CS 352
Reliability: Sliding Windows

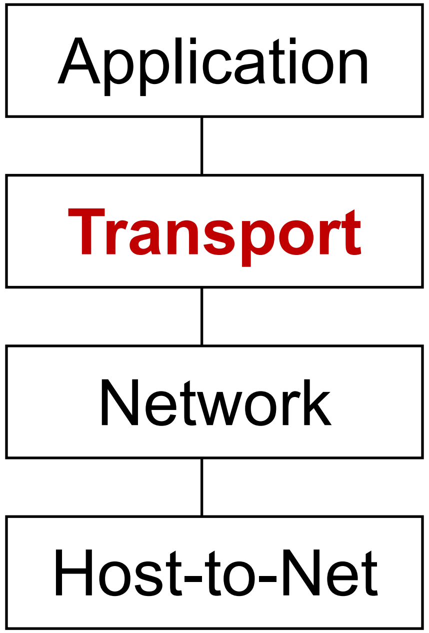
CS 352, Lecture 10.1

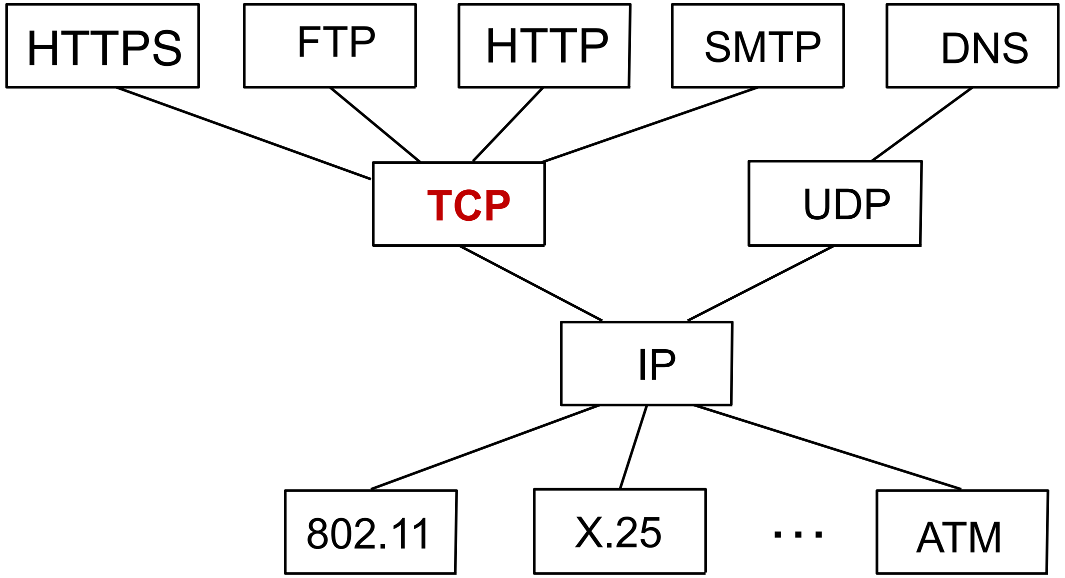
http://www.cs.rutgers.edu/~sn624/352

Srinivas Narayana

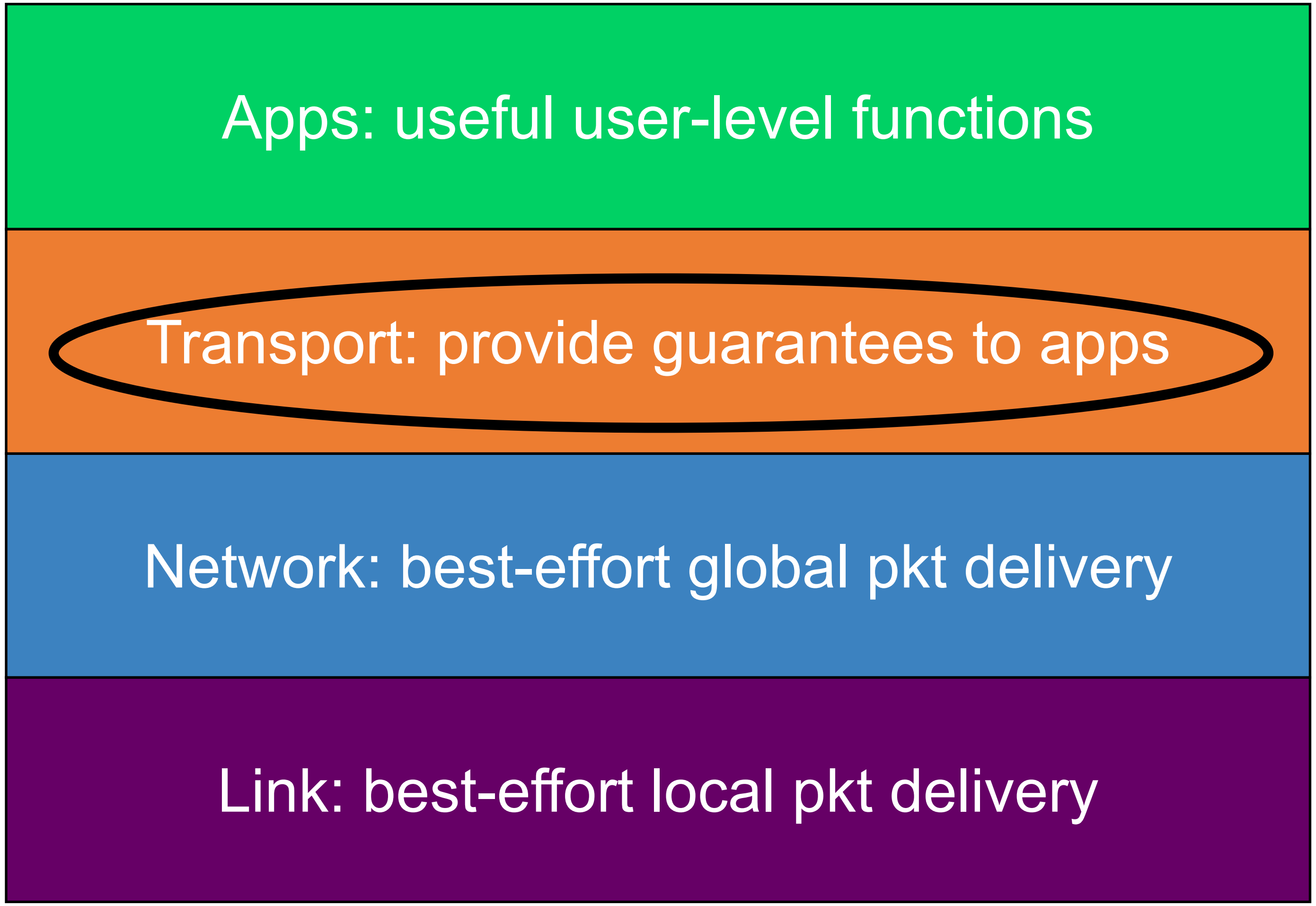
RUTGERS
UNIVERSITY | NEW BRUNSWICK

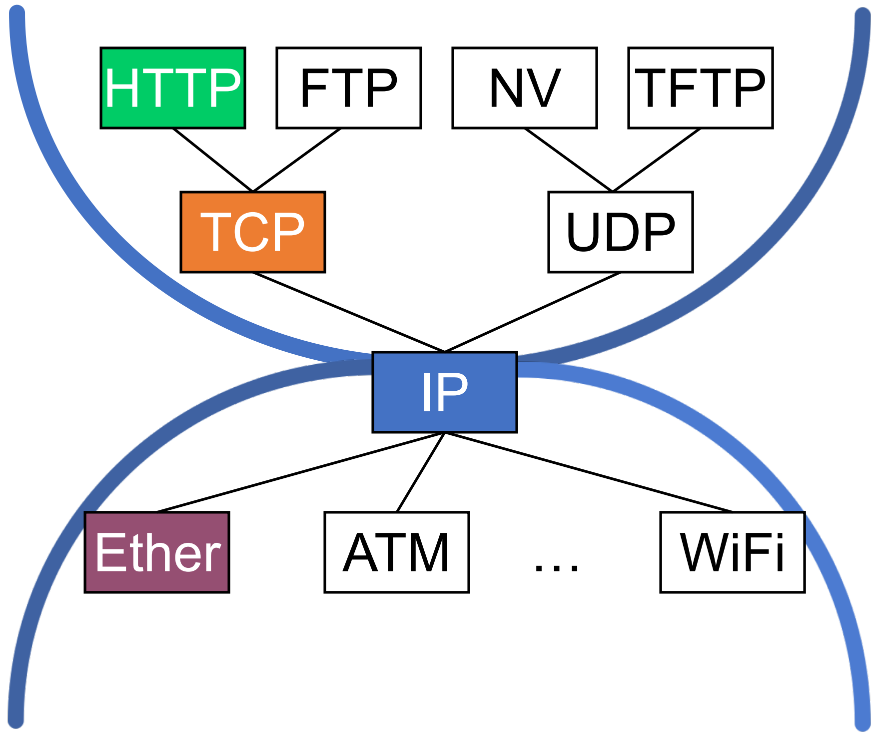
Transport


Application
Transport
Network
Host-to-Net


HTTPS
FTP
HTTP
SMTP
DNS
TCP
UDP
IP
802.11
X.25
...
ATM

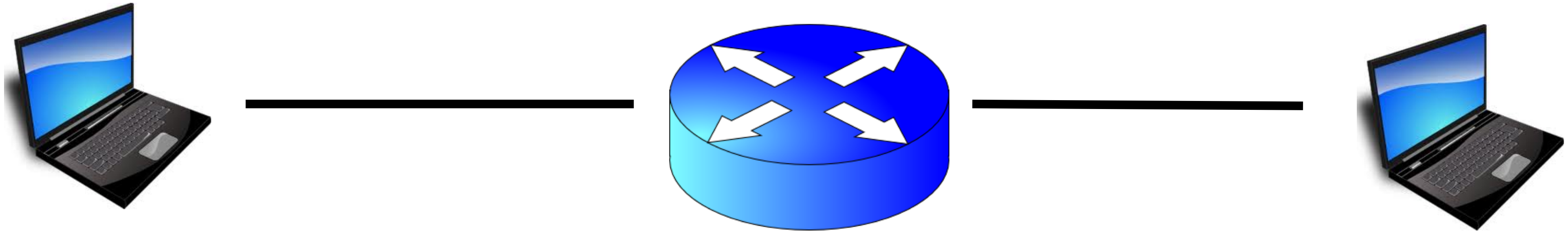
Modularity through layering





How do apps get perf guarantees?

- The network core provides no guarantees on packet delivery

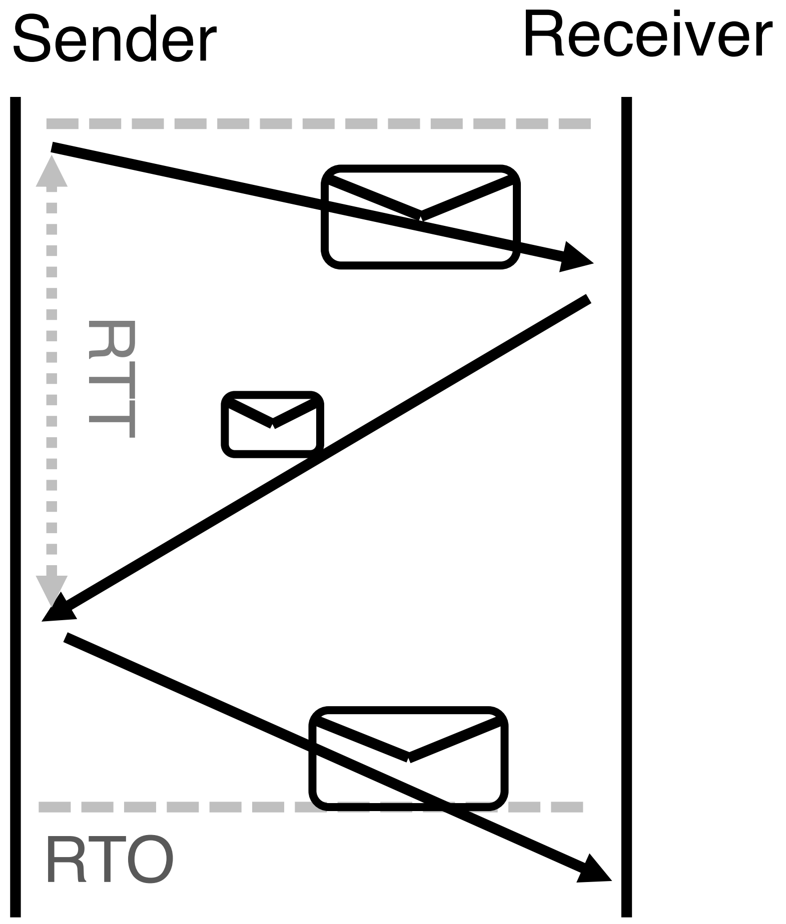


- Transport software on the endpoint oversees implementing guarantees on top of a best-effort network
- Three important kinds of guarantees
  - Reliability
  - Ordered delivery
  - Resource sharing in the network core

Transmission Control Protocol (TCP)

Review: Stop-and-Wait Reliability

- Stop and wait: sender waits for an ACK/RTO before sending another packet
- Suppose no packets are dropped
  - RTT = RTO = 100 milliseconds
  - Packet size: 12 Kbit (1 K = 10^3)
  - Link rate: 12 Mbit/s (1 M = 10^6)
- Rate of data transmission?
  - one packet per RTT = 12Kbits / 100ms = 120 Kbit/s



120 Kilobit/s == 1% of link rate

Making reliable transmissions efficient

- Terminology: unACKed data / packets in flight
  - Data that has been sent, but not known (by the sender) to be received
- With just one packet in flight, the data rate is limited by the packet delay (RTT) rather than available bandwidth (link rate)
  - Larger the delay, slower the data rate, regardless of link rate
- Idea: Keep many packets in flight!
  - More packets in flight improves throughput
- We say such protocols implement pipelined reliability

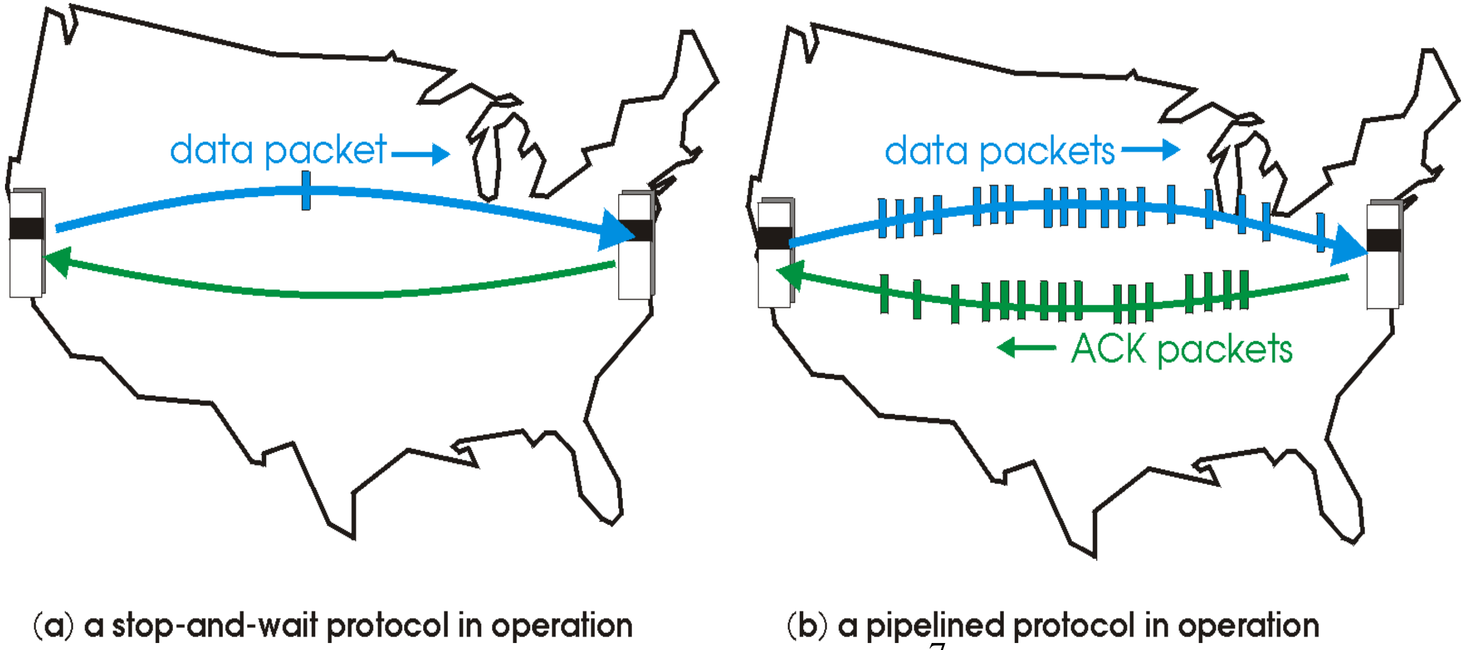
Why does pipelined reliability help?

Suppose sender has multiple, in-flight (yet-to-be-acknowledged) packets

New packets transmitted *concurrently* with in-flight packets

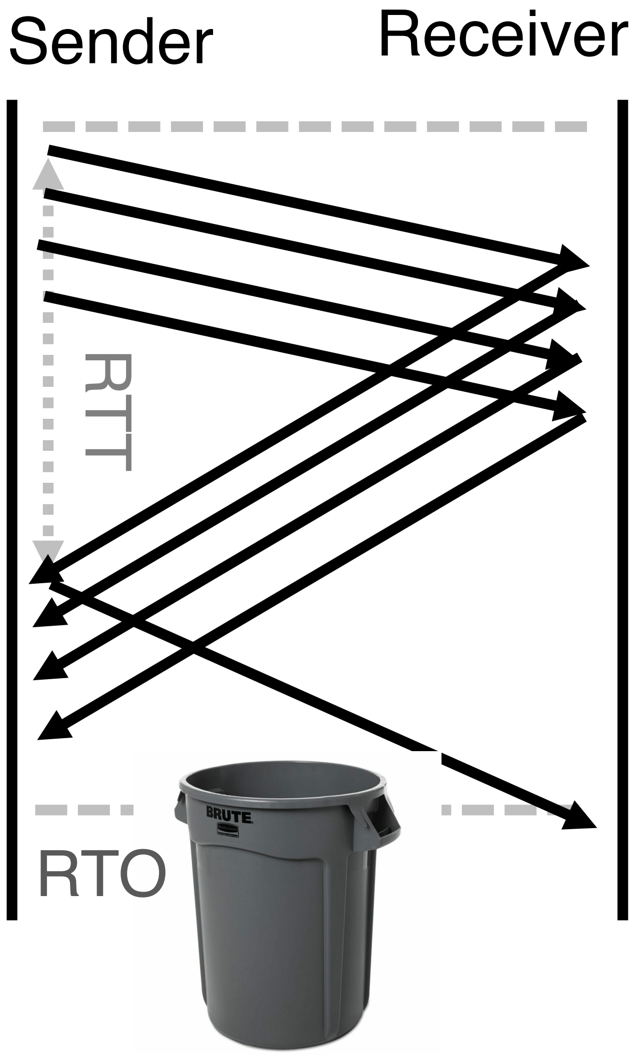
Packets and ACKs (of prior packets) are concurrently transmitted

➔ More data and ACKs transmitted within the same duration



Tracking packets in flight

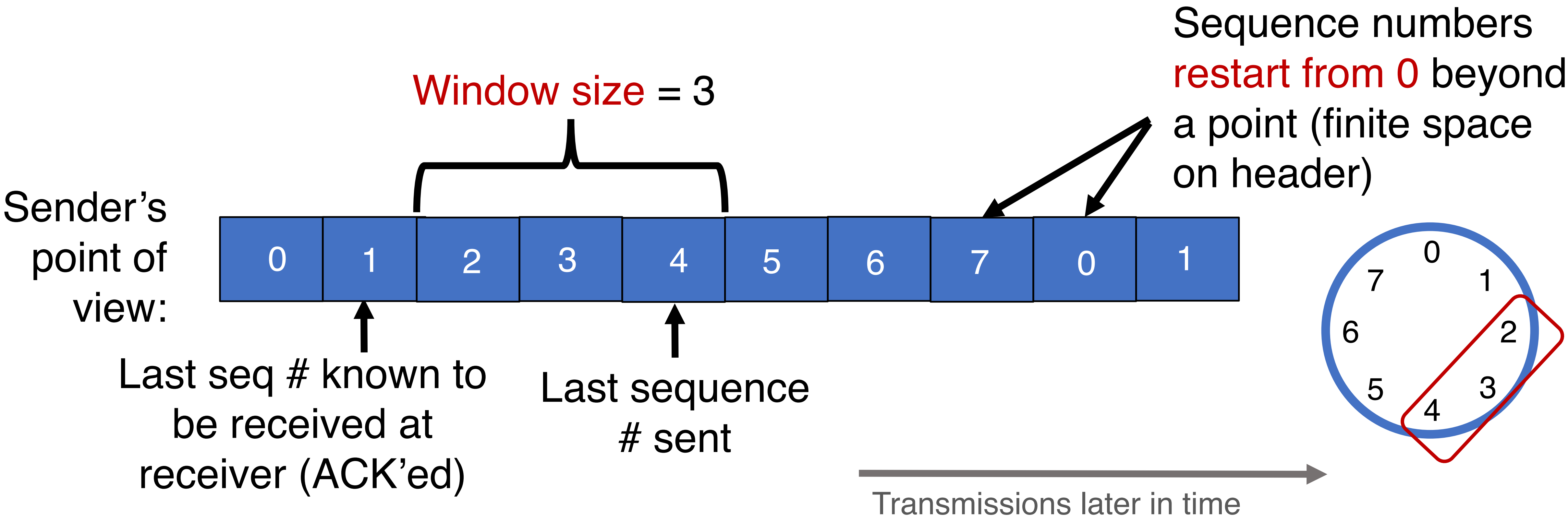
- If there are N packets in flight, throughput improves by N times relative to stop-and-wait.
  - Stop and wait: send 1 packet per RTT
  - Pipelined: send N packets per RTT
- We term the in-flight data the window
- We term the amount of in-flight data the window size



Sliding Windows

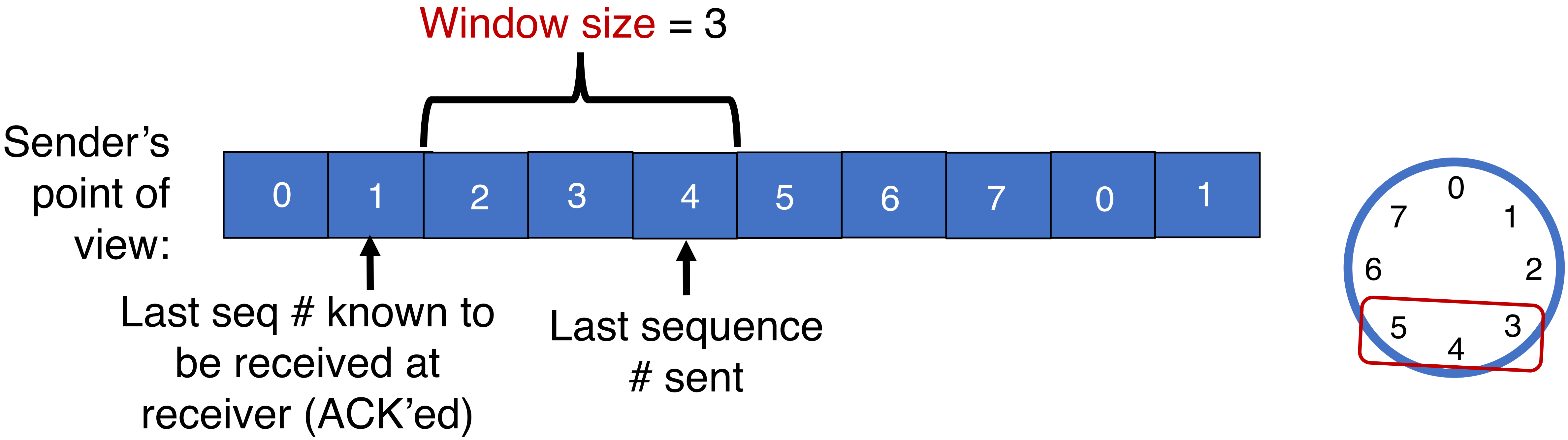
Window

- Window: Sequence numbers of in-flight data
- Window size: The amount of in-flight data (unACKed)



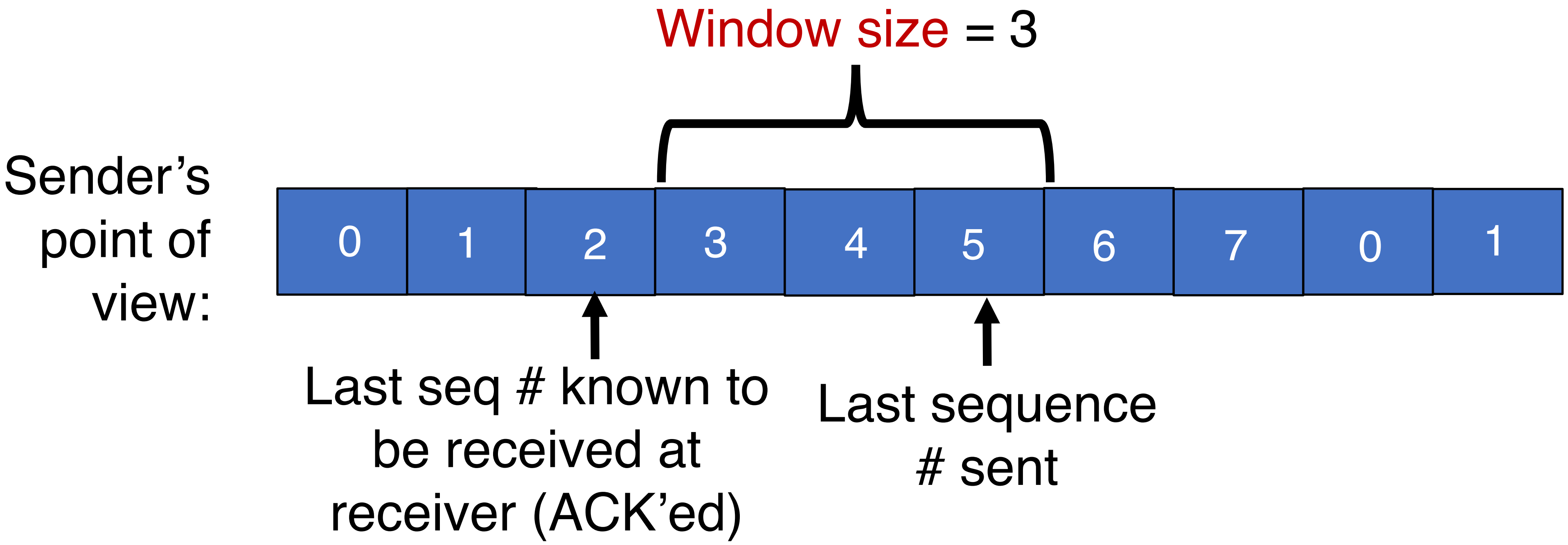
Sliding window (sender side)

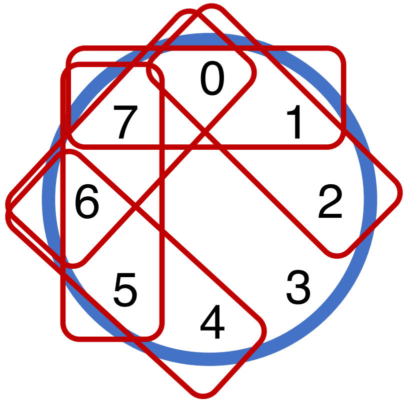
- Suppose sequence number 2 is acknowledged by the receiver
  - Sender can transmit sequence # 5
  - The window “slides” forward



Sliding window (sender side)

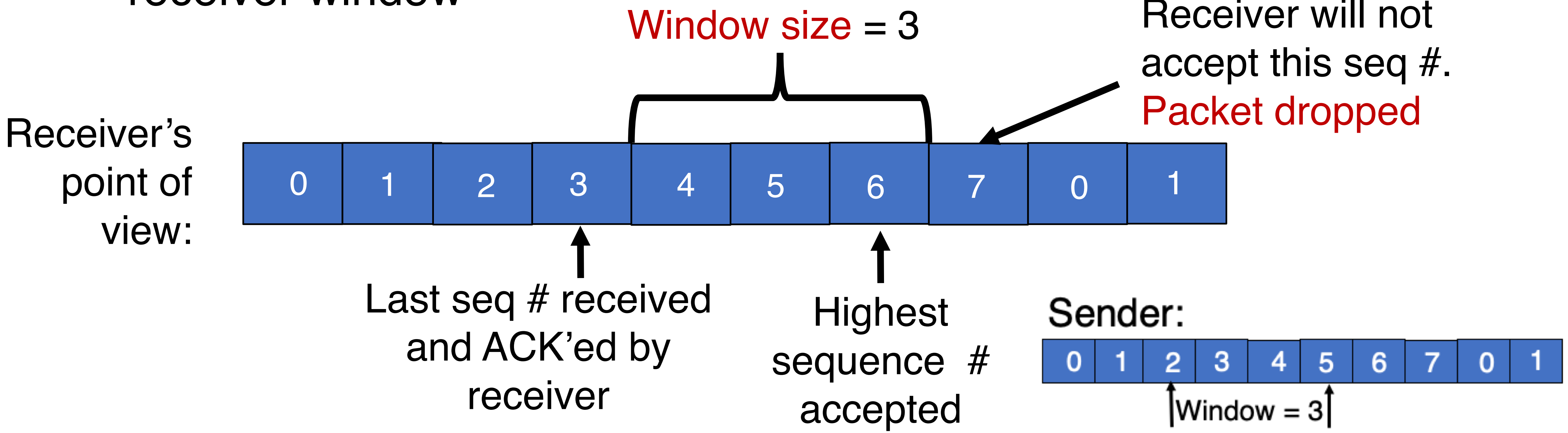
- Suppose sequence number 2 is acknowledged by the receiver
  - Sender can transmit sequence # 5
  - The window "slides" forward





Sliding window (receiver side)

- Window of in-flight packets can look different between sender and the receiver: receiver has more recent info of in-flight
- Receiver only accepts sequence #s as allowed by the current receiver window



Summary of sliding windows

- Sender and receiver can keep several packets of in-flight data
  - Book-keep the sequence numbers using the window
- Windows slide forward as packets are ACKed (at receiver) and ACKs are received (at sender)
- Common case: Improve throughput by sending and ACKing more packets in the same duration
- Key challenge: how should the sender and receiver collaboratively track the packets that must be retransmitted?

CS 352
Making Retransmissions Efficient

CS 352, Lecture 10.2

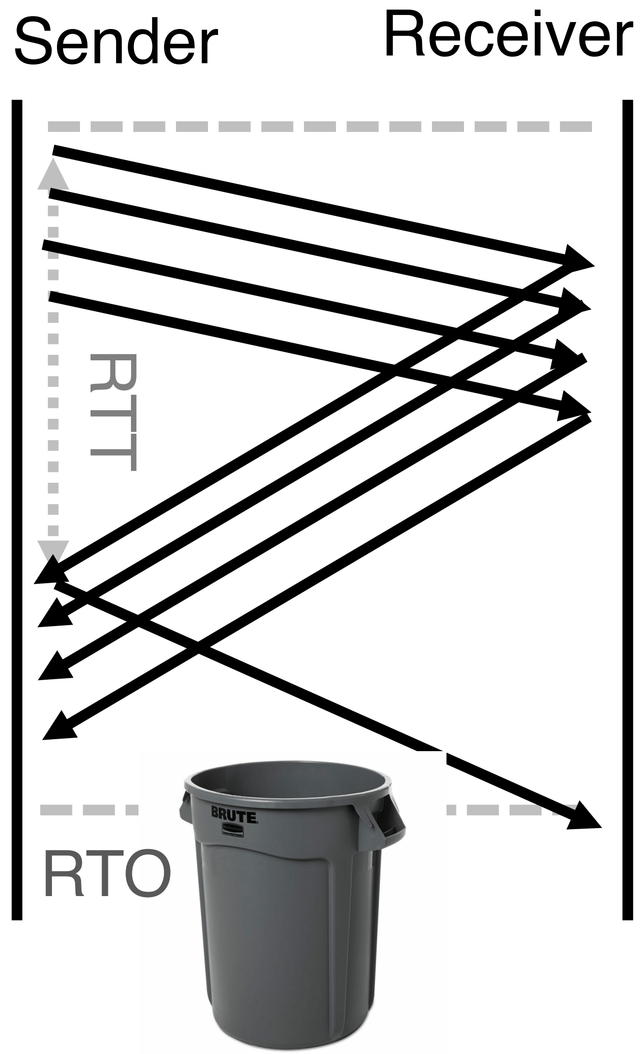
http://www.cs.rutgers.edu/~sn624/352

Srinivas Narayana

RUTGERS
UNIVERSITY | NEW BRUNSWICK

Pipelined Reliability

- If there are N packets in flight, throughput improves by N times relative to stop-and-wait.
  - Stop and wait: send 1 packet per RTT
  - Pipelined: send N packets per RTT



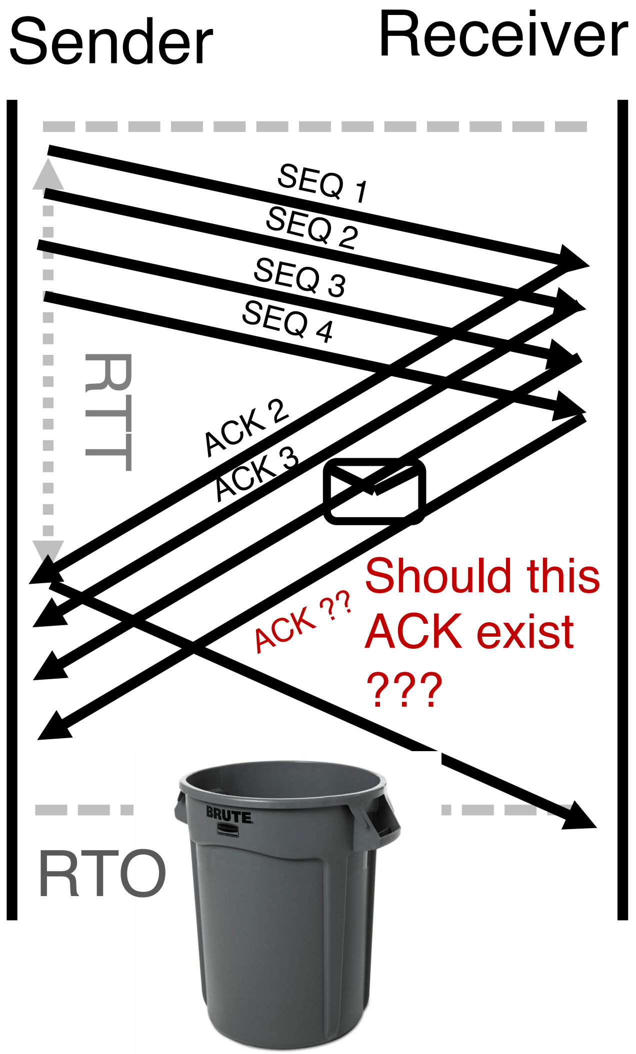
- Q1: how should sender efficiently identify which pkts were dropped and (hence) retransmitted?

- Q2: how much data to keep in flight (i.e., what is N?) to reduce drops/retransmits?

Q1. Identifying the Dropped Packets

Q1: Identifying dropped packets

- Suppose 4 packets were sent, but one was dropped. How does sender know which one(s) were dropped?
- Recall: Receiver writes sequence numbers on the ACK
  - Sender infers which bytes were received successfully using the ACK #s
- Q1.1: Should receivers ACK subsequent packets upon detecting data loss?
- Q1.2: If so, what sequence number should receiver put on the ACK?



Receiver strategies upon packet loss

ACK subsequent pkts?

No

Go-back-N

Yes

Selective Repeat

What seq # on ACK?

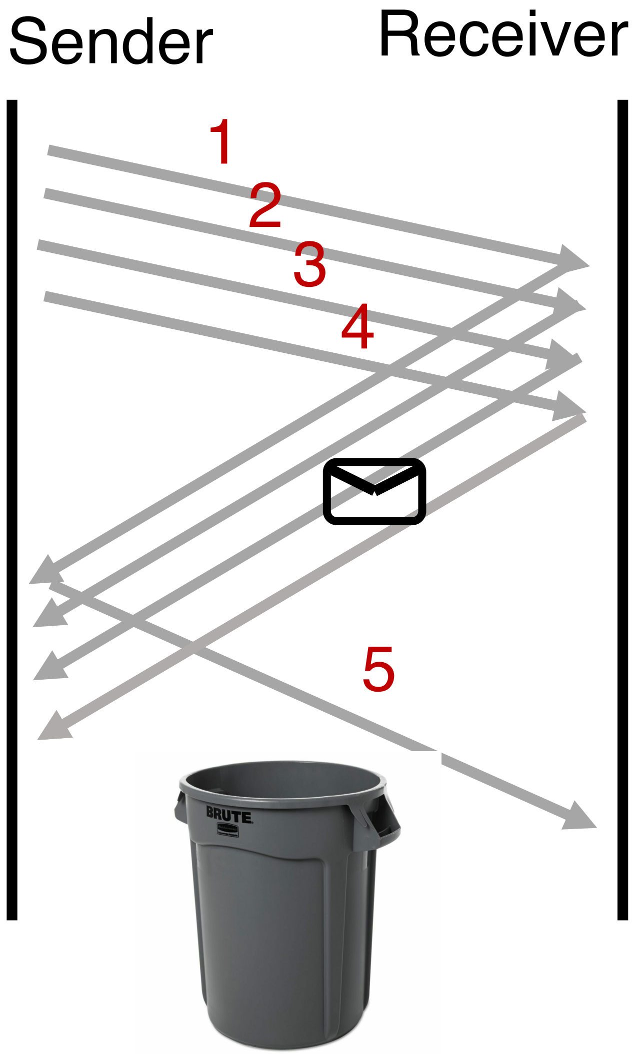
Last pkt in order

Cumulative ACK

Seq # ranges received so far

Selective ACK

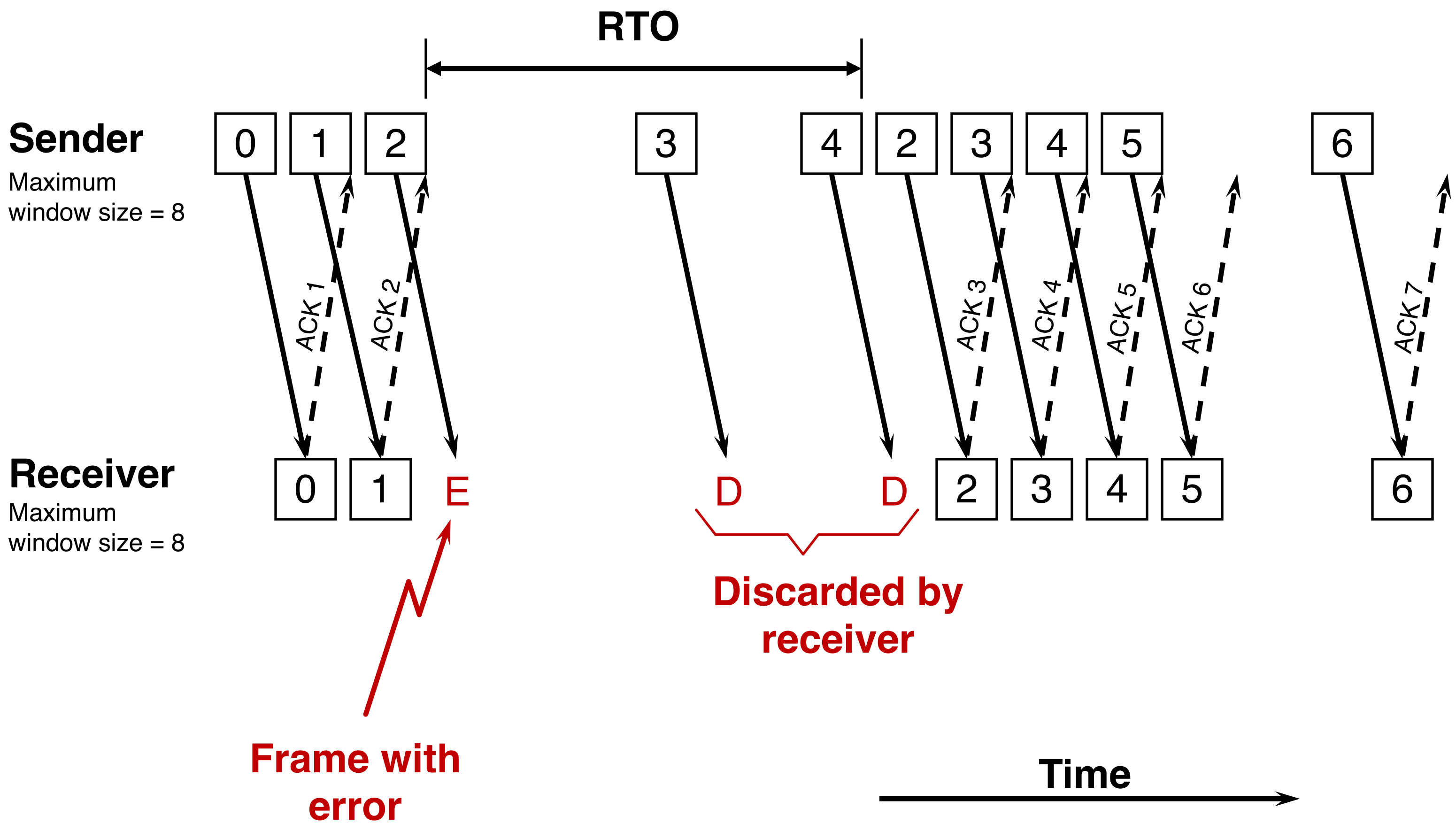
TCP's default



Sliding Window with Go Back N

- When the receiver notices a missing or erroneous frame:
- It simply discards all frames with greater sequence numbers
    - The receiver will send no ACK
- The sender will eventually time out and retransmit all the frames in its sending window

Go back N


RTO
Sender
Maximum
window size = 8
0
1
2
3
4
2
3
4
5
6
ACK 1
ACK 2
ACK 3
ACK 4
ACK 5
ACK 6
ACK 7
Receiver
Maximum
window size = 8
0
1
E
D
D
2
3
4
5
6
Discarded by
receiver
Frame with
error
Time

Go back N

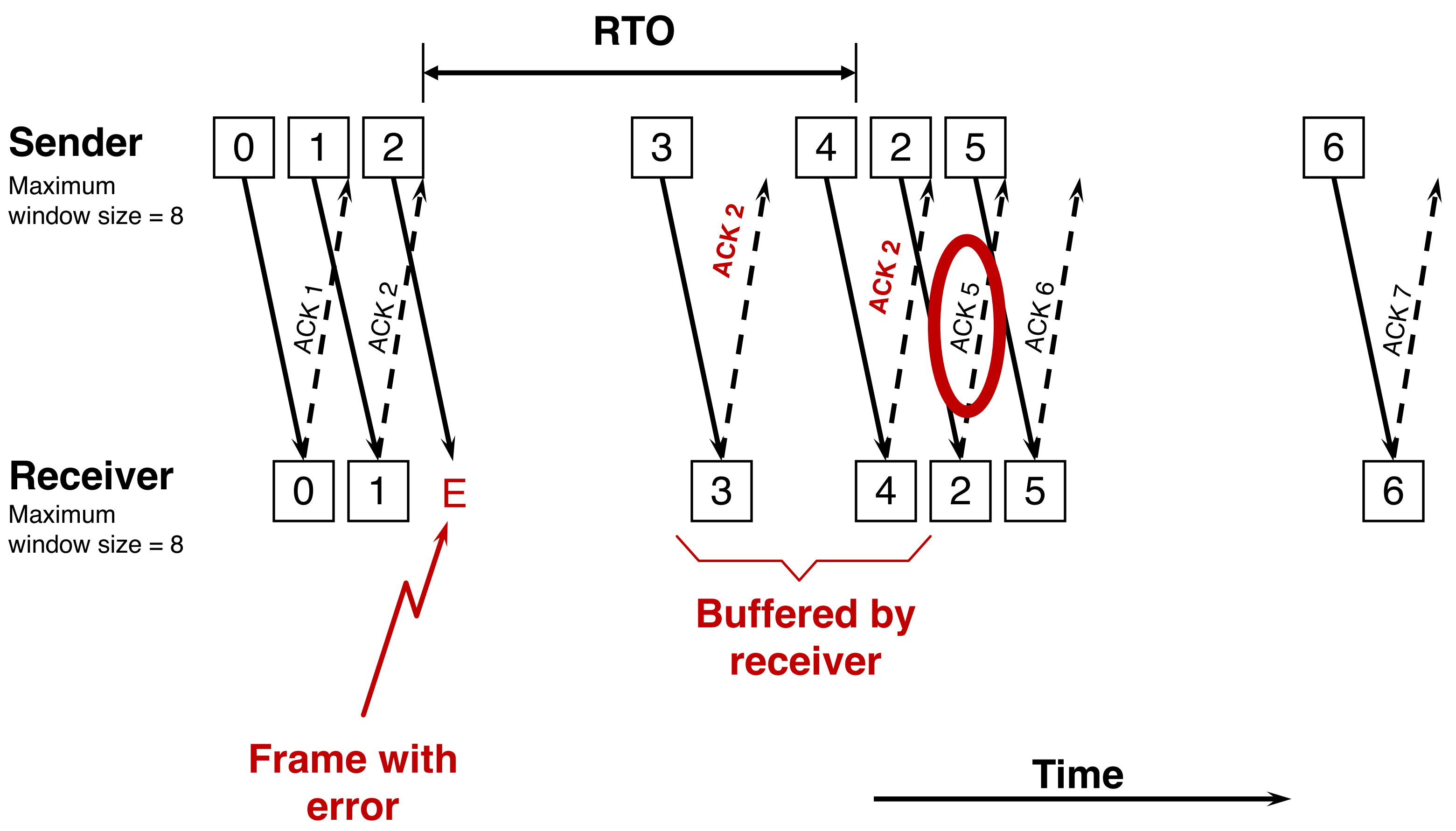
- Go Back N can recover from erroneous or missing frames.
- But it is wasteful.
- If there are errors, the sender will spend time and network bandwidth retransmitting data the receiver has already seen.

Selective repeat with cumulative ACK

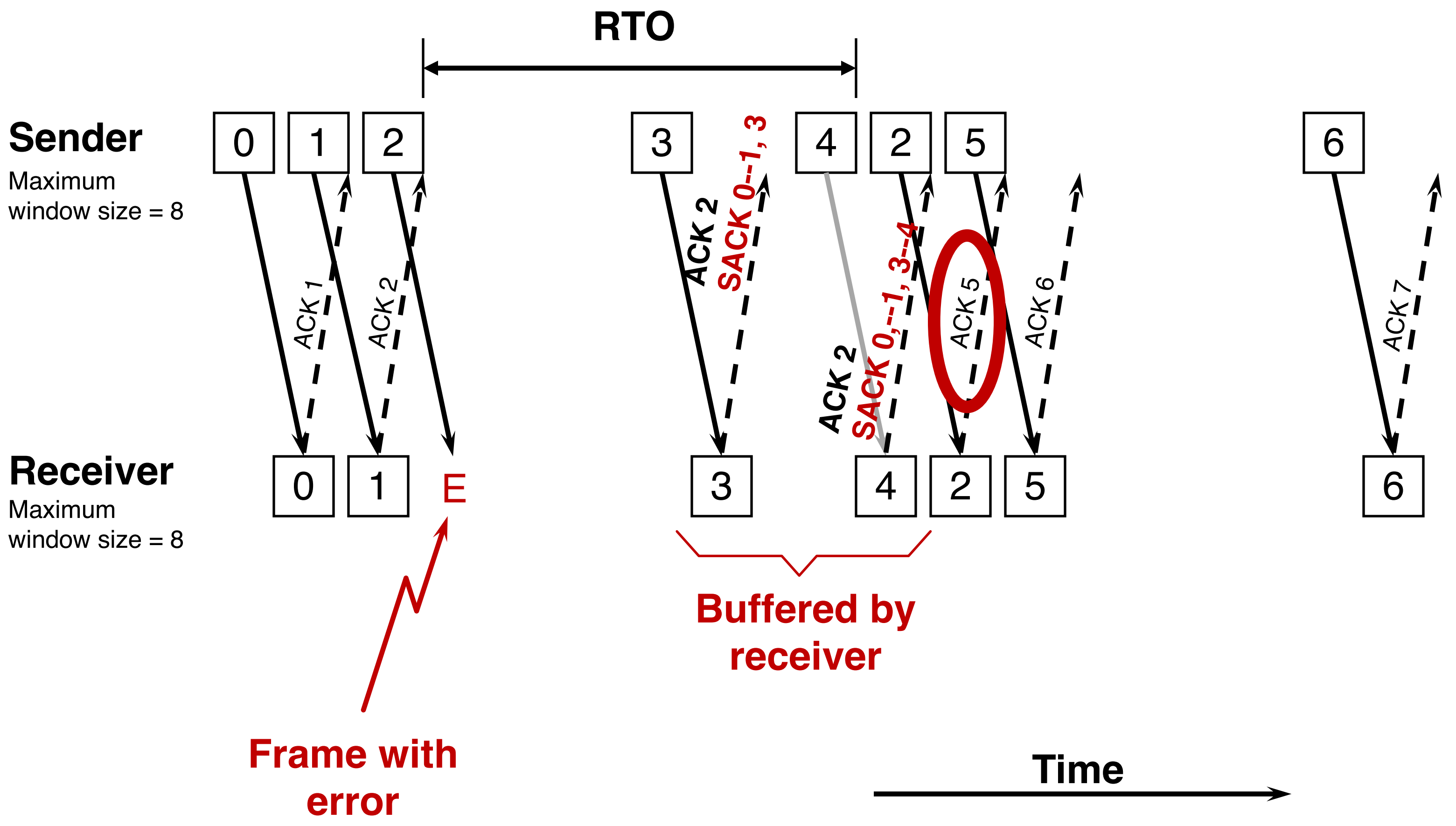
Idea: sender should only retransmit dropped/corrupted segments.

- The receiver stores all the correct frames that arrive following the bad one. (Note that the receiver requires a memory buffer for each sequence number in its receiver window.)
- When the receiver notices a skipped sequence number, it keeps acknowledging the first in-order sequence number it wants to receive. This is what we mean by cumulative ACK.
- When the sender times out waiting for an acknowledgement, it just retransmits the first unacknowledged packet, not all its successors.
- Note that the RTO applies independently to each sequence #

Selective repeat with cumulative ACK

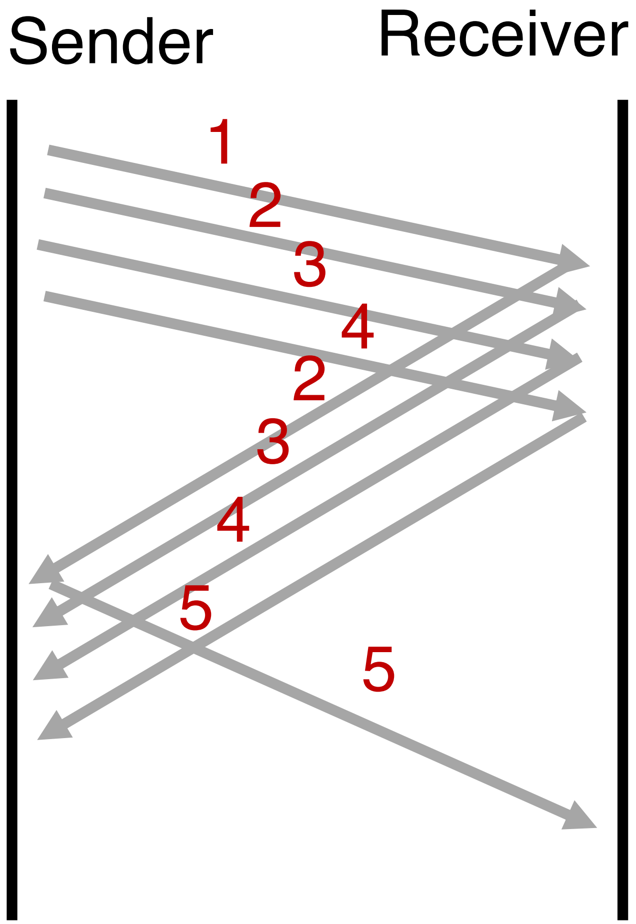


Selective repeat with selective ACK



TCP: Cumulative & Selective ACKs

- Sender retransmits the seq #s it thinks aren't received successfully yet
- Pros & cons: selective vs. cumulative ACKs
  - Precision of info available to sender
  - Redundancy of retransmissions
  - Packet header space
  - Complexity (and bugs) in transport software
- On modern OSes, TCP uses selective ACKs by default



Memory Buffers in the Transport Layer

Receiver-side sockets need memory buffers

- Since TCP uses selective repeat, the receiver must buffer data that is received out of order:
  - e.g., hold packets so that only the “holes” (due to drops) need to be filled in later, without having to retransmit packets that were received successfully

- Apps read from the receive-side socket buffer when you do a `recv()` call.

- Even if data reliably received in order, applications may not always read the data immediately
  - What if you invoked recv() in your socket program infrequently (or never)?
  - For the same reason, UDP sockets also have buffers

Sender-side sockets need memory buffers

- The possibility of packet retransmission in the future means that data can’t be immediately discarded from the sender once transmitted.

- Transport layer must wait for ACK of a piece of data before reclaiming the memory for that data.

Q2. How much data to keep in flight?

Q2: How much data to keep in flight?

- Challenging question! We want to increase throughput. But:
- The receiving app must keep up: otherwise, receiver socket buffer will fill up. Once full, subsequent packets are dropped.
- Even if receiving app is fast, there must be sufficient buffering for selective repeat, if some data is dropped/corrupted
- The network path must be able to keep up.
- We don't want window to be so large that pkts dropped anyway
- Challenge: The sender must figure out where the bottleneck is: receiving app? Socket buffer? A link along the network path?
- Flow control and congestion control

Inspecting TCP stack parameters

- A small demo

Info on (tuning) TCP stack parameters

- https://www.ibm.com/support/knowledgecenter/linuxonibm/liaag/wkvm/wkvm_c_tune_tcpip.htm

- https://cloud.google.com/solutions/tcp-optimization-for-network-performance-in-gcp-and-hybrid