


Computer Networks

CMSC 417 : Spring 2024

COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

Topic: Transport Layer Protocols (TCP)
(Textbook chapter 5)

Nirupam Roy

Tu-Th 2:00-3:15pm
CSI 2117

March 12th, 2024



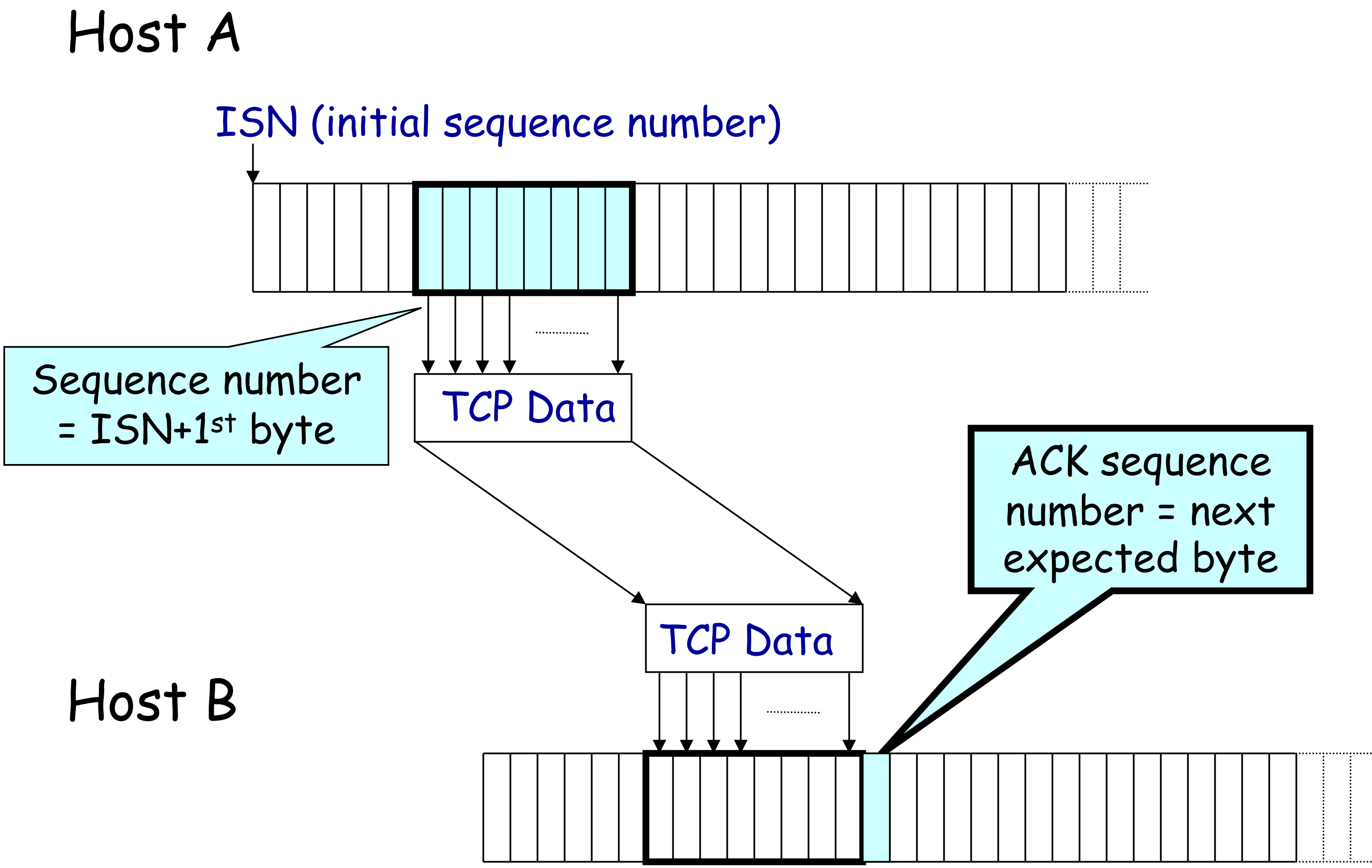

Retransmission hints

- Acknowledgments from receiver
  - Positive: "okay" or "uh huh" or "ACK"
  - Negative: "please repeat that" or "NACK"
- Retransmission by the sender
  - After *not* receiving an "ACK"
  - After receiving a "NACK"
- Timeout by the sender ("stop and wait")
  - Don't wait forever without some acknowledgment

TCP Support for Reliable Delivery

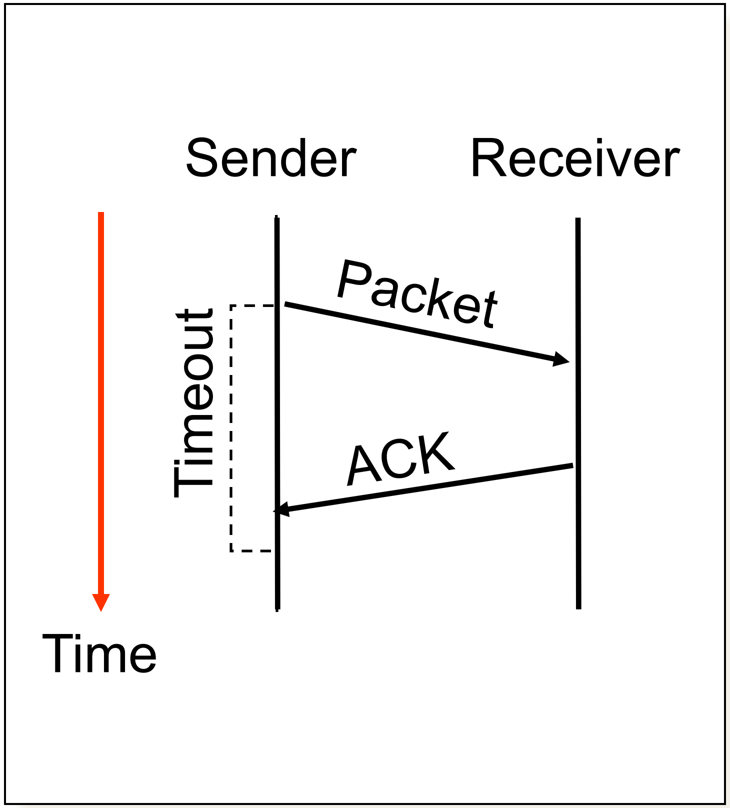
- **Detect bit errors:** checksum
  - Used to detect corrupted data at the receiver
  - …leading the receiver to drop the packet
- **Detect missing data:** sequence number
  - Used to detect a gap in the stream of bytes
  - ... and for putting the data back in order
- **Recover from lost data:** retransmission
  - Sender retransmits lost or corrupted data
  - Two main ways to detect lost packets

TCP Acknowledgments



Automatic Repeat reQuest (ARQ)

- ACK and timeouts
  - Receiver sends ACK when it receives packet
  - Sender waits for ACK and times out
- Simplest ARQ protocol
  - Stop and wait
  - Send a packet, stop and wait until ACK arrives

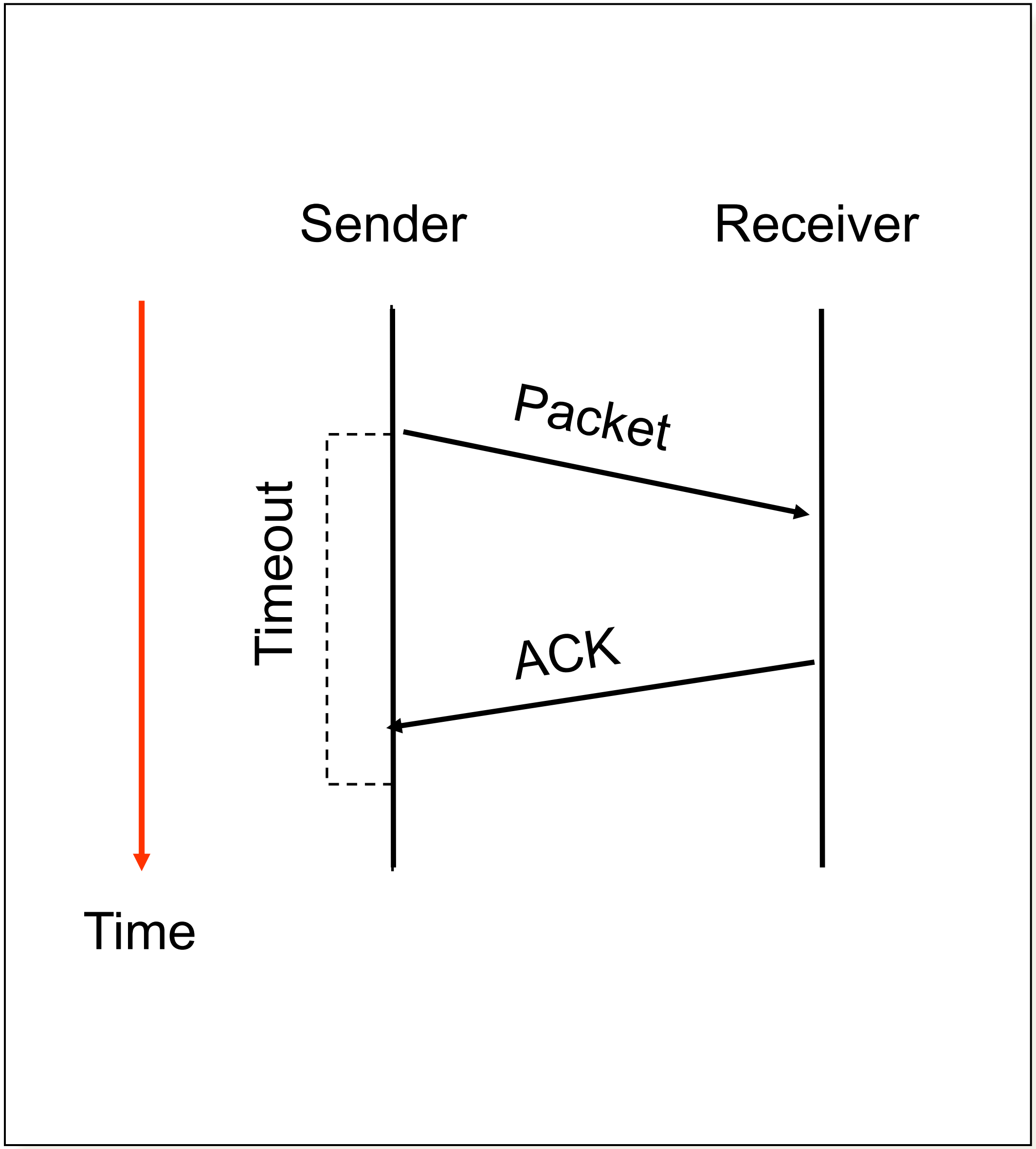


Automatic Repeat reQuest (ARQ)

- We will discuss two variations
    1. Stop and Wait
    2. Sliding window

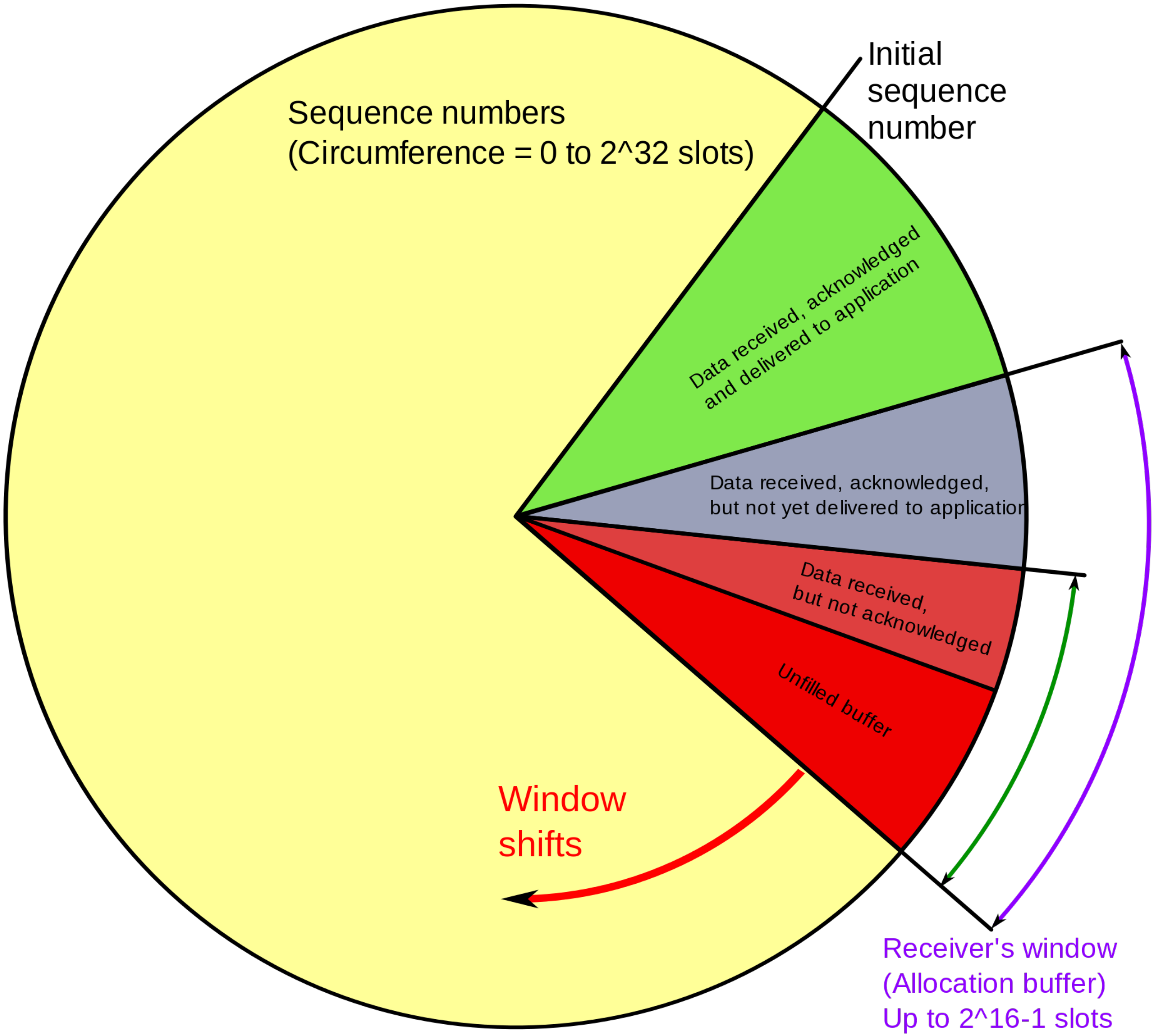
Reliable Delivery on a Lossy Channel With Bit Errors

Stop-and-Wait Protocol


Sender
Receiver
Packet
Timeout
ACK
Time

- A few design decisions
    1. Unique packet identifiers: Why and how
    2. Duplicate packets: Why and how to eliminate
    3. Timeout: picking the ideal value

- A few design decisions
    1. Unique packet identifiers: Why and how
    2. Duplicate packets: Why and how to eliminate
    3. Timeout: picking the ideal value



PC: https://en.wikipedia.org/wiki/Transmission_Control_Protocol

- Wrap around is possible

Network	B*8 bits/sec	B bytes/sec	Twrap secs
ARPANET	56kbps	7KBps	3*10**5 (~3.6 days)
DS1	1.5Mbps	190KBps	10**4 (~3 hours)
Ethernet	10Mbps	1.25MBps	1700 (~30 mins)
DS3	45Mbps	5.6MBps	380
FDDI	100Mbps	12.5MBps	170
Gigabit	1Gbps	125MBps	17

REF: RFC 1323

- Wrap around-aware sequence number comparison

If A and B are sequence numbers,

A < B if 0 < (B - A) < 231,**

computed in unsigned 32-bit arithmetic

- PAWS: Protect Against Wrapped Sequence Numbers (RFC 1323) – Use sequence number and timestamps.

REF: RFC 1323

- A few design decisions
    1. Unique packet identifiers: Why and how
    2. Duplicate packets: Why and how to eliminate
    3. Timeout: picking the ideal value

- A few design decisions

2. Duplicate packets: Why and how to eliminate

- Receiver maintain the last in-sequence byte number in a variable, say rcv_seqnum

- If the Receiver receives a packet with seq number less than or equal to rcv_seqnum

- A few design decisions

2. Duplicate packets: Why and how to eliminate

- Receiver maintain the last in-sequence byte number in a variable, say rcv_seqnum

- If the Receiver receives a packet with seq number less than or equal to rcv_seqnum

>>It sends an ACK for that packet, and discards the packet

- If the Receiver receives a packet with seq number (rcv_seqnum+1):

- A few design decisions

2. Duplicate packets: Why and how to eliminate

- Receiver maintain the last in-sequence byte number in a variable, say rcv_seqnum

- If the Receiver receives a packet with seq number less than or equal to rcv_seqnum

>>It sends an ACK for that packet, and discards the packet

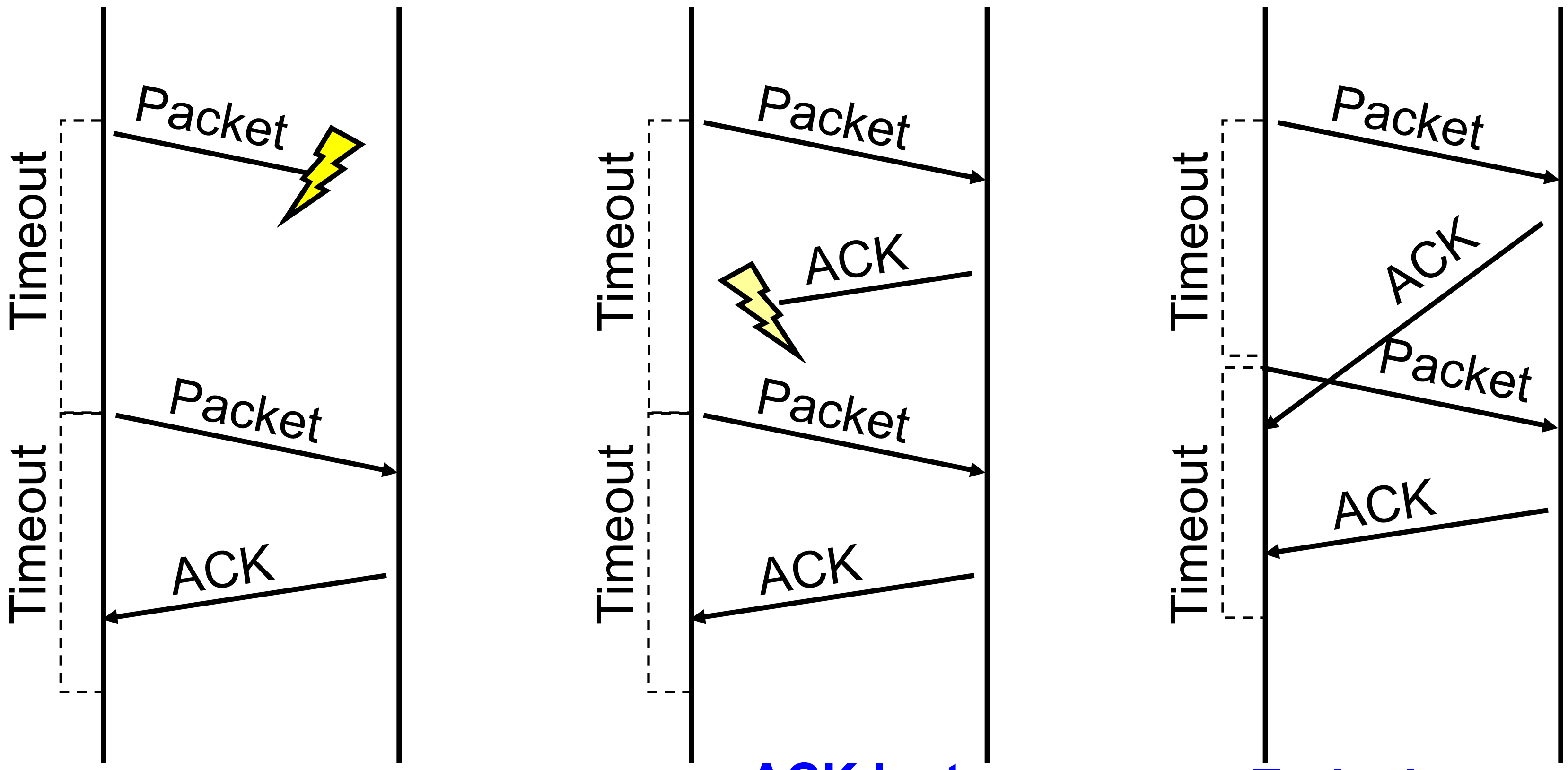
- If the Receiver receives a packet with seq number (rcv_seqnum+1):

>> It sends an ACK for that packet, and includes the packet to the application buffer

- What if the receiver gets a packet with seq number greater than (rcv_seqnum+1) ?

- A few design decisions
    1. Unique packet identifiers: Why and how
    2. Duplicate packets: Why and how to eliminate
    3. Timeout: picking the ideal value

Reasons for Retransmission



Packet lost

ACK lost
DUPLICATE
PACKET

Early timeout
DUPLICATE
PACKETS

How Long Should Sender Wait?

- Sender sets a timeout to wait for an ACK
  - Too short:
    - >> wasted retransmissions
  - Too long:
    - >> excessive delays when packet lost

How Long Should Sender Wait?

- Sender sets a timeout to wait for an ACK
  - Too short:
    - >> wasted retransmissions
  - Too long:
    - >> excessive delays when packet lost
- TCP sets timeout as a function of the RTT
  - Expect ACK to arrive after a “round-trip time”
  - … plus a fudge factor to account for queuing
- But, how does the sender know the RTT?

How Long Should Sender Wait?

- Sender sets a timeout to wait for an ACK
  - Too short:
    - >> wasted retransmissions
  - Too long:
    - >> excessive delays when packet lost
- TCP sets timeout as a function of the RTT
  - Expect ACK to arrive after a "round-trip time"
  - … plus a fudge factor to account for queuing
- But, how does the sender know the RTT?
  - Running average of delay to receive an ACK

TCP timestamp

```
+--------+--------+----------------------+----------------------+
|Kind=8 |  10    |   TS Value (TSval)   |TS Echo Reply (TSecr)|
+--------+--------+----------------------+----------------------+
    1        1               4                      4
```

TCP timestamp

Sender's current time

```
+--------+--------+---------------------+---------------------+
|Kind=8 |   10   |   TS Value (TSval)  |TS Echo Reply (TSecr)|
+--------+--------+---------------------+---------------------+
    1        1              4                     4
```

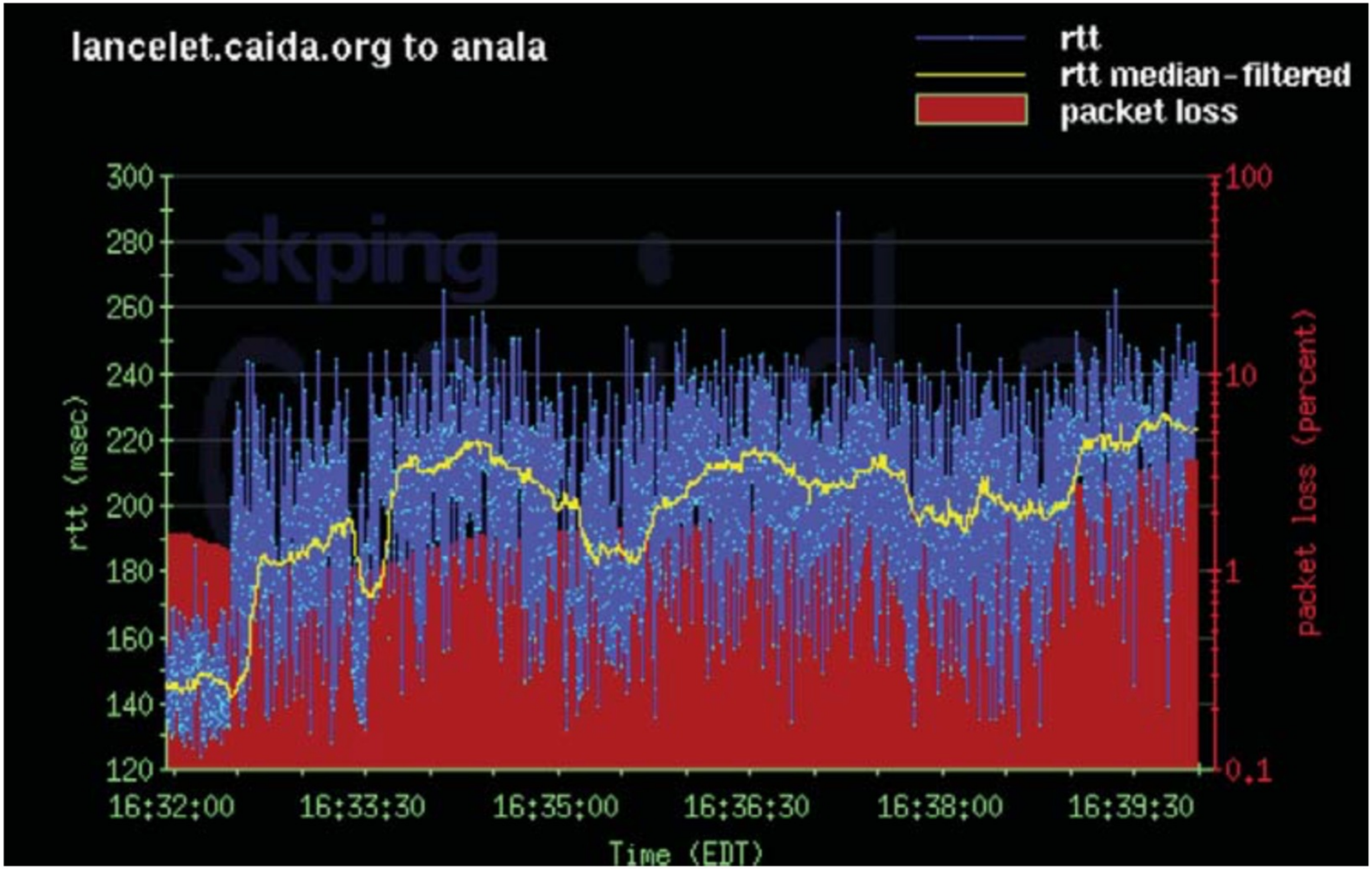
TCP Timestamp Option (10 bytes)

Sender’s current time

```
+--------+--------+---------------------+---------------------+
|Kind=8  |   10   |    TS Value (TSval) |TS Echo Reply (TSecr)|
+--------+--------+---------------------+---------------------+
     1        1               4                     4
```

Copied timestamp from the packet being acknowledged.
(valid only when the packet contains an ACK. i.e., the ACK bit is set in TCP header).

Example RTT Estimation

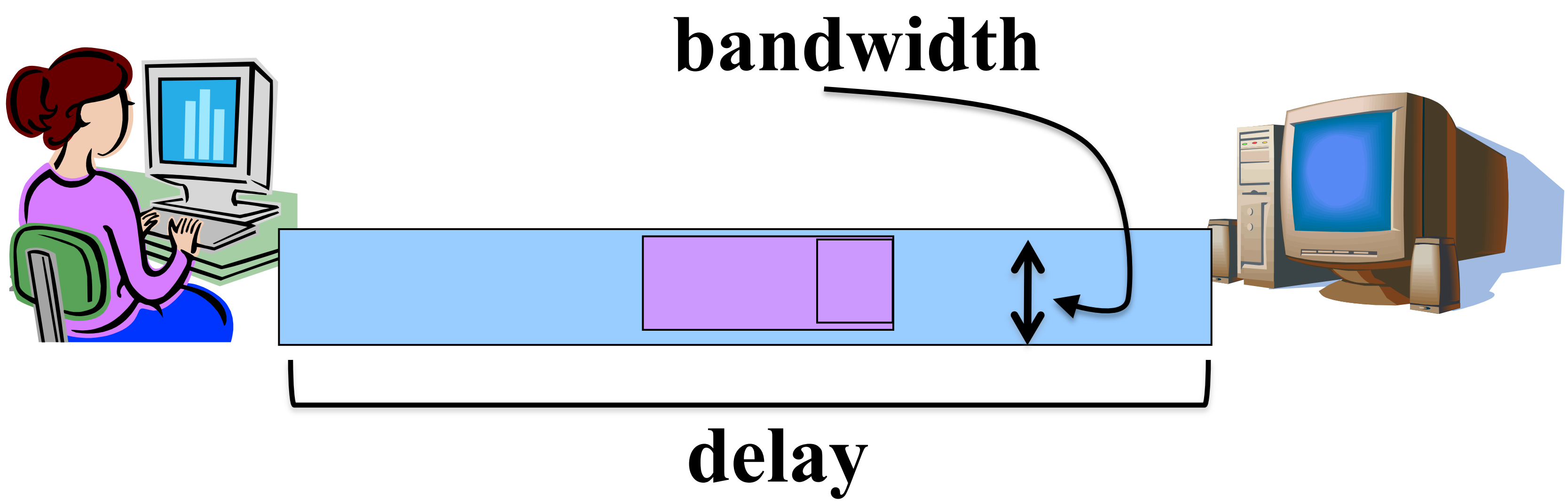


Reliable Delivery on a Lossy Channel With Bit Errors

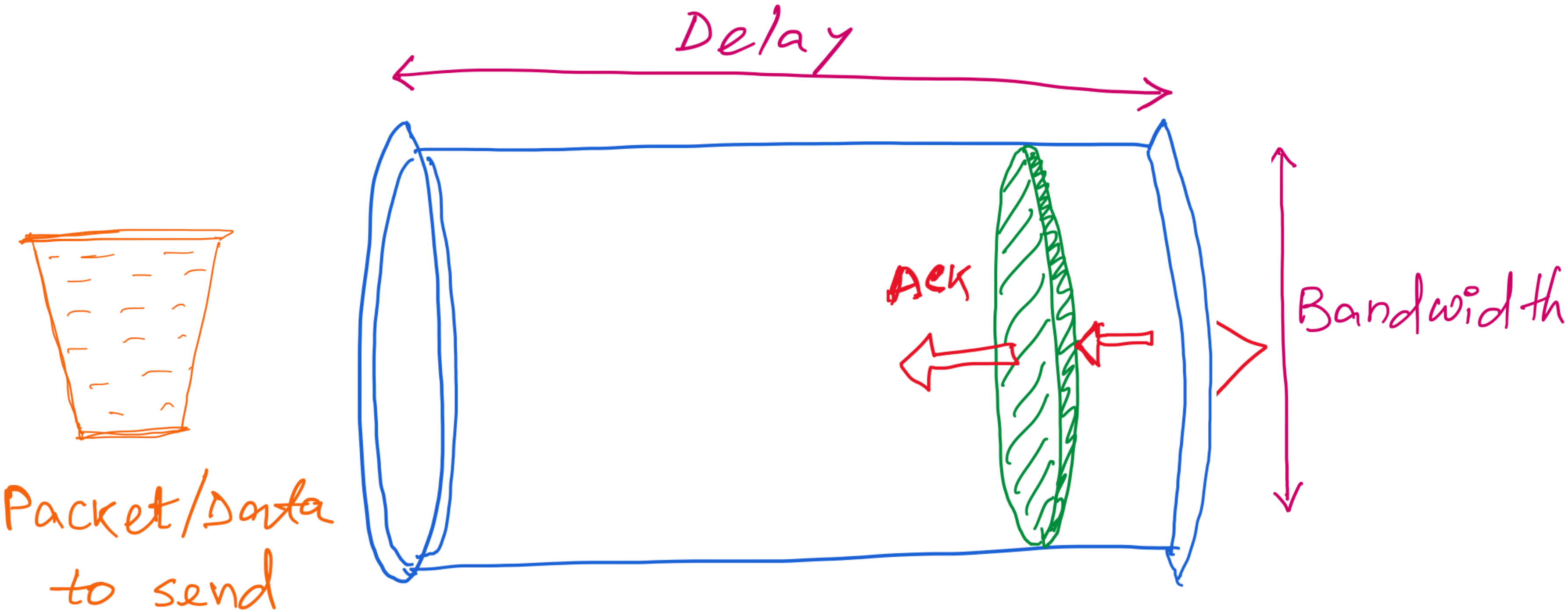
TCP Flow control: Sliding Window Protocol

Motivation for Sliding Window

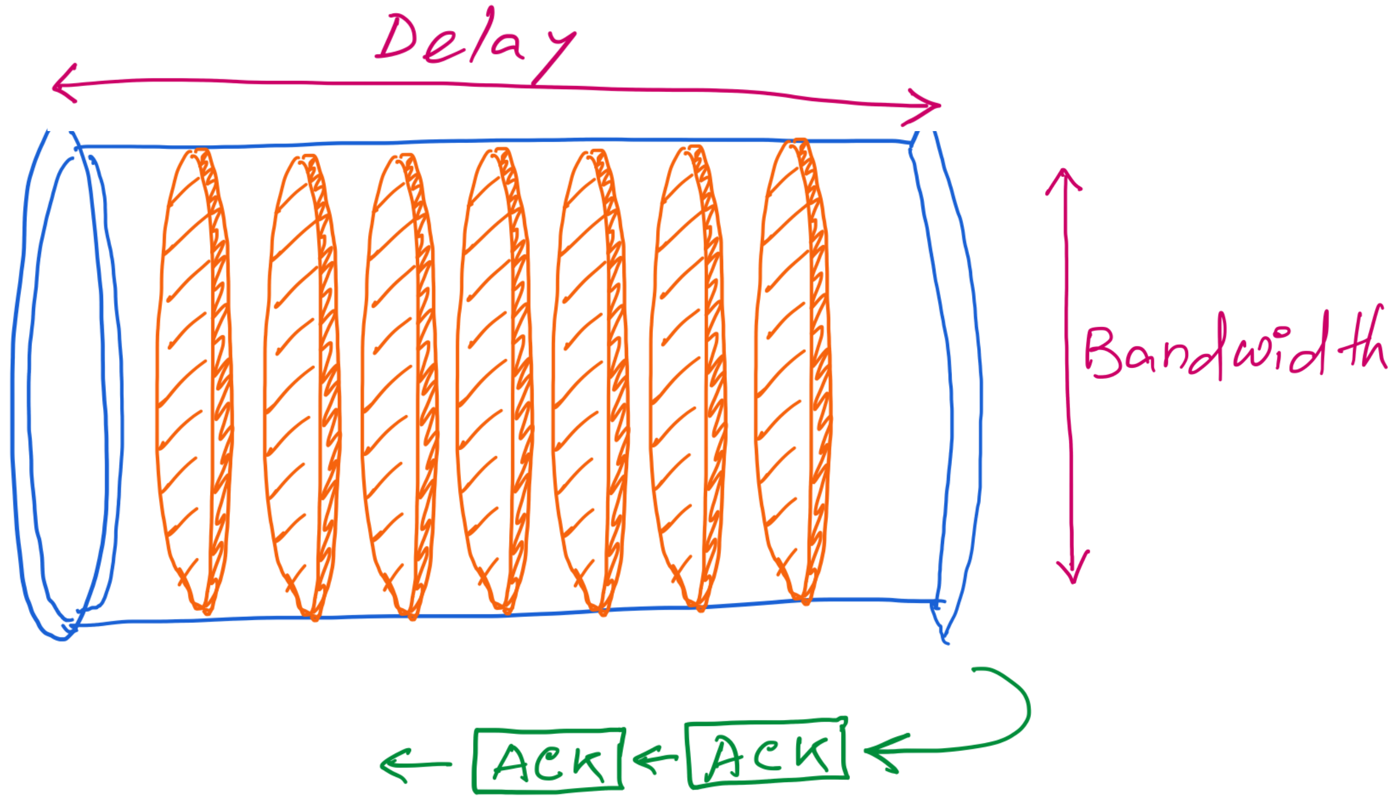
- Stop-and-wait is inefficient
  - Only one TCP segment is "in flight" at a time
  - Especially bad for high "delay-bandwidth product"



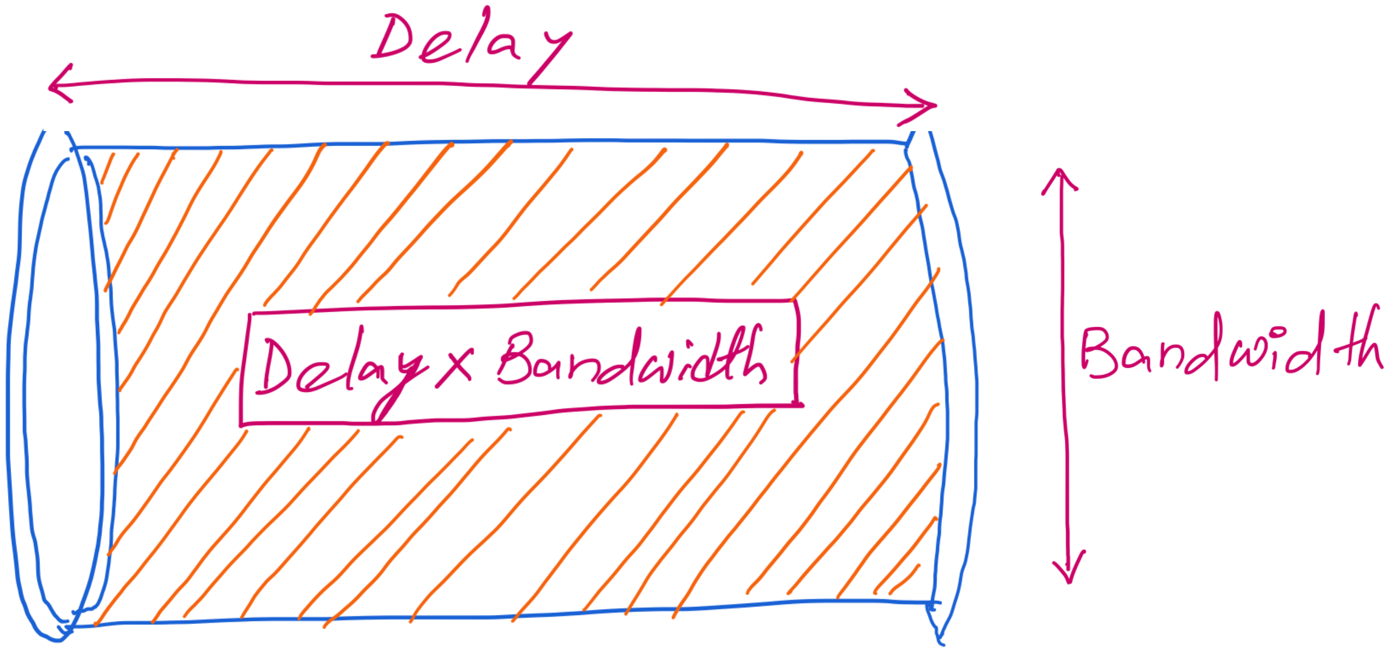
Revisiting “delay X bandwidth”



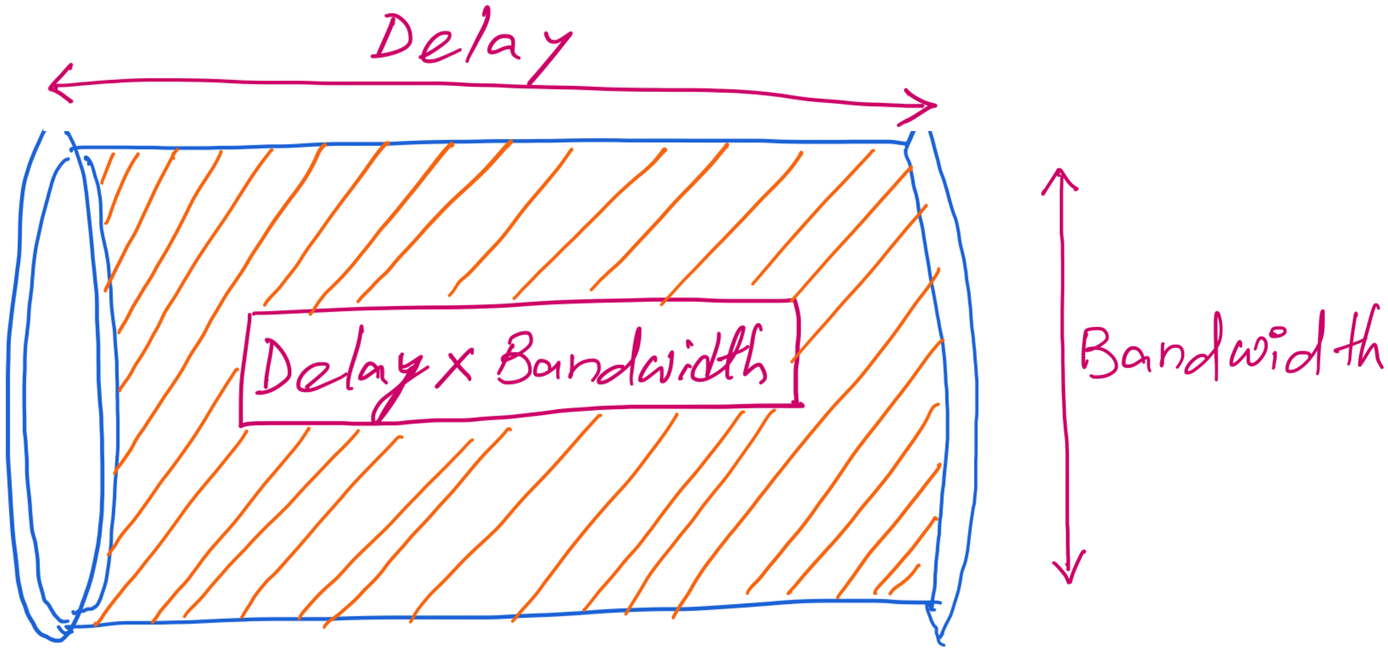
Revisiting “delay X bandwidth”



Revisiting “delay X bandwidth”

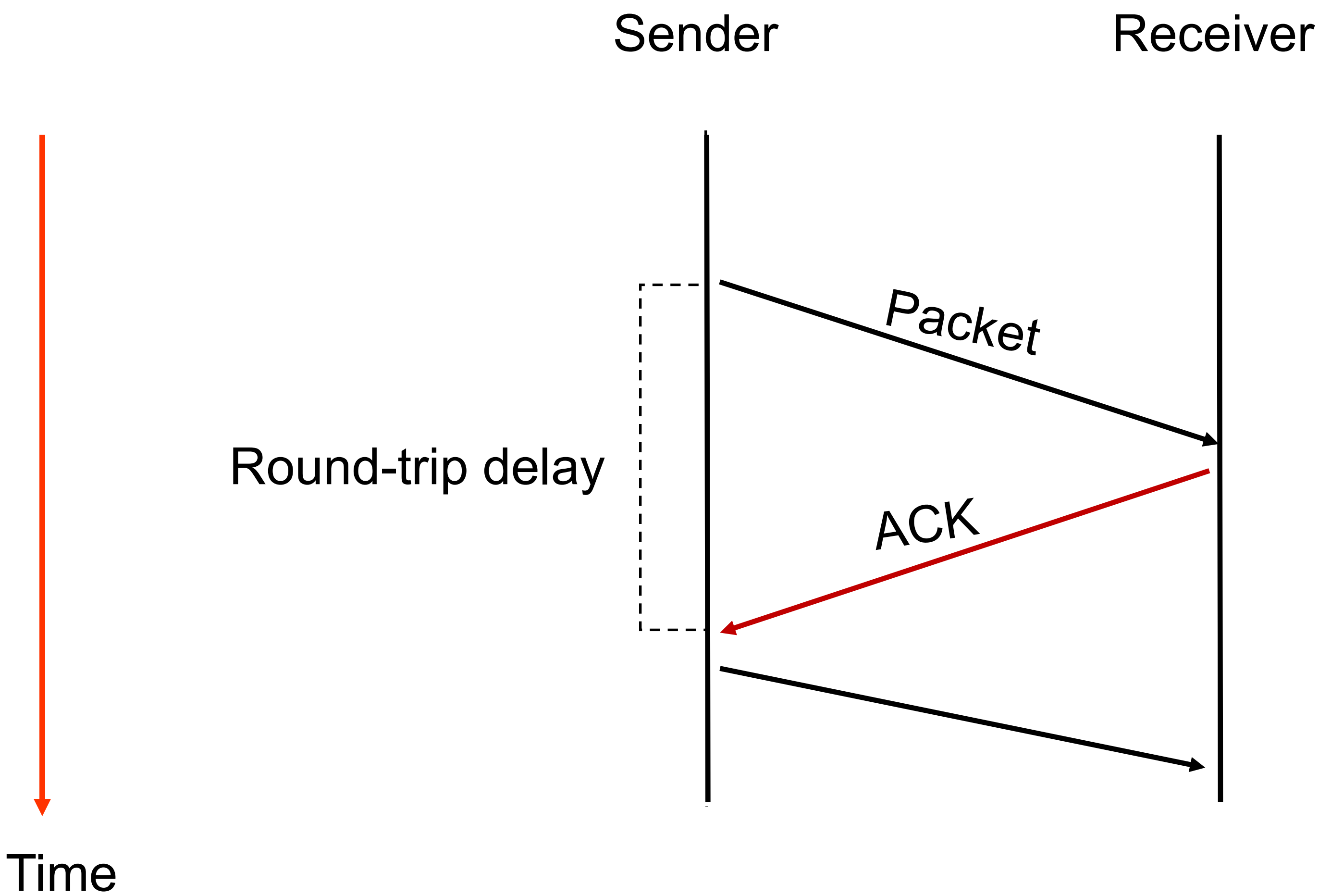


Revisiting “delay X bandwidth”

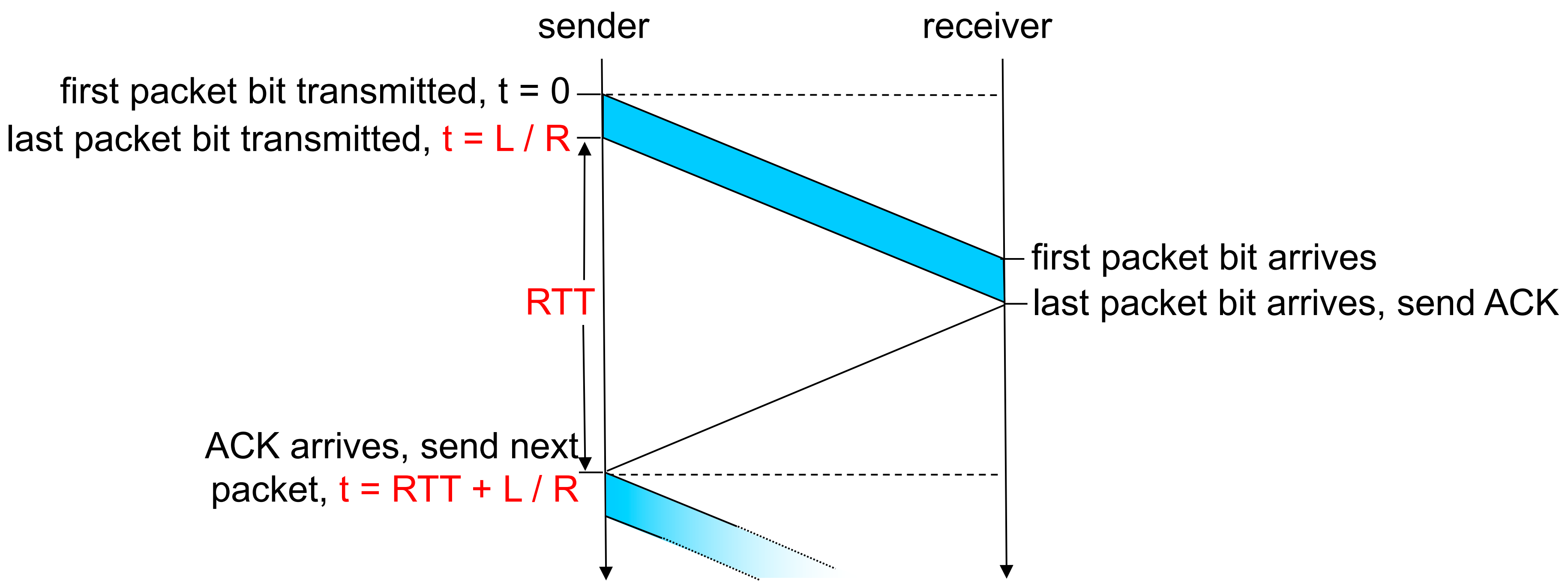


Consider two-way delay
or round-trip delay

Revisiting “delay X bandwidth”



stop-and-wait operation



$$U_{sender} = \frac{L / R}{RTT + L / R} = \frac{.008}{30.008} = 0.00027$$

Packet size = L bits

Network bandwidth = R bits-per-sec

Performance – an implementation of stop-and-wait

- Stop-and-wait works, but performance stinks
- ex: 1 Gbps link, 15 ms prop. delay, 8kbit (8000 bit) packet:

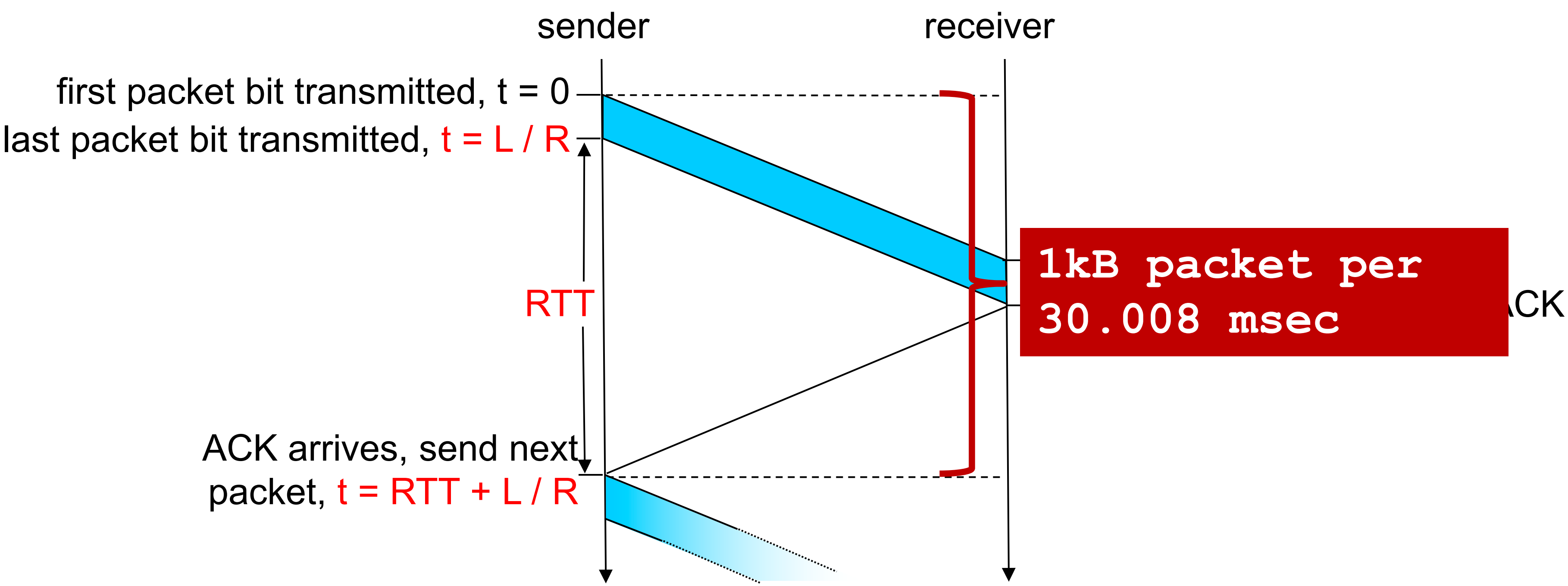
$$d_{trans} = \frac{L}{R} = \frac{8000\text{bits}}{10^9\text{ bps}} = 8\text{ microseconds}$$

  - U_{sender}: utilization – fraction of time sender busy sending

$$U_{sender} = \frac{L / R}{RTT + L / R} = \frac{.008}{30.008} = 0.00027$$

  - 1KB pkt every 30 msec -> 33kB/sec thruput over 1 Gbps link
  - network protocol limits use of physical resources!

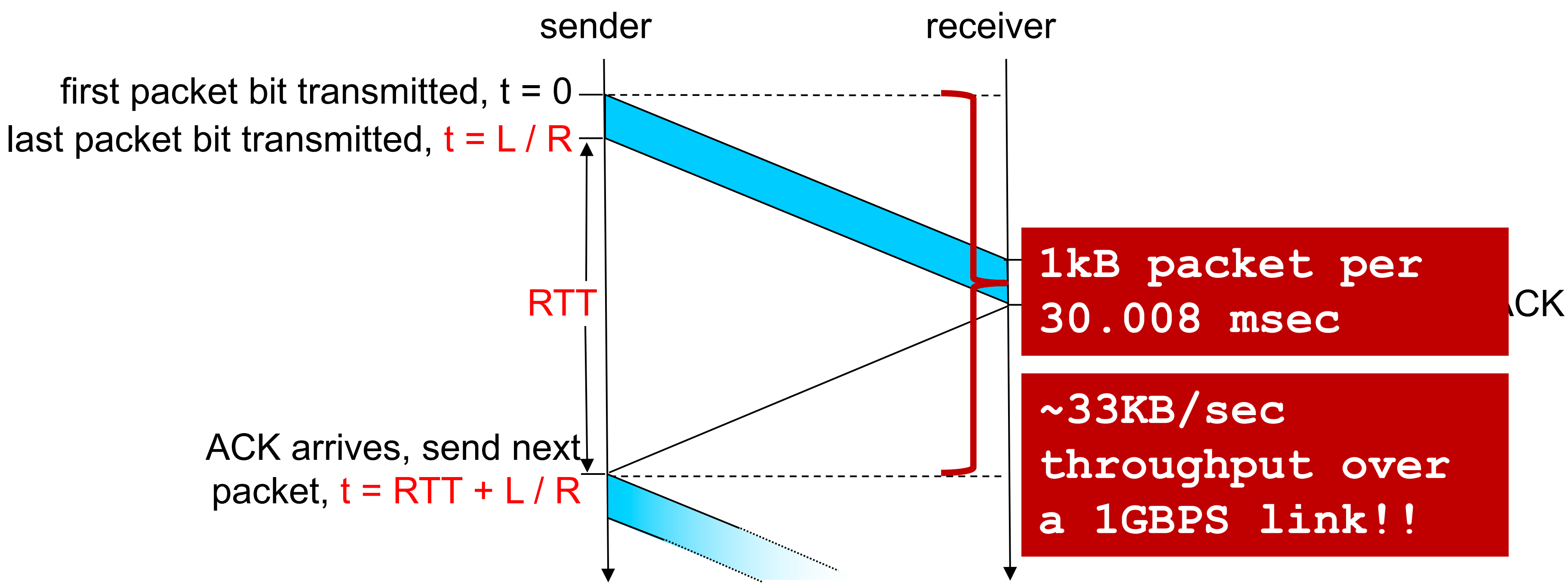
stop-and-wait operation



$$U_{sender} = \frac{L / R}{RTT + L / R} = \frac{.008}{30.008} = 0.00027$$

Packet size = L bits
Network bandwidth = R bits-per-sec

stop-and-wait operation

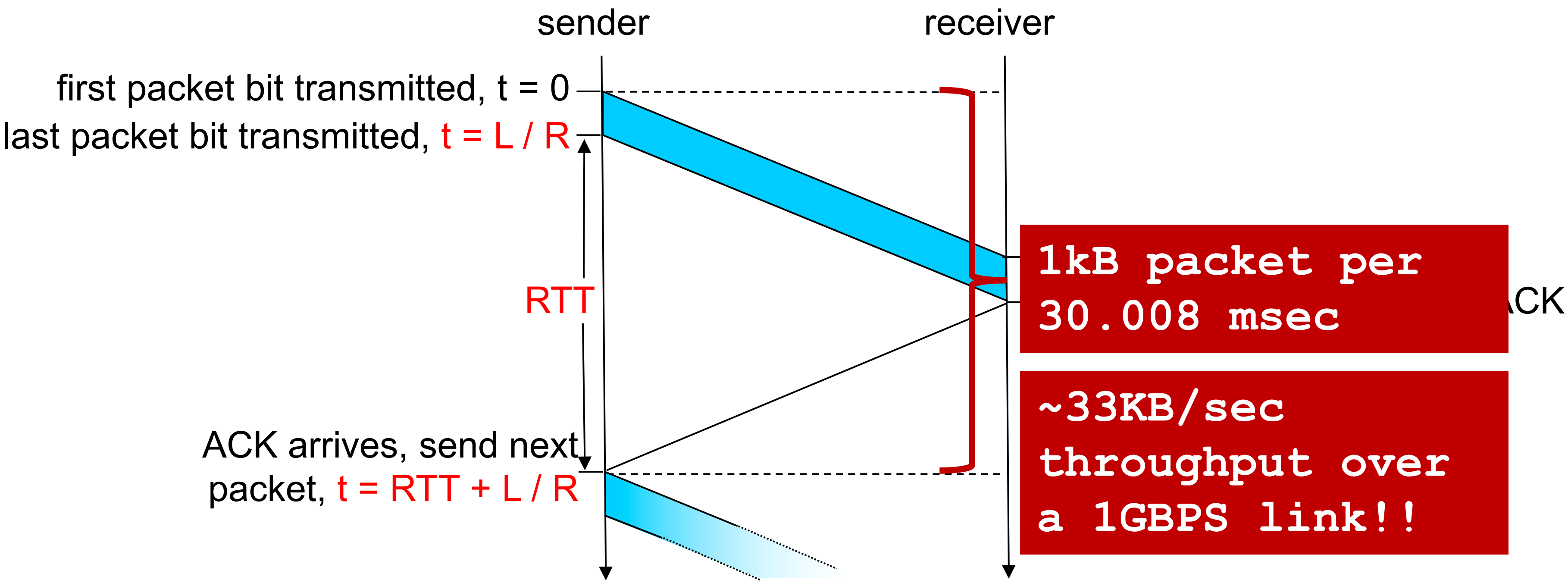


$$U_{\text{sender}} = \frac{L / R}{RTT + L / R} = \frac{.008}{30.008} = 0.00027$$

Packet size = L bits

Network bandwidth = R bits-per-sec

stop-and-wait operation



Network protocol limits the full utilization of physical resources.

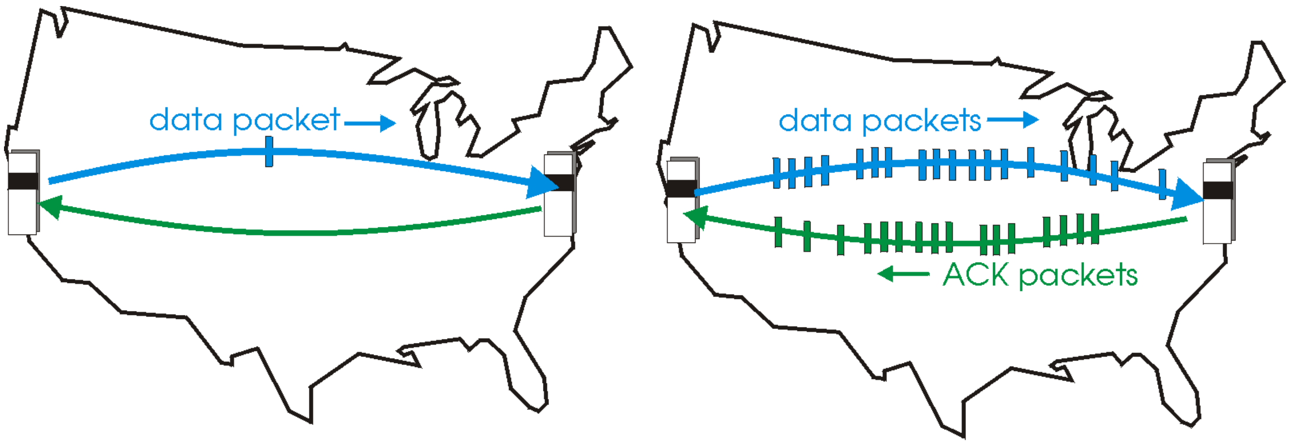
Transport-layer vs Link-layer capacity gap

Network bandwidth = R bits-per-sec

Pipelined protocols

Pipelining: sender allows multiple, "in-flight", yet-to-be-acknowledged pkts

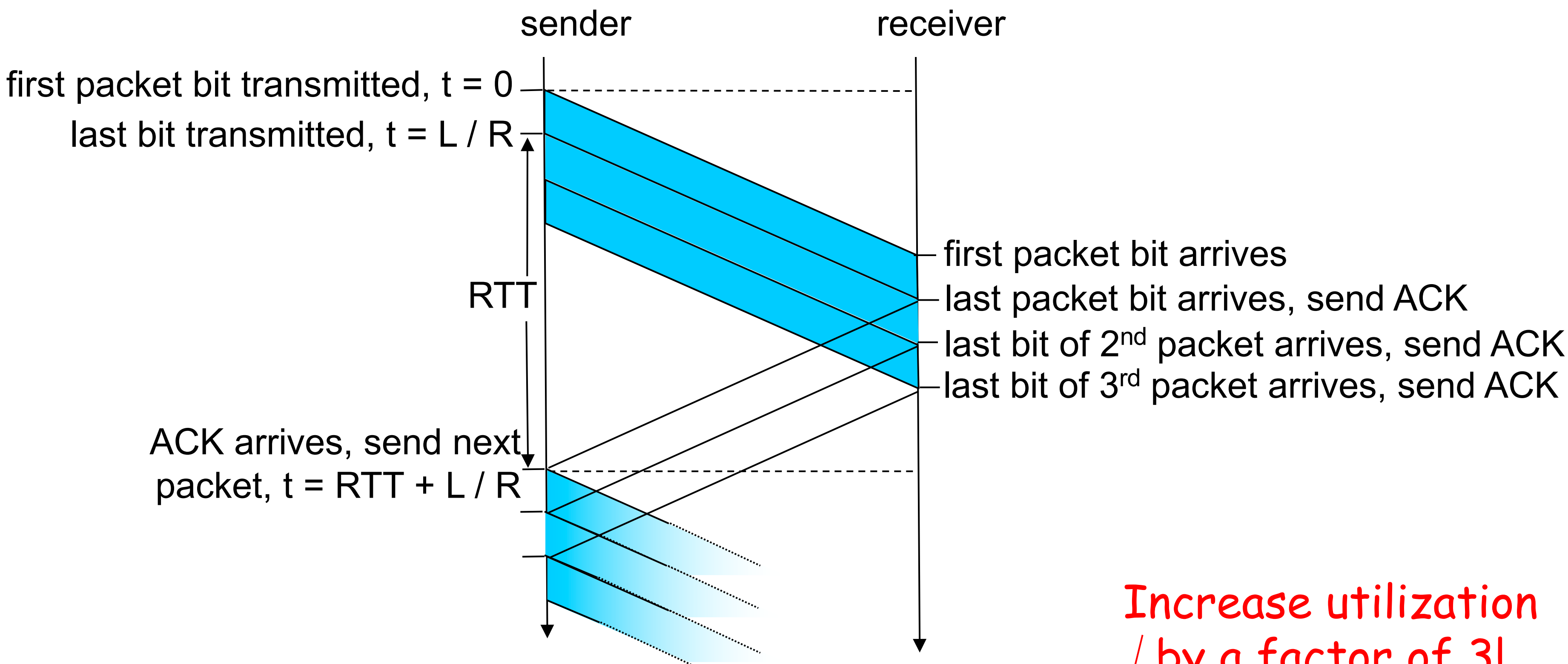
- range of sequence numbers must be increased
- buffering at sender and/or receiver



(a) a stop-and-wait protocol in operation (b) a pipelined protocol in operation

- Two generic forms of pipelined protocols (or sliding window protocol) depending on the retransmission strategy: *go-Back-N, selective repeat*

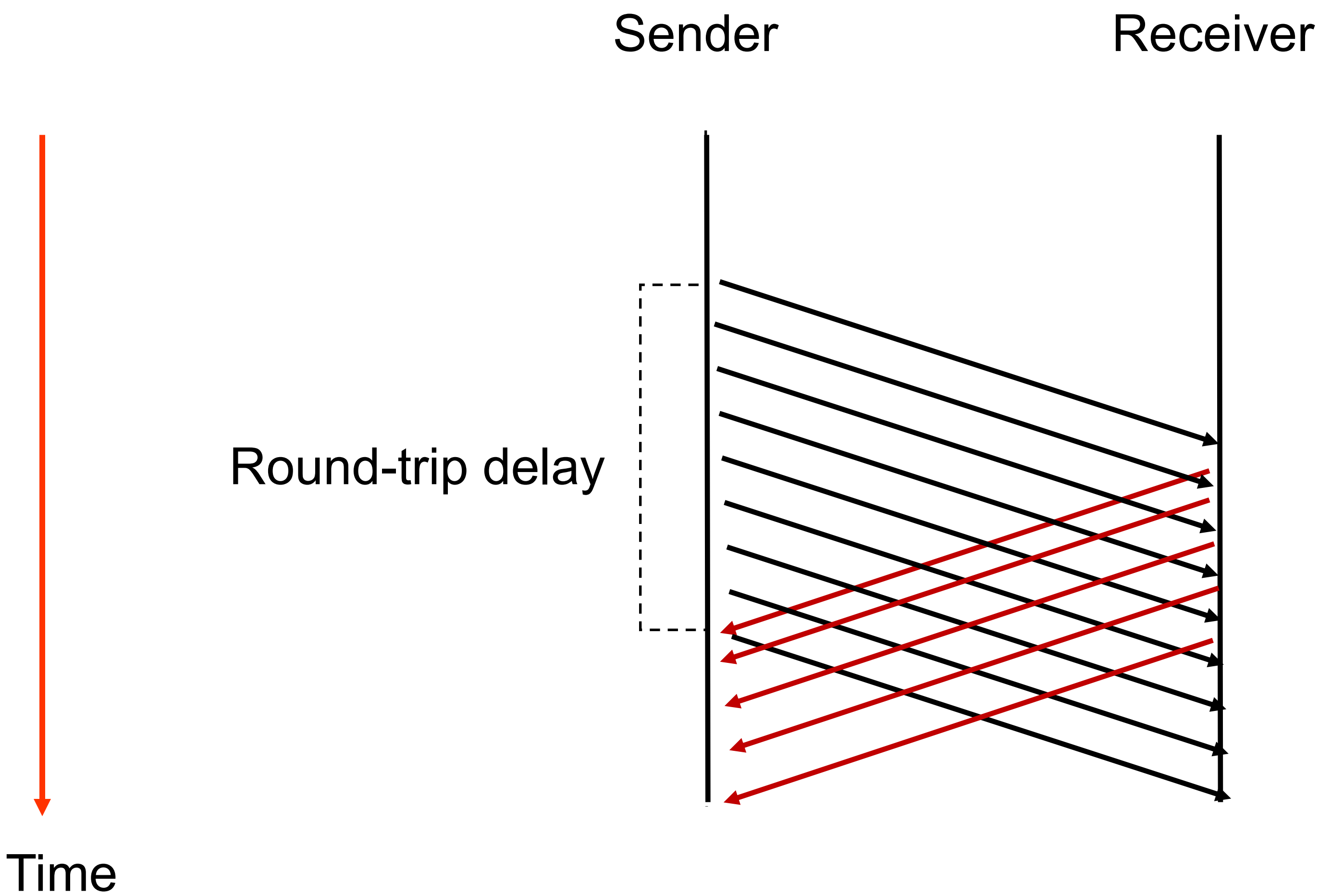
Pipelining: increased utilization



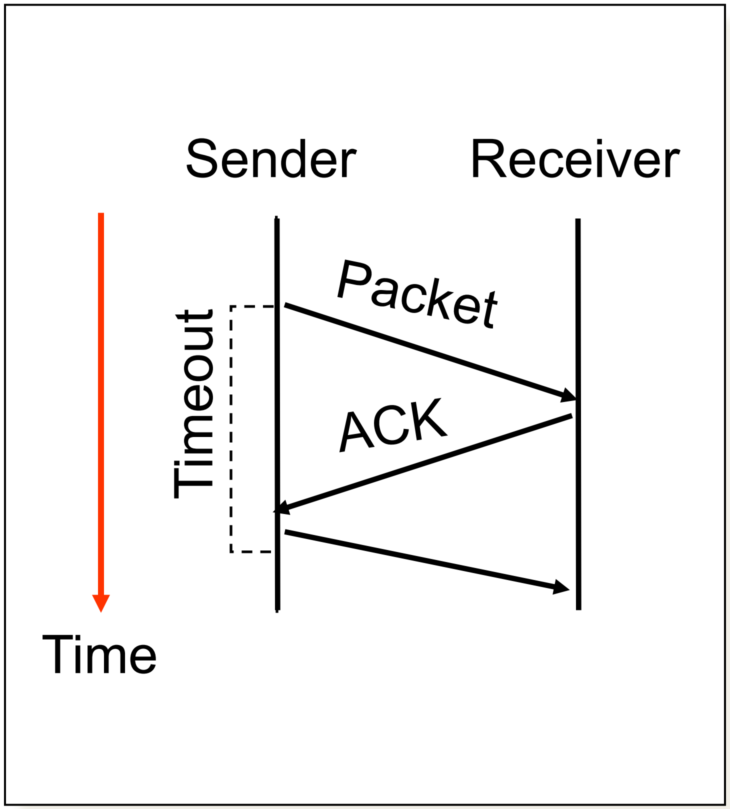
Increase utilization by a factor of 3!

$$U_{sender} = \frac{3 * L / R}{RTT + L / R} = \frac{.024}{30.008} = 0.0008$$

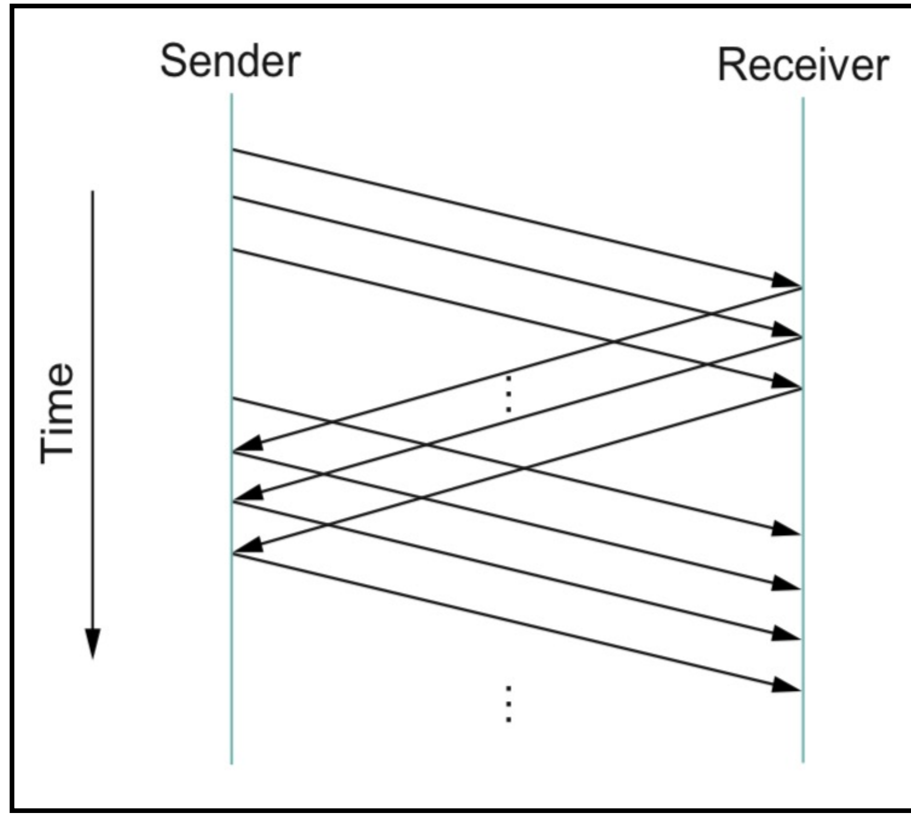
Revisiting "delay X bandwidth"



Stop-and-wait

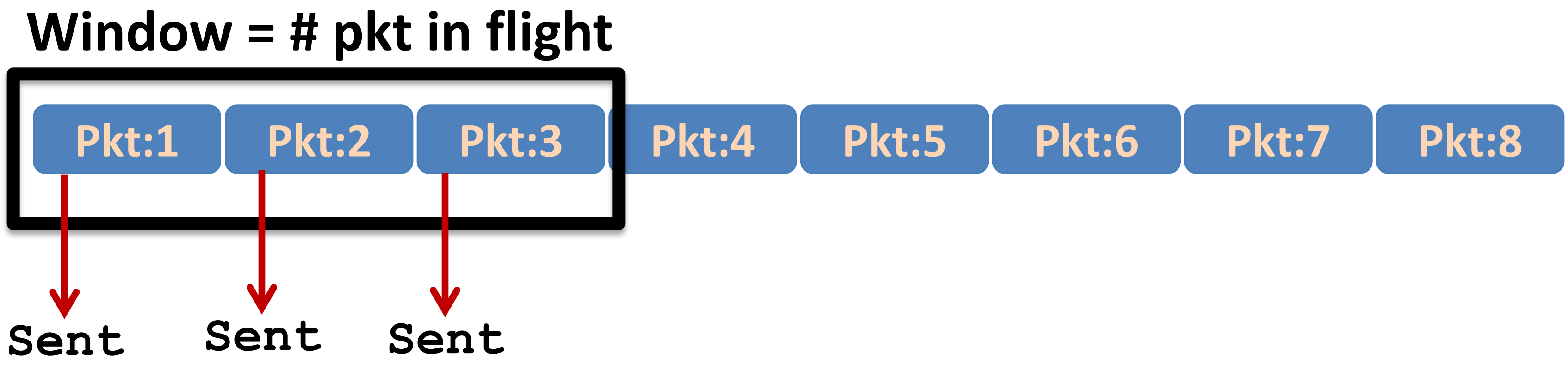


Sliding Window

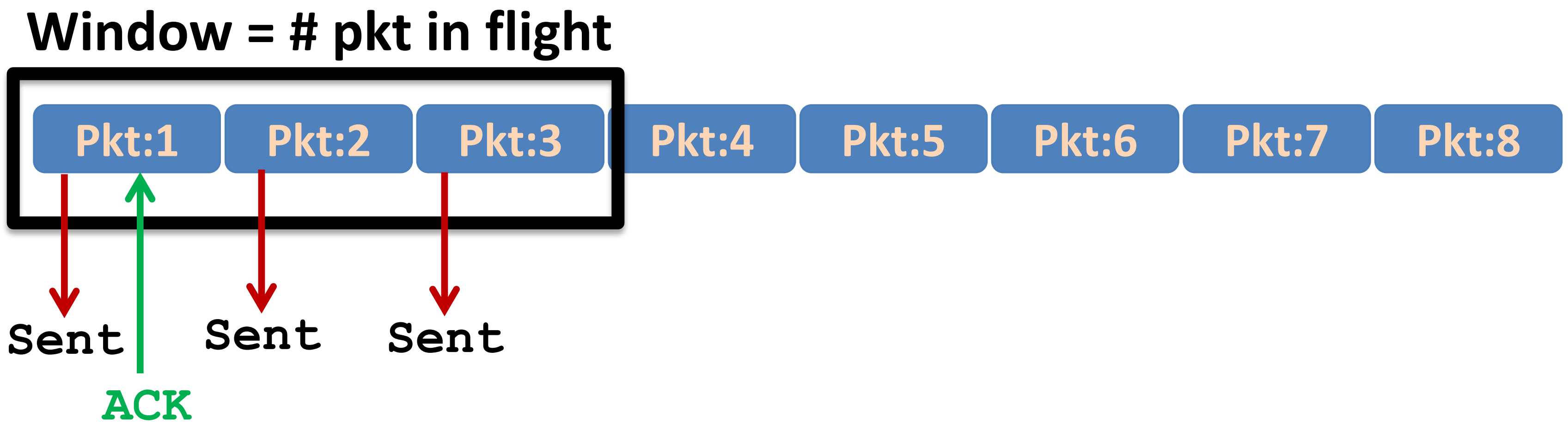


Sliding Window Protocol

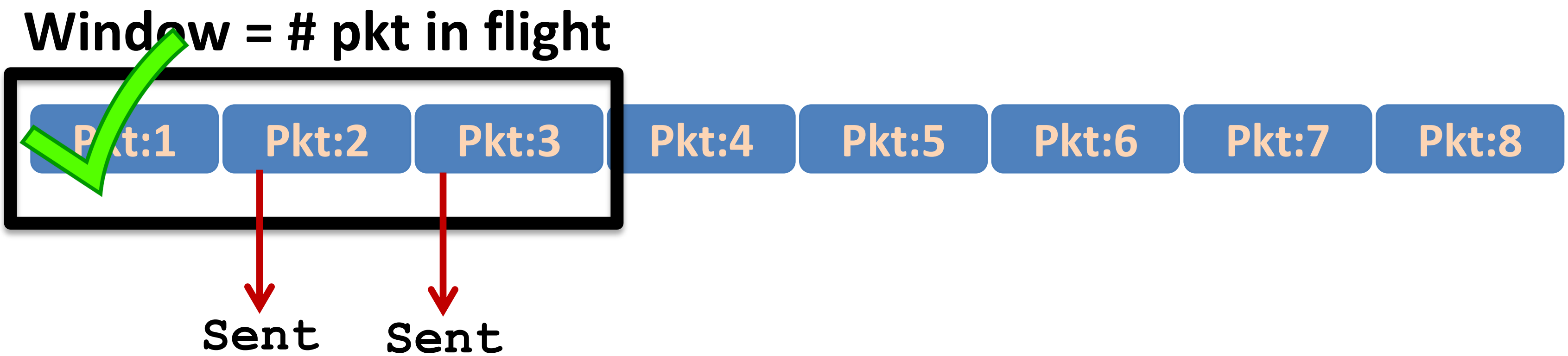
Sliding Window: A series of packets to be sent reliably



Sliding Window: A series of packets to be sent reliably

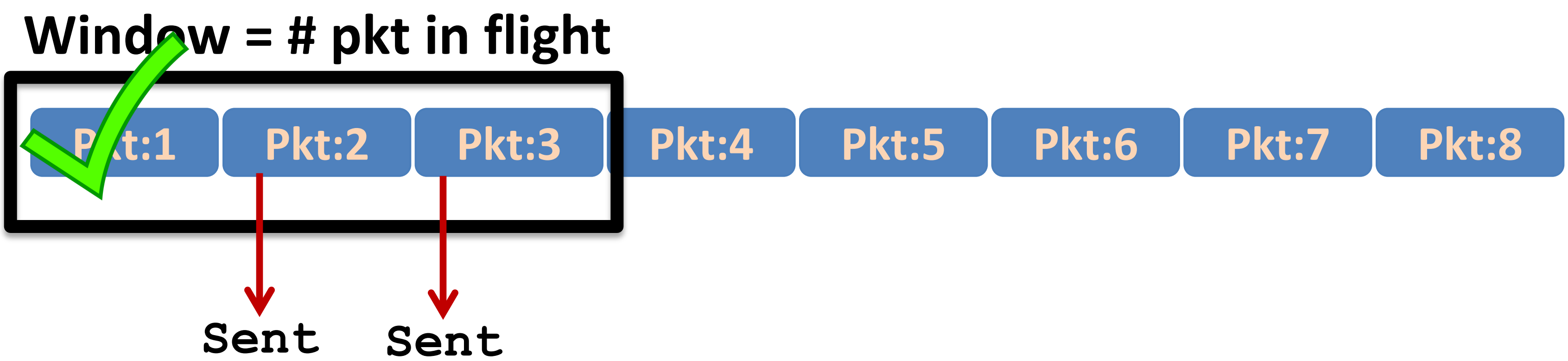


Sliding Window: A series of packets to be sent reliably



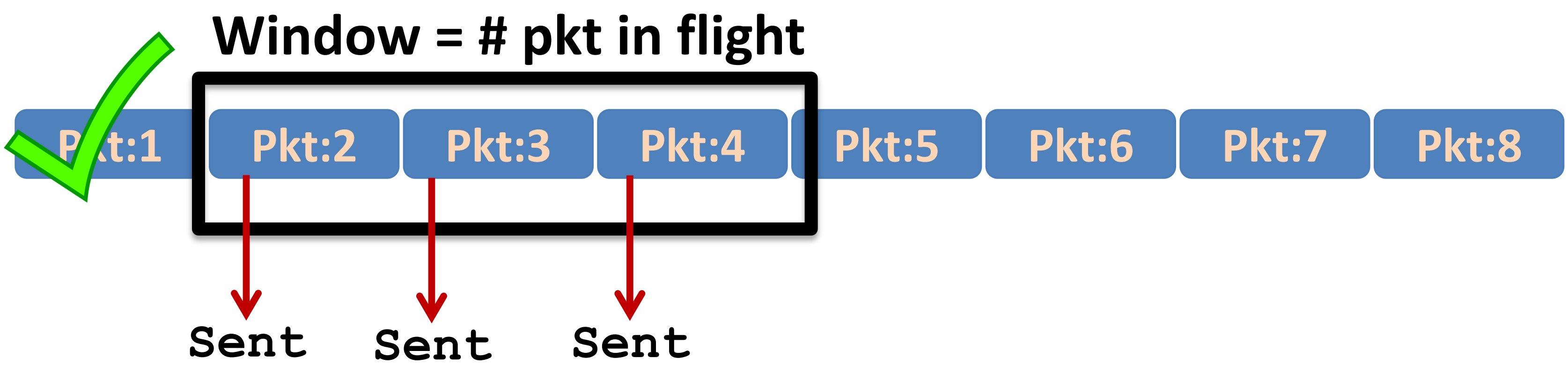
Packet 1 is delivered

Sliding Window: A series of packets to be sent reliably



Packet 1 is delivered. Slide the window.

Sliding Window: A series of packets to be sent reliably



Packet 1 is delivered. Slide the window.

Sliding Window: A series of packets to be sent reliably