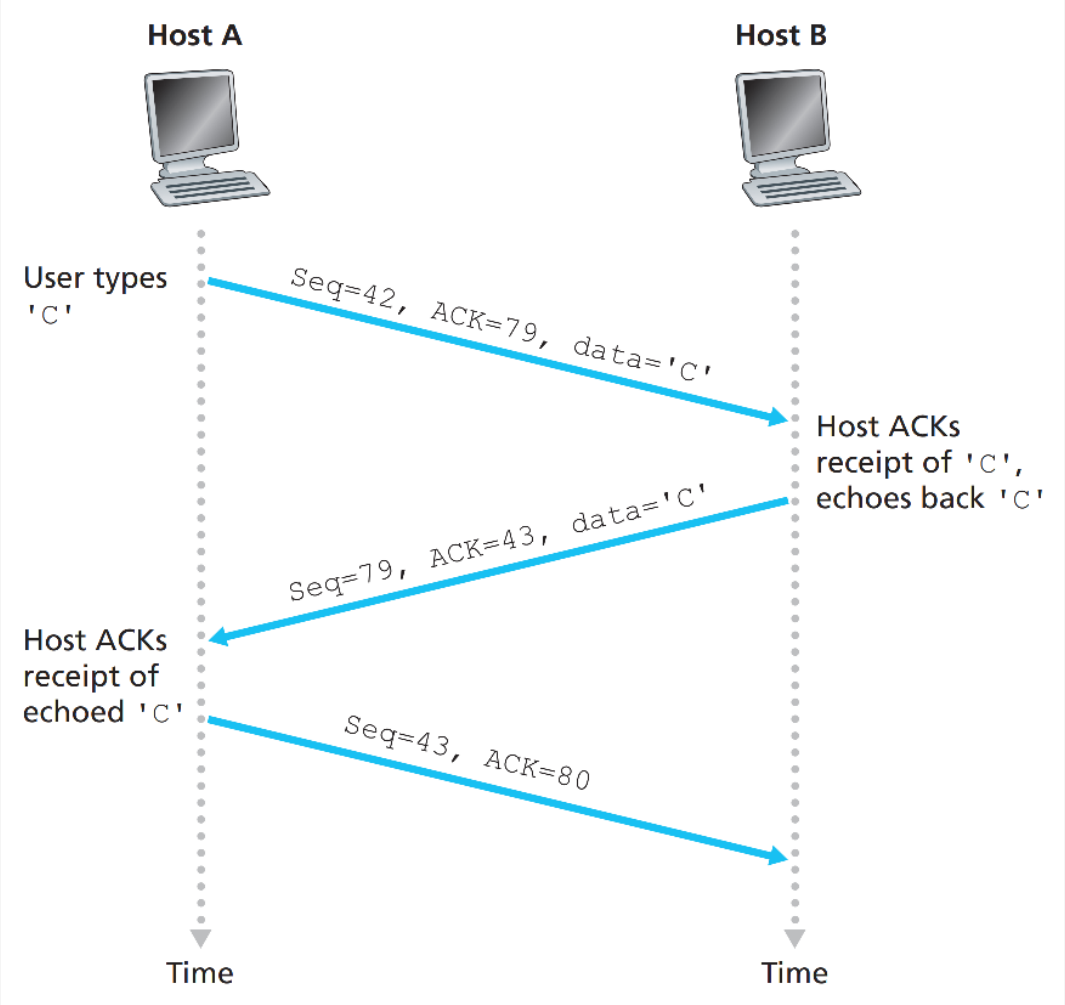


Figure: Sequence and acknowledgment numbers for a simple Telnet application over TCP

TCP Timer Estimation

- Selecting the value for the timer is an interesting and challenging task.
- A premature timeout will result in unnecessary retransmission.

When is it possible to have a premature timeout?

- The connection round-trip time (RTT) influences the selection of the timeout.
- The connection **RTT** is not fixed; it will **fluctuate** from segment to segment and based on the network traffic intensity.

Estimating the Round-Trip Time I

- The TCP estimates the RRT using a sampling approach.
- From time to time, the TCP will measure the time between passing a segment to the network layer and receiving the acknowledgment of that segment.
- The TCP will only measure the RTT for a new outgoing segment and not for a retransmitted one (why?)
- The TCP uses the following formula to estimate the RTT

$$EstimatedRTT = (1 - \alpha) \times EstimatedRTT + \alpha \times SampleRTT^{1}$$

Estimating the Round-Trip Time II

- The variability of the RTT is measure by the following formula
- The initial timeout value is 1 second, then after that, we use:

$$TimeoutInterval = EstimatedRTT + 4 \times DevRTT$$

- When a timeout occurs, the value of the timeout is doubled WHY??

[1] where $\alpha = 0.125$

Estimating RTT

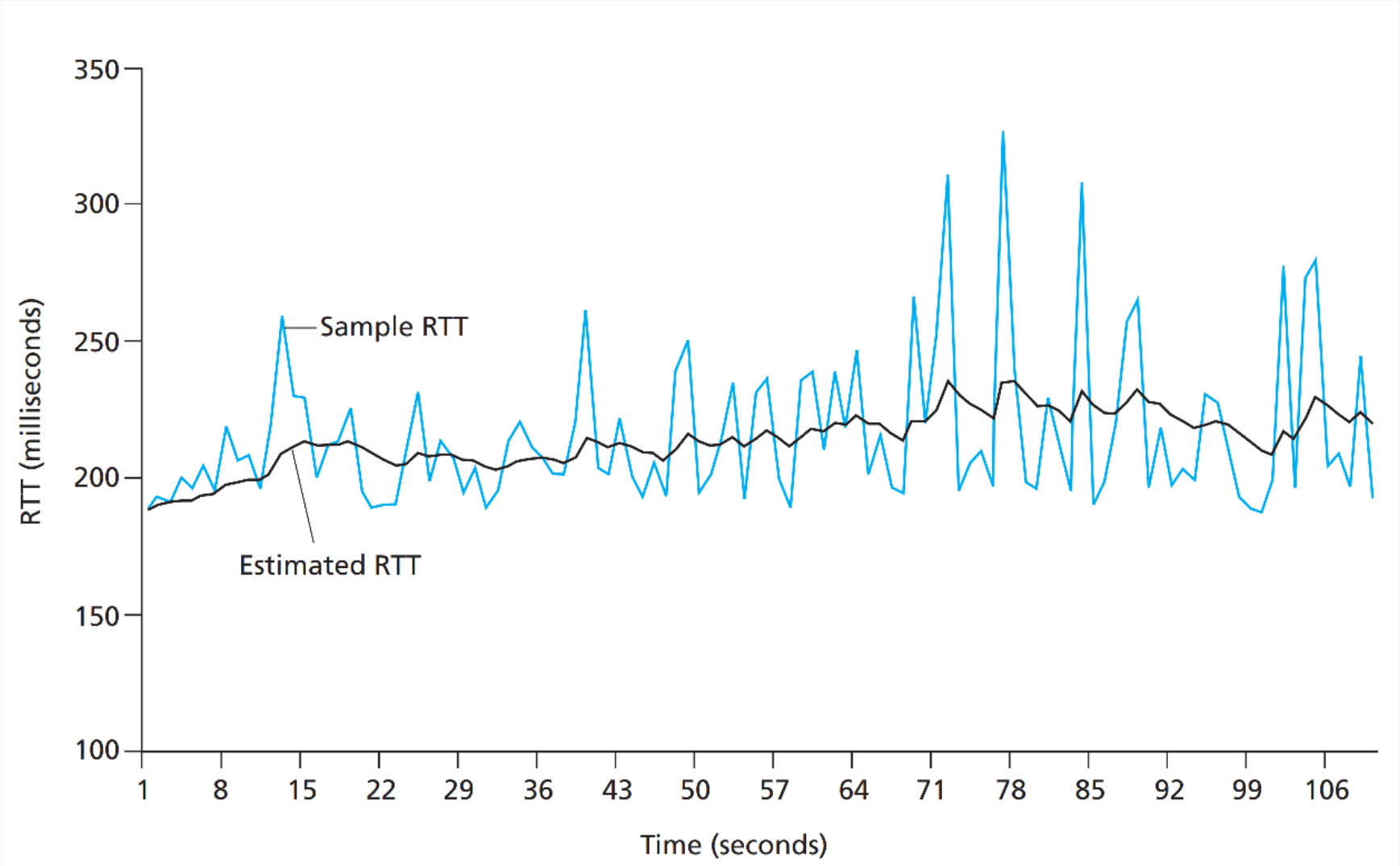
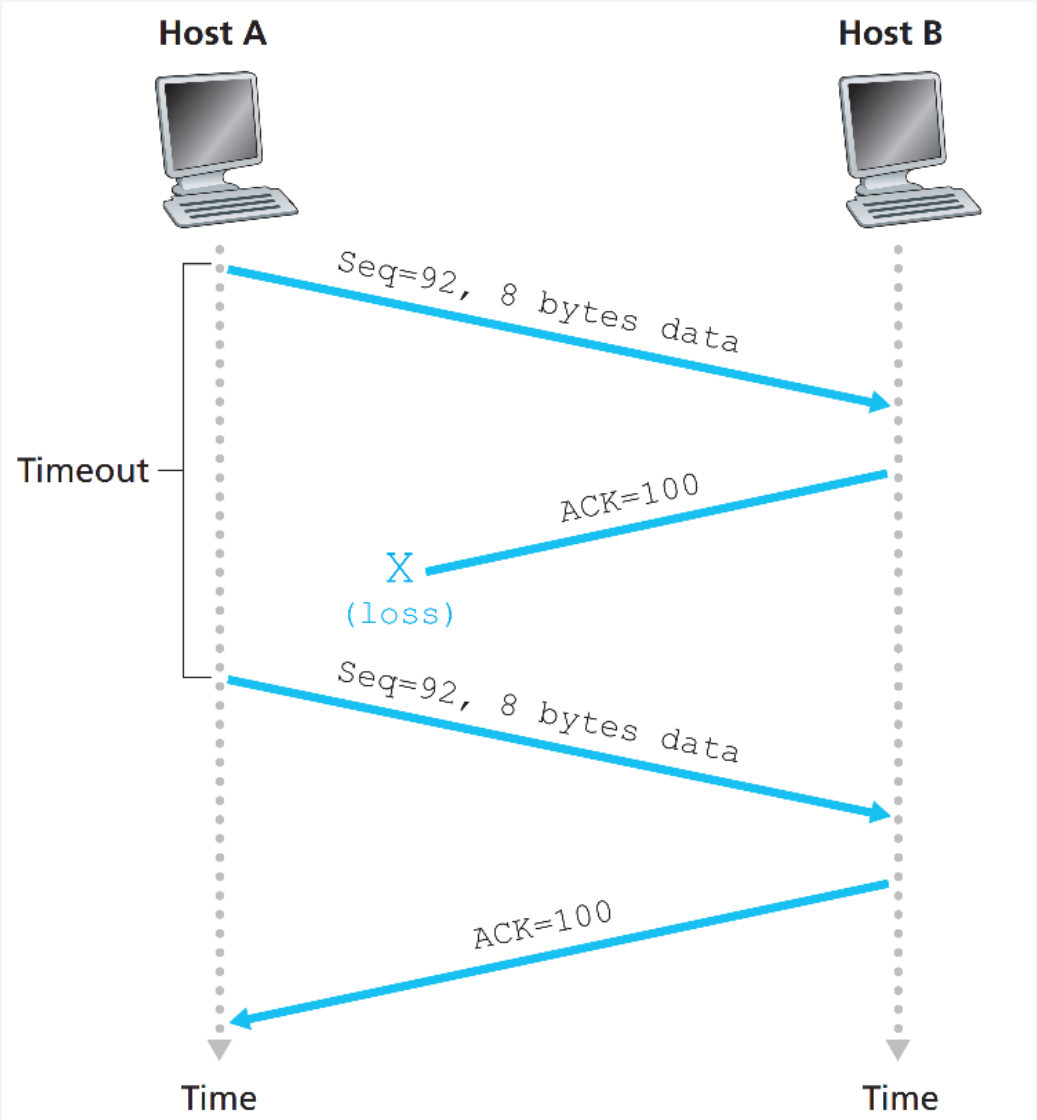
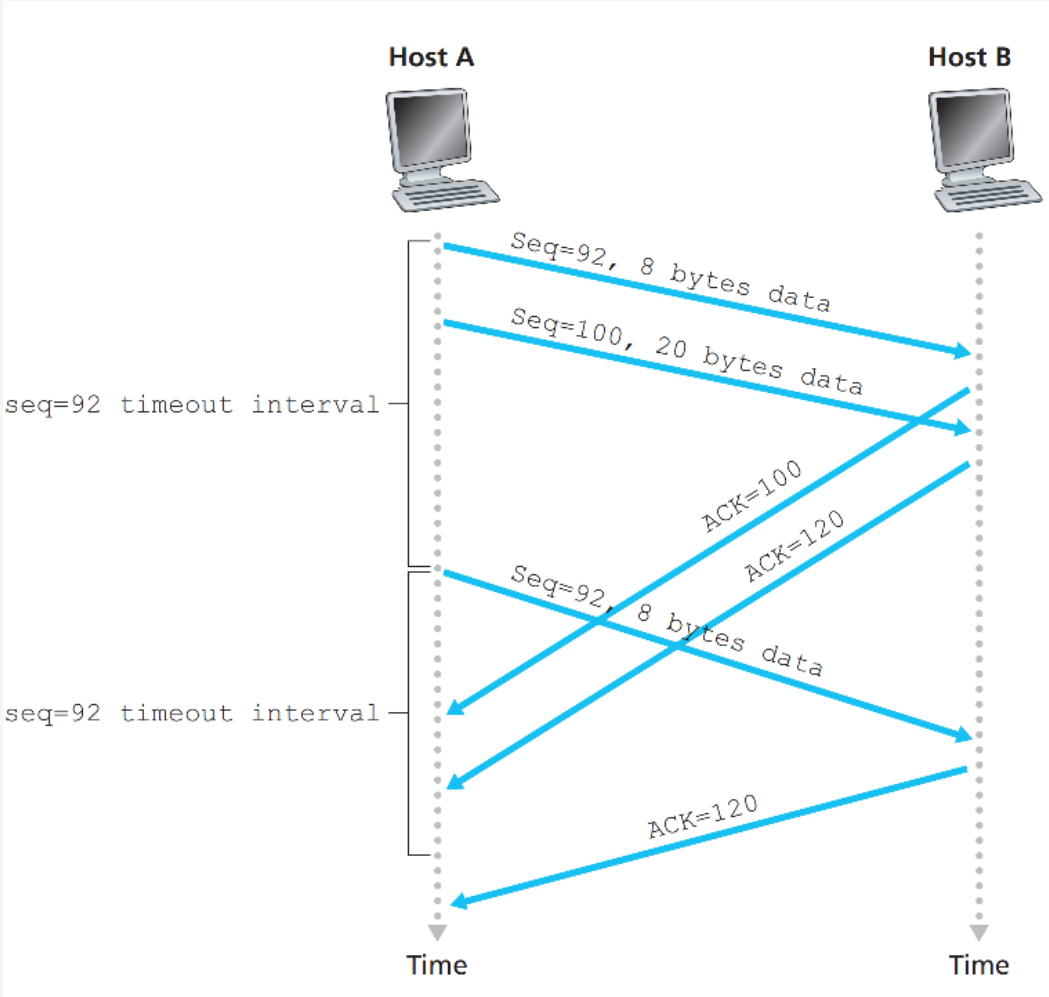


Figure: RTT samples and RTT estimates

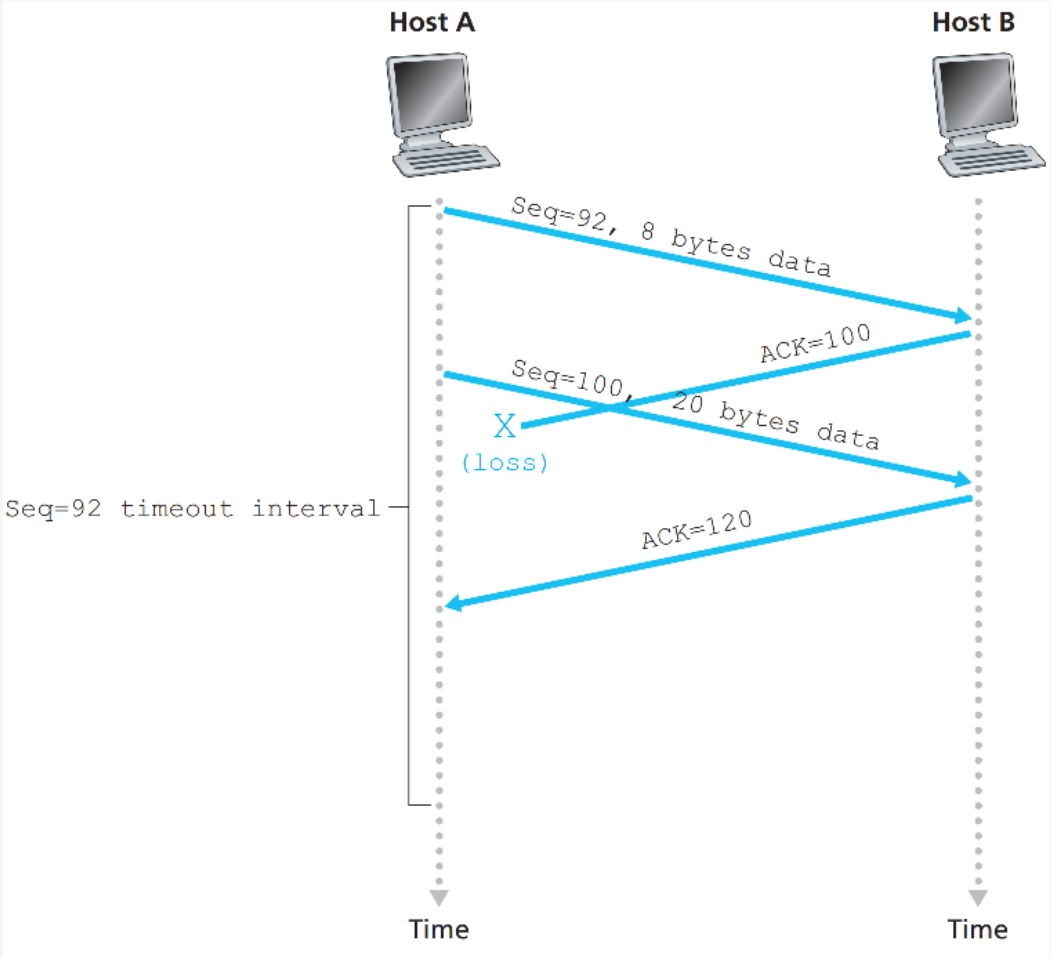
Retransmission due to a lost acknowledgment



Packets Arrived and Acknowledgement Delayed



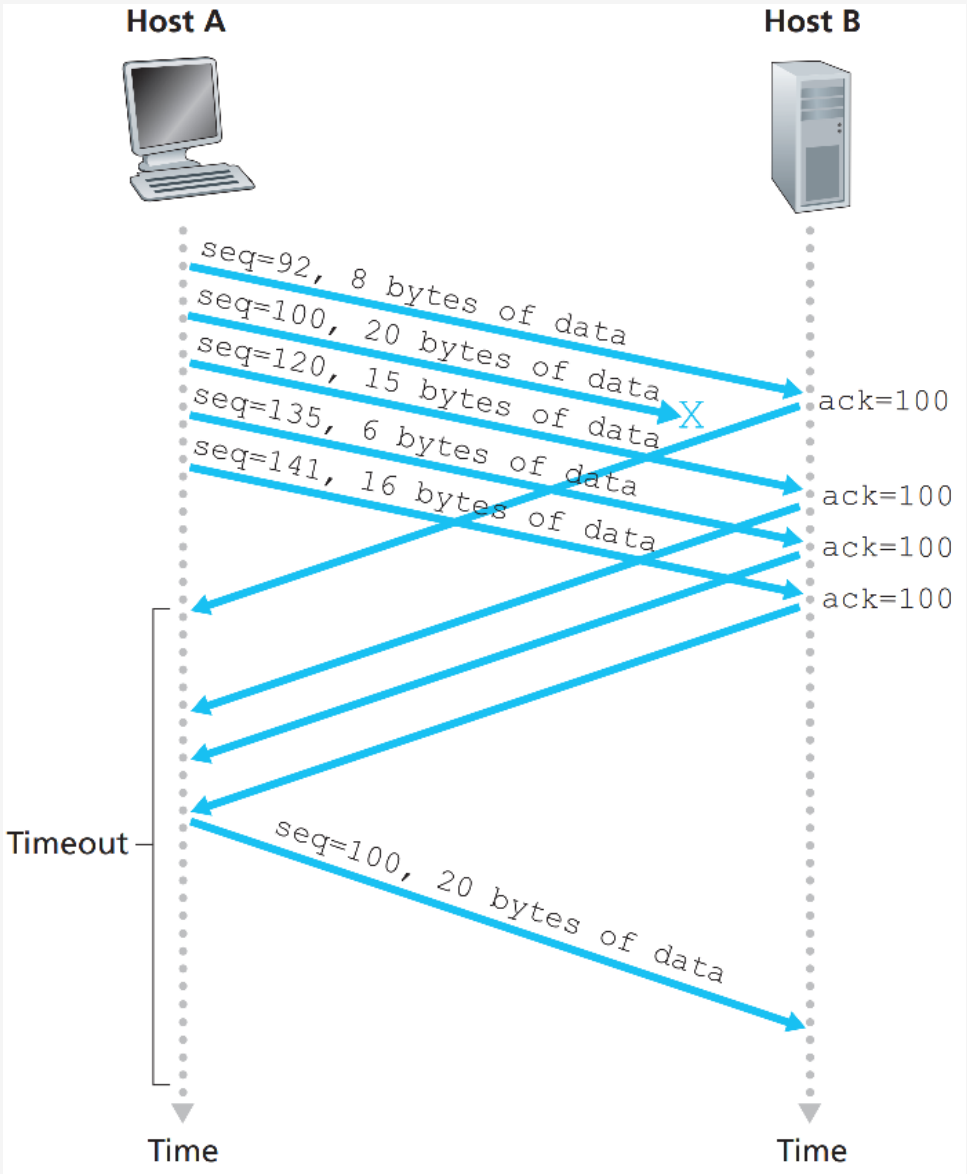
Cumulative Acknowledgment



TCP ACK Generation Recommendation [RFC 5681]

Event	TCP Receiver Action
Arrival of in-order segment with expected sequence number. All data up to expected sequence number already acknowledged.	Delayed ACK. Wait up to 500 msec for arrival of another in-order segment. If next in-order segment does not arrive in this interval, send an ACK.
Arrival of in-order segment with expected sequence number. One other in-order segment waiting for ACK transmission.	One Immediately send single cumulative ACK, ACKing both in-order segments.
Arrival of out-of-order segment with higher-than-expected sequence number. Gap detected.	Immediately send duplicate ACK, indicating sequence number of next expected byte (which is the lower end of the gap).
Arrival of segment that partially or completely fills in gap in received data.	Immediately send ACK, provided that segment starts at the lower end of gap.

Fast Retransmit



Flow Control I

- The TCP provides a flow control service; this service prevents the sender from sending data that could result in buffer-overflow at the receiver's side.
- If the receiver buffer becomes full, it will not be able to store any incoming data.
- The sender needs to slow down or stop sending data until the receiver buffer becomes free.
- The receiver keeps track of the last byte received and the last byte read by the application layer.

$$RcvBuffer \geq [LastByteRcvd - LastByteRead]$$

Flow Control II

- The sender keeps track of the receive window size (how much data it could send without overflowing the receiver.

$$rwnd = RcvBuffer - [LastByteRcvd - LastByteRead]$$

- The sender needs to make sure that the rwnd satisfies the following

$$rwnd \geq [LastByteSent - LastByteAcked]$$

What happens when $rwnd = 0$?

TCP Connection Management

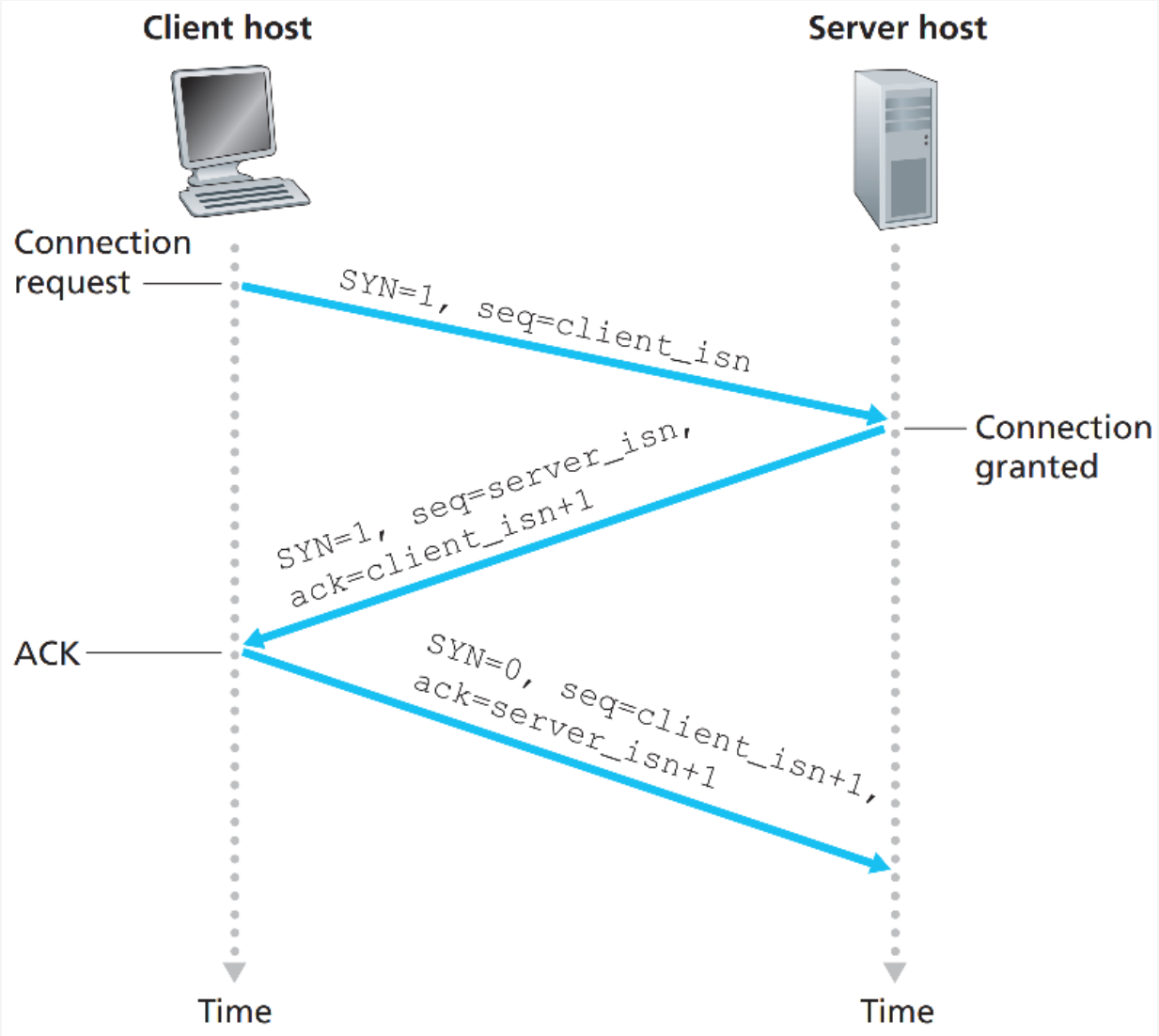


Figure: TCP three-way handshake

TCP Connection Management

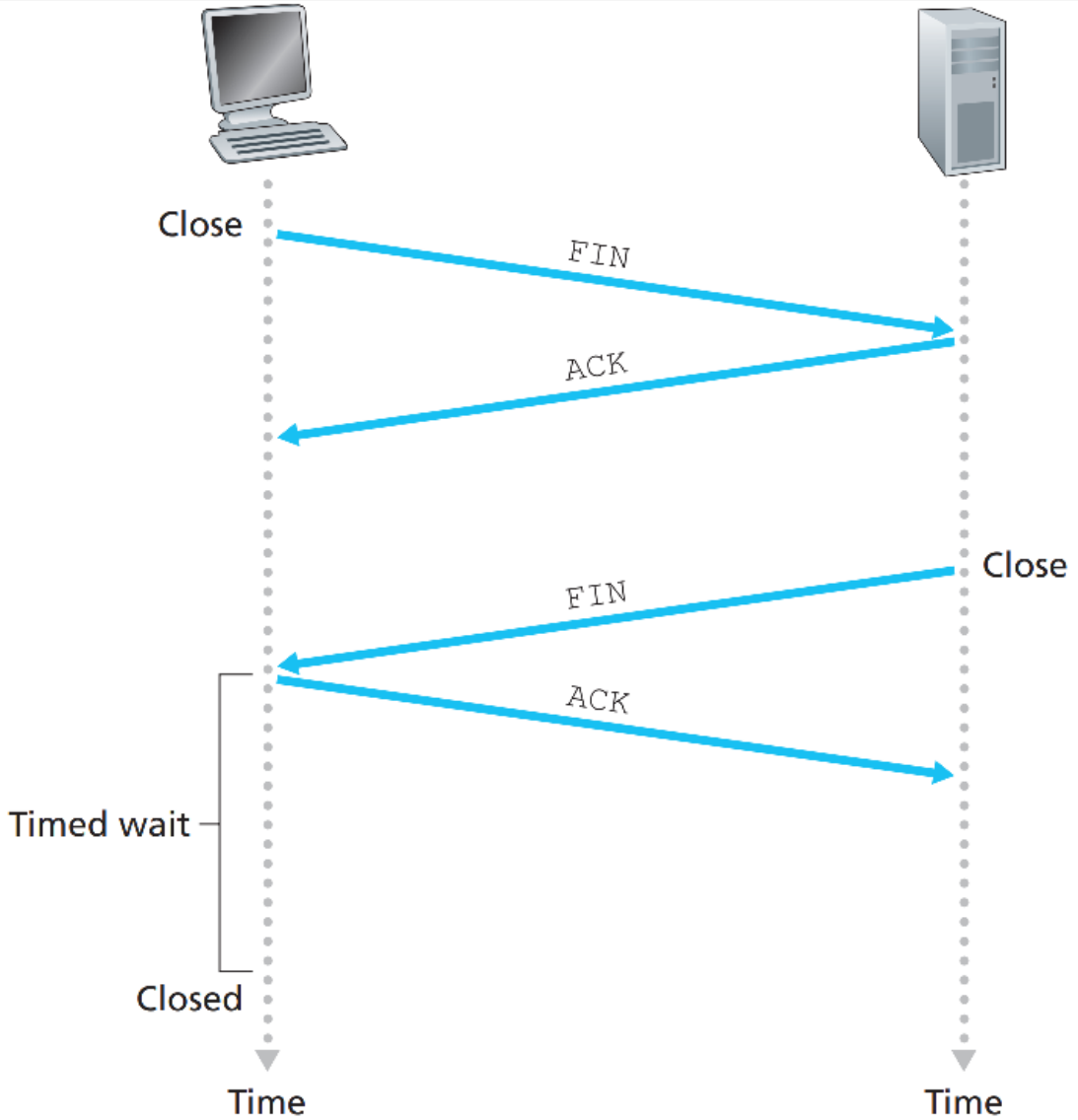


Figure: Closing a TCP connection

TCP Connection Client States

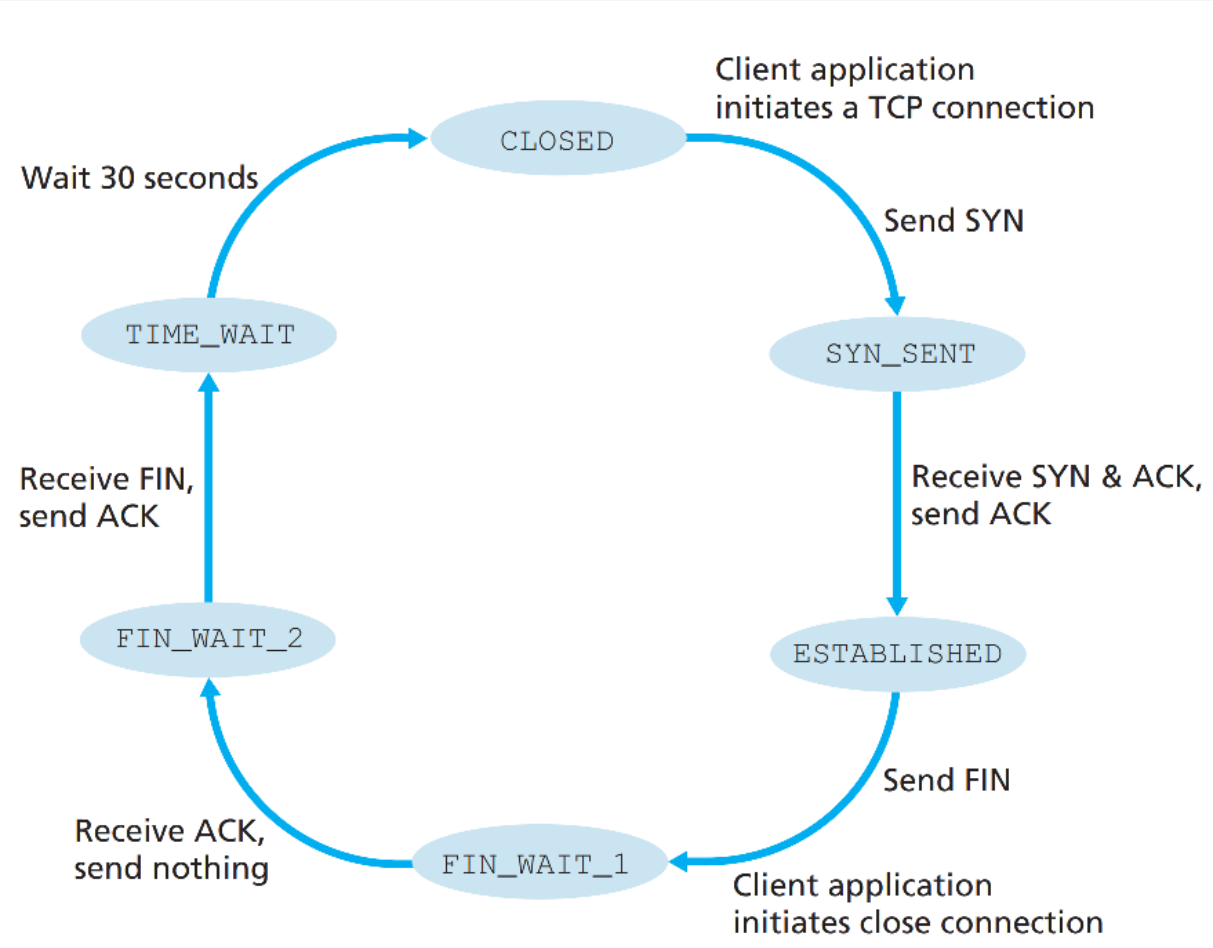


Figure: Fig 3.41

TCP Connection Server States

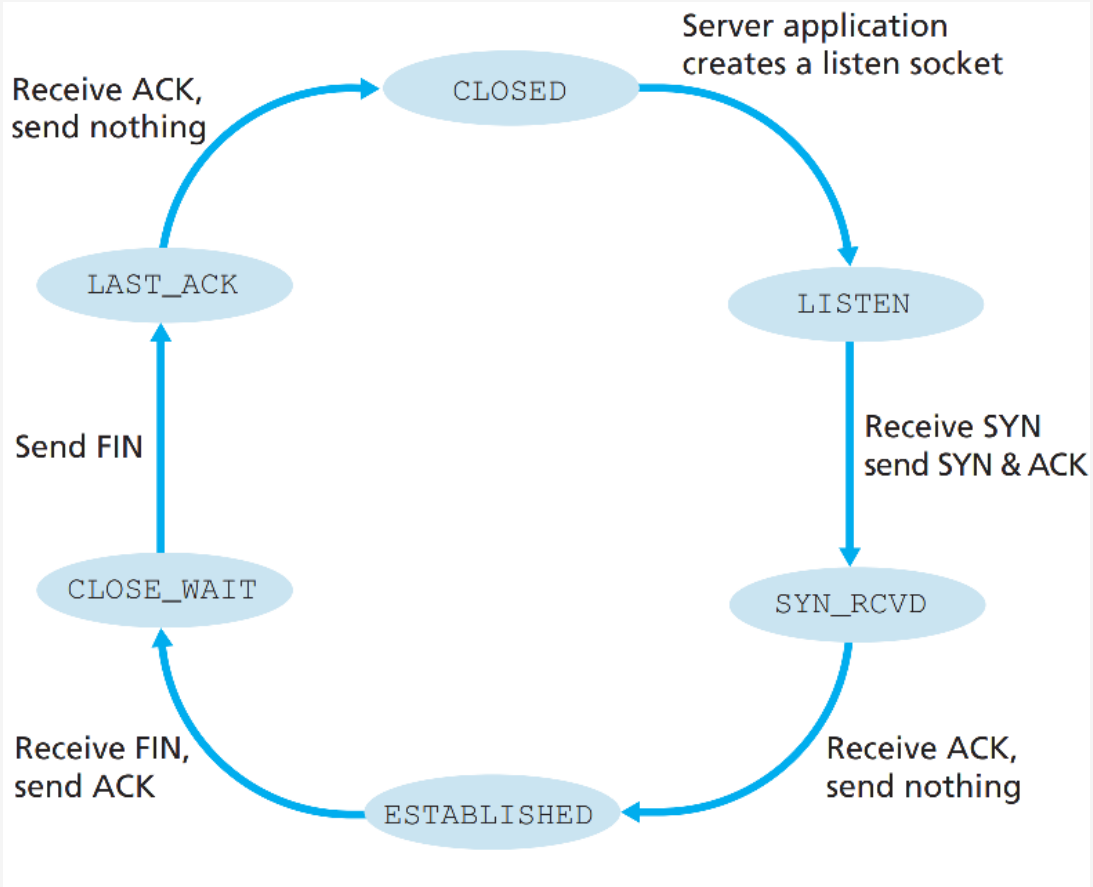


Figure: Fig 3.42

Congestion Control I

- Packet loss is the outcome of network congestion.
- Network congestions result in overflowing the routers' buffers and resources.
- Retransmission of lost packets or TCP segments helps in mitigating the impact of network congestion but does not solve the cause of network congestion.
- Congestion occurs when too many end systems attempt to send data simultaneously, which eventually overwhelms one or more routers.

Congestion Control II

- The packet switch networks, particularly the IP protocol, do not support the TCP to avoid reaching a congested state. Why??
- The TCP or any other transport service protocol running on top of the IP protocol needs to figure that the network is congested and then try to resolve the congestion.

TCP Congestion Control

- The TCP sender is responsible for detecting congestion on the path to the receiver's host.
- The TCP sender then is responsible for reducing or limit the rate at which it sends data over the congested path.
- The TCP sender is responsible for managing the congestion.

1. How to detect congestion?
2. How to control the sending rate of the sender?
3. How to manage the congestion?

How to detect congestion?

- The occurrence of timeout event
- Receiving duplicate acknowledgement
- or receiving three duplicates acknowledgment (ack a packet three times)

Both timeout and duplicate acknowledgements indicate network congestion, but which one indicate a more severe congestion, and why?

How to Control Sending Rate?

- The TCP sender uses a variable called congestion window (cwnd)
- This congestion window imposes a constraint on the rate at which the sender could send data

$$LastByteSent - LasteByteAcked \leq min\{cwnd, rwnd\}$$

- The value of cwnd will be updated based on the number of unacknowledged packets.

Principles of Congestion Control in TCP

- A lost segment indicates congestion, and the TCP sender's rate needs to be decreased. How?
- An acknowledged segment indicates that the network is operating and can deliver data to the receiver; this means we can increase the sender's rate.
- The TCP's approach for managing congestion keeps increasing the sending rate until congestion is detected; when the congestion occurs, it decreases the sending rate.
- When we reach a congestion-free state, it starts increasing the sending rate again

TCP congestion control algorithm

The algorithm alternates between three main stages: slow start, congestion avoidance, and fast recovery.

Slow Start

- The TCP set the value of the cwnd tp 1 MSS, this means the initial sending rate is MMS/RTT
- Then on successful acknowledgement, it doubles the cwnd size $cwnd = cwnd \times 2$
- Then when a failure happens (timeout), it reset the cwnd and set a cwnd threshold (ssthresh) to be cwnd/2

Slow Start

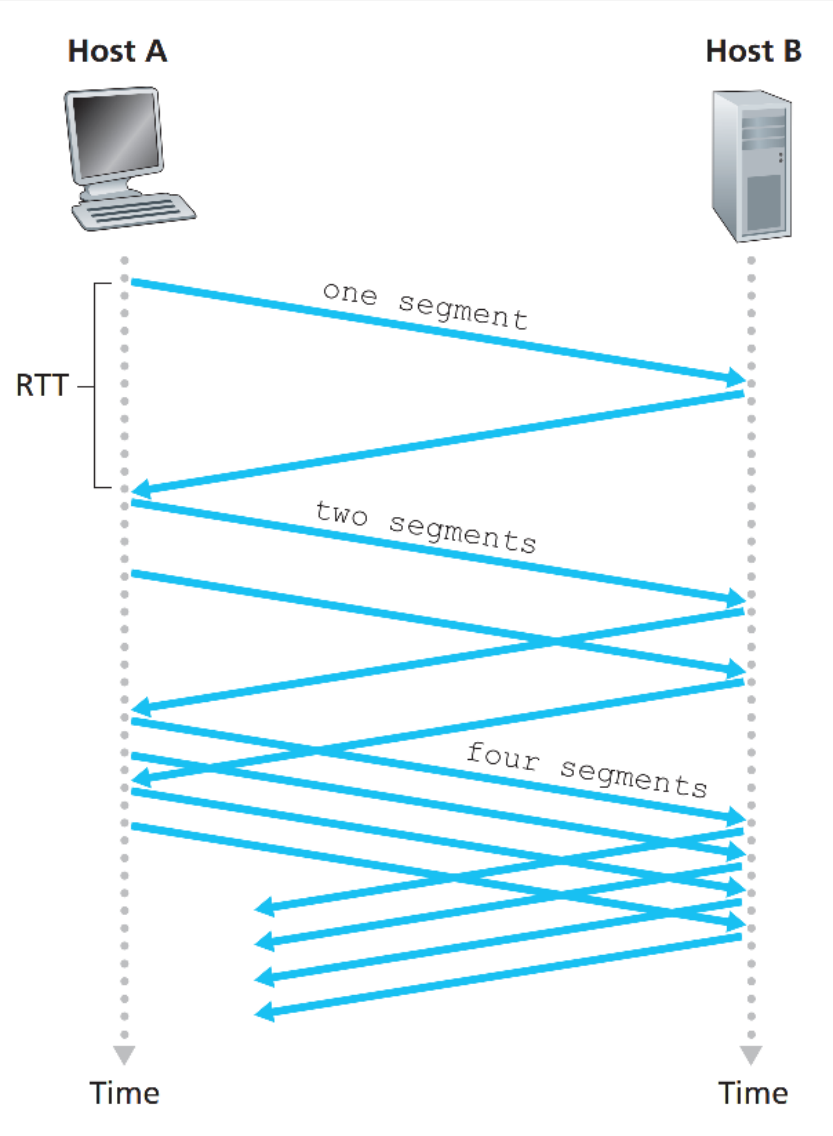


Figure: Slow Start Operation

TCP congestion control algorithm

Congestion Avoidance

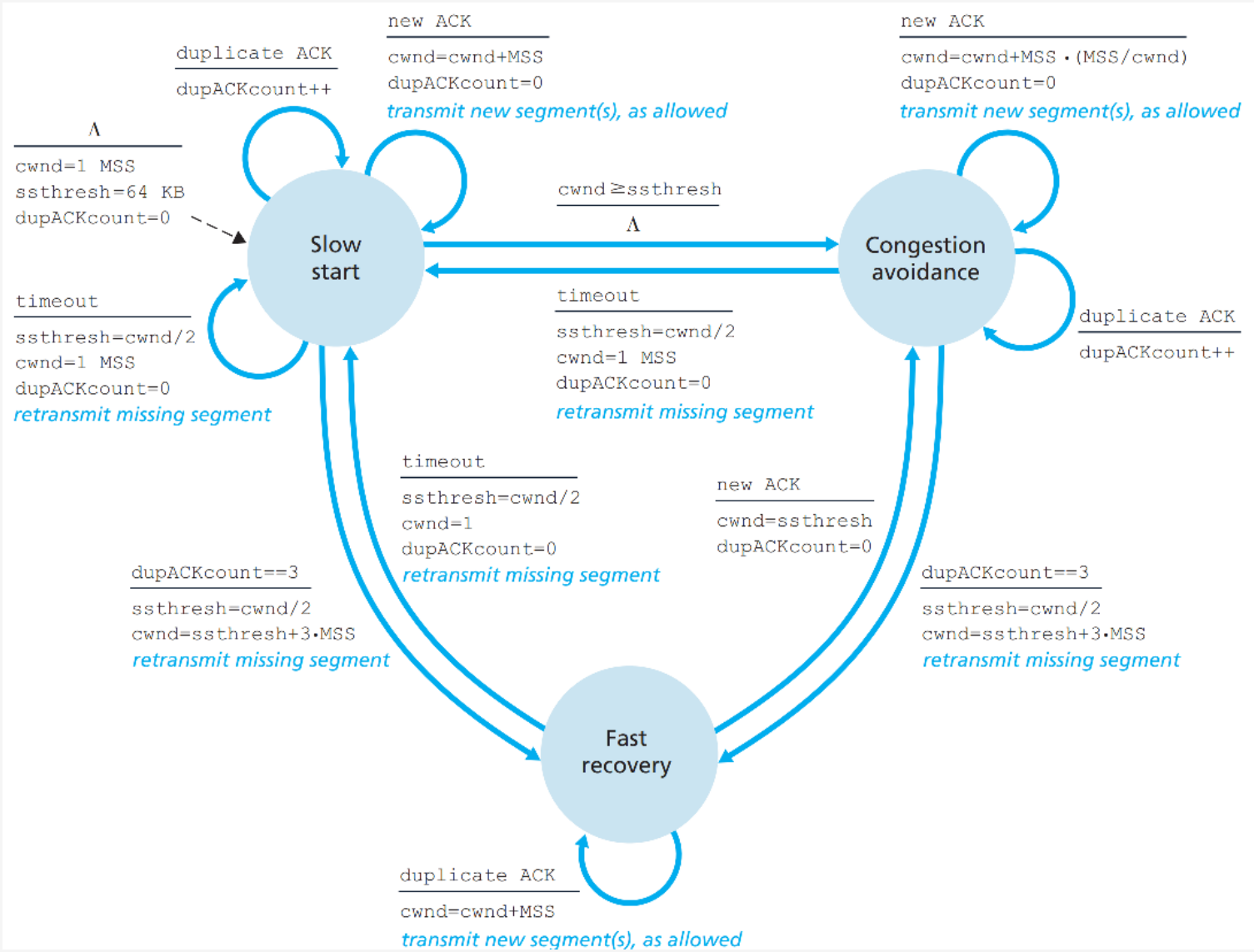
- When the value of cwnd reaches the value of ssthresh, the TCP enters the congestion avoidance state.
- When cwnd = ssthresh, the next double increment of the cwnd is the same cwnd value that we detected the congestion at this point.
- Then, rather than doubling the cwnd, the tcp increase it by 1 MSS or by MSS/cwnd
- As long as the TCP receiving acknowledgment or double acknowledgment, it remains in the congestion avoidance state
- When a timeout is detected, the TCP again reset the cwnd to 1 and set the ssthresh to be cwnd/2 and enter the slow start again.

TCP congestion control algorithm

Fast Recovery

- If the TCP sender received a triple acknowledgment, then it enters the fast recovery state.
- The TCP could move from a slow start or congestion avoidance to fast recovery.
- In fast recovery, the cwnd increased by 1 for every successful acknowledgment.
- If time out go to slow start
- If and acknowledgment and cwnd = ssthresh then go to congestion avoidance.

TCP congestion control



TCP Congestion Control Summary

- Different versions of the TCP implementation will set the conditions for switching between states a bit differently. For instance, TCP Tahoe does not have a fast recovery state, while TCP Reno has a fast recovery state.
- The TCP congestion control follows an additive-increase, multiplicative decrease (AIMD) pattern
- The TCP congestion control algorithm is a distributed asynchronous optimization algorithm

Additive-Increase, Multiplicative Decrease

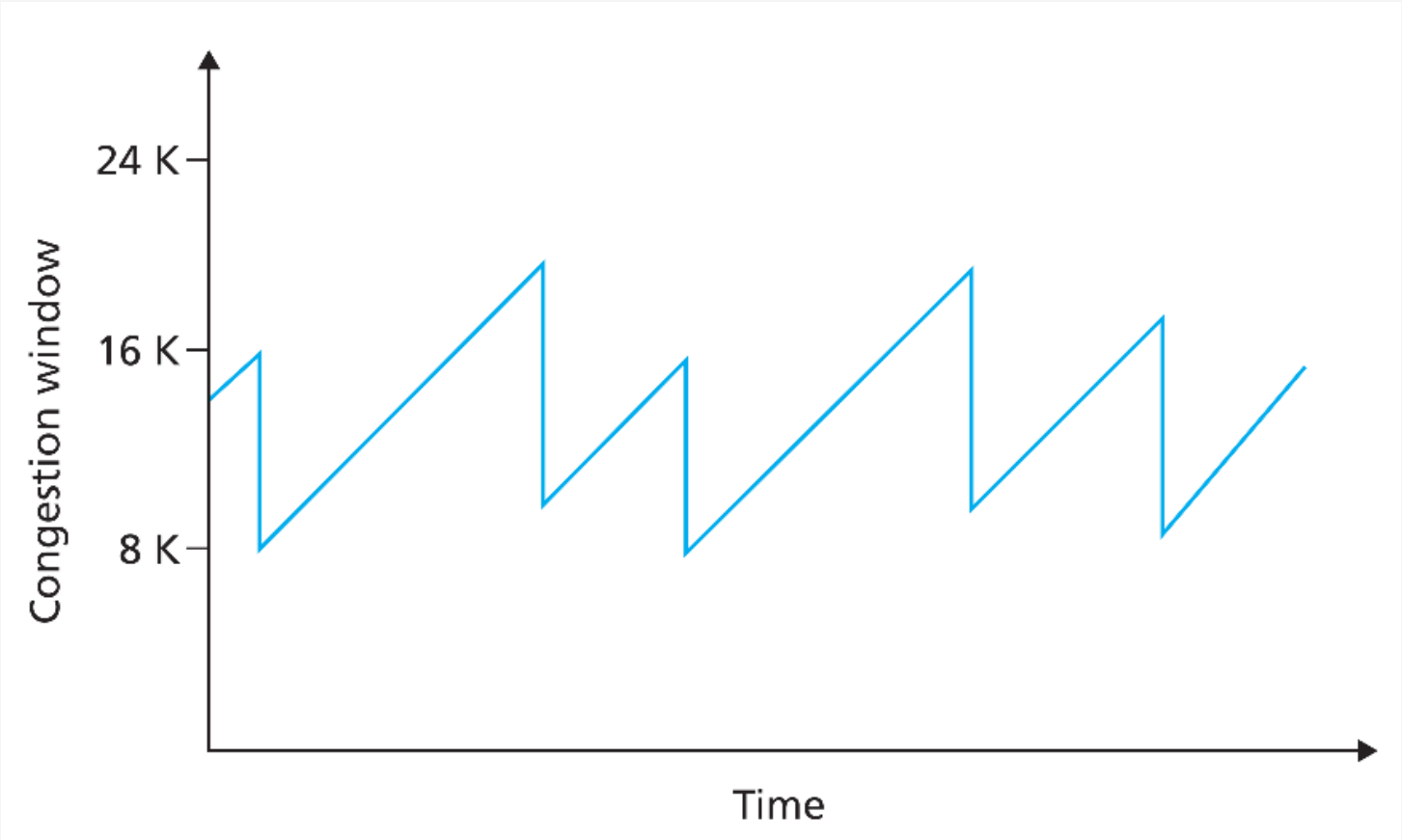


Figure: Additive-increase, multiplicative-decrease congestion control

Fairness and Congestion Control

- For a given link with transmission rate $\boldsymbol{R}$ bps, and $\boldsymbol{K}$ TCP connections are going through this link.
- A congestion control mechanism is said to be fair if the average transmission rate of each connection is approximately $\frac{R}{K}$; that is, each connection gets an equal share of the link bandwidth.
- In general, the TCP congestion control (TCP's AIMD algorithm) is fair to expect under two scenarios:
  - TCP congestion control regulates an application's transmission rate via the congestion window mechanism, but in the presence of UDP connections, the UDP is likely to crowd out TCP traffic.
  - Fairness is also not guaranteed in the case of parallel TCP connections by the same process or application.

Explicit Congestion Notification

- TCP sender receives no explicit congestion indications from the network layer.
- TCP sender relies on timeout events and duplicate acknowledgments to detect congestion (after the fact)
- Modern updates and extensions of the IP and TCP have been proposed to enable the network layer to notify the TCP sender and receiver about congestion.
- Explicit Congestion Notification, the router that experiences congestion can set a flag in the network layer header of the packets passing through.
- The arrival of a packet with an ECN flag informs the receiver that it should informs the sender to decrease the sending rate.

Explicit Congestion Notification

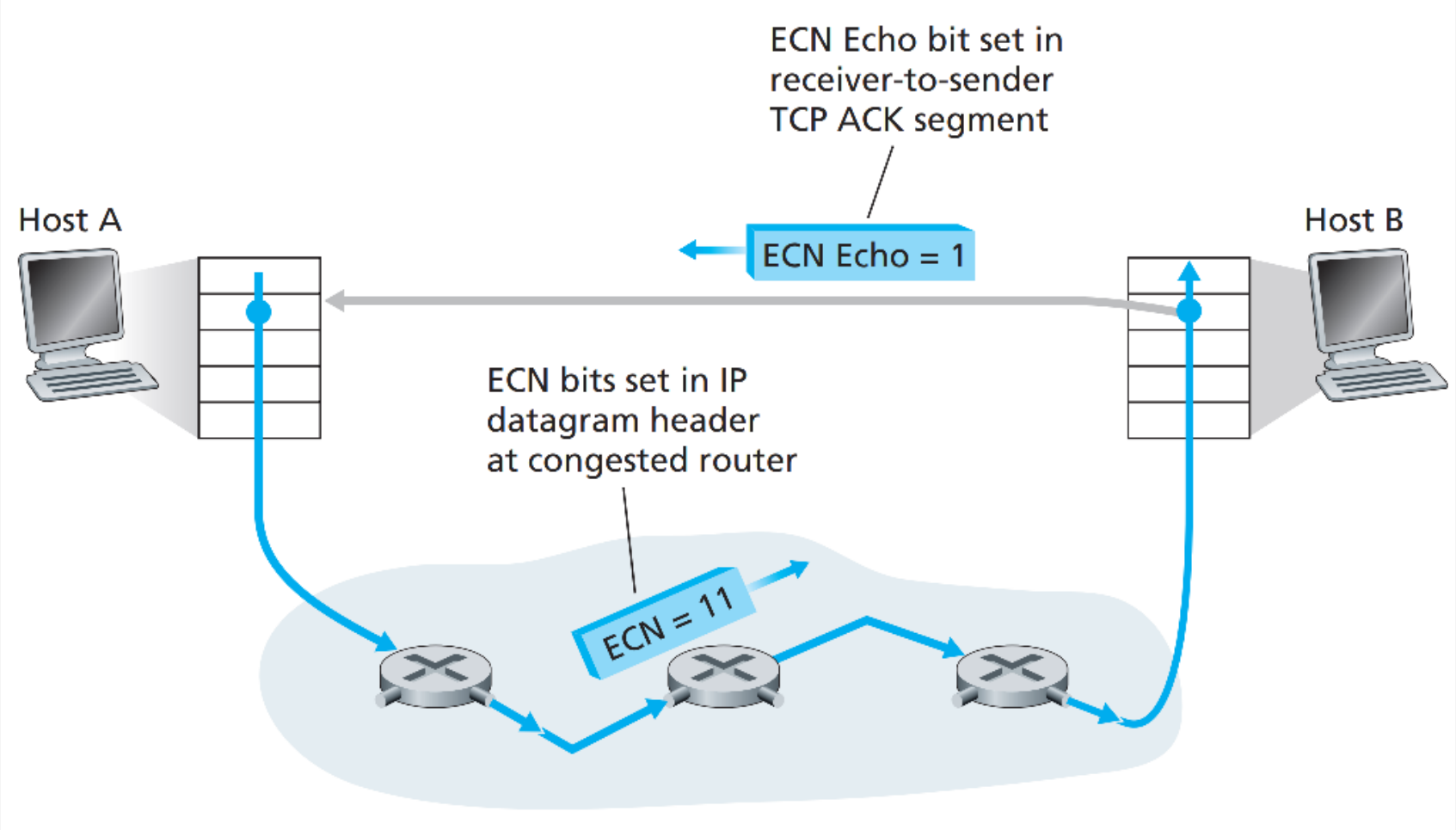


Figure: Fig 3.53

What is the plan for next week?

- Network Layer (The Data Plane)
- Quiz 3 (6%) cover the "Transport Layer"

The End (Questions?)

References I


James Forshaw, *Attacking network protocols: A hacker's guide to capture, analysis, and exploitation*, 1st ed., No Starch Press, USA, 2017.


John S. Gero and Thomas Mc Neill, *An approach to the analysis of design protocols*, Design Studies **19** (1998), no. 1, 21–61.


James F. Kurose and Keith W. Ross, *Computer networking: A top-down approach*, 7 ed., Pearson, Boston, MA, 2016.


M.Rose, *Beep: Building blocks for application protocols.*, 2001.


Rose and Malamud, *Rfc3117:on the design of application protocols*, 2001.


Andrew S. Tanenbaum and Maarten van Steen, *Distributed systems: Principles and paradigms*, 2 ed., Pearson Prentice Hall, Upper Saddle River, NJ, 2007.