CS120: Computer Networks

Lecture 16. TCP 2

Haoxian Chen

Slides adopted from: Zhice Yang

Today's lecture

- TCP sliding window
- Flow control
- Triggering Transmission
- Adaptive timeout
- TCP extensions

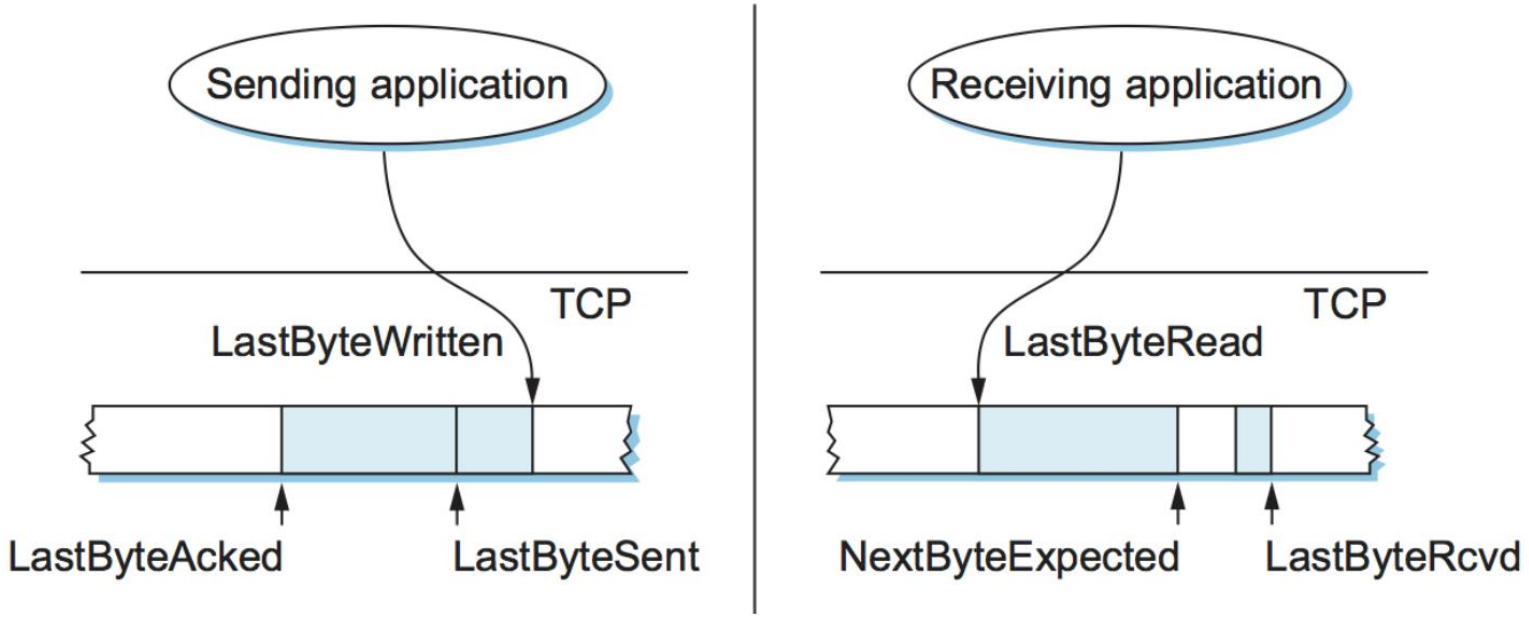
Transmission Control Protocol (TCP)

- RFC: 793,1122,1323, 2018, 2581
- Goal: Reliable, In-order Delivery
  - Connection oriented
  - Flow control
  - Congestion control

➢Core Algorithm: Sliding Window

Sliding Window

Receiver and sender maintains a buffer, respectively.



Send buffer

Receive buffer

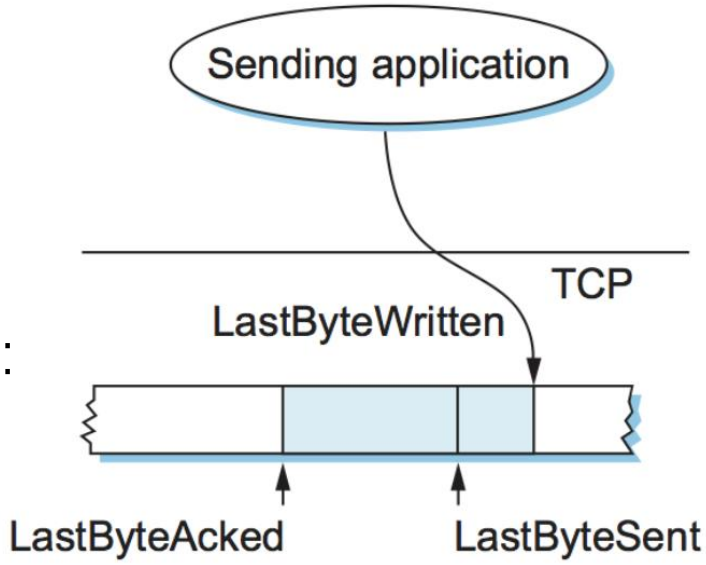
Send buffer

- Cannot ack a byte that has not been sent:

  LastByteAcked <= LastByteSent

- Cannot send a byte that has not been written:

  LastByteSent <= lastByteWritten



Send buffer

Receive buffer

A byte cannot be read until:

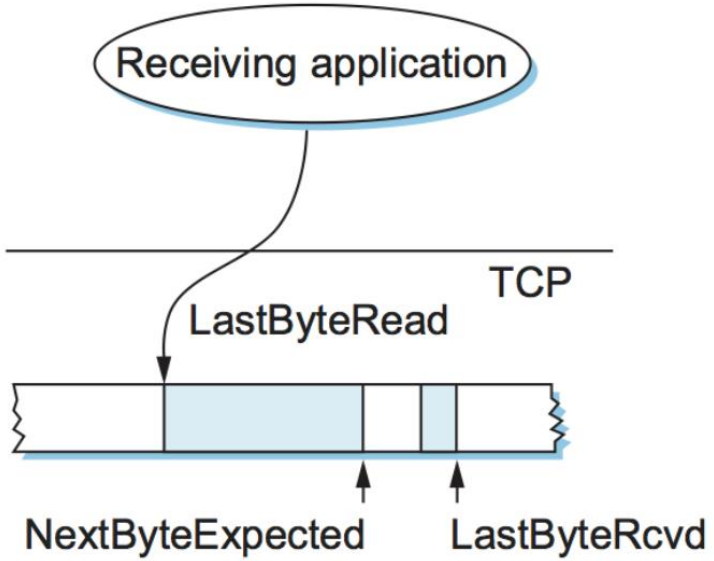
- it is received and
- all preceeding bytes are received

LastByteRead < NextByteExpected

The NextByteExpected is:

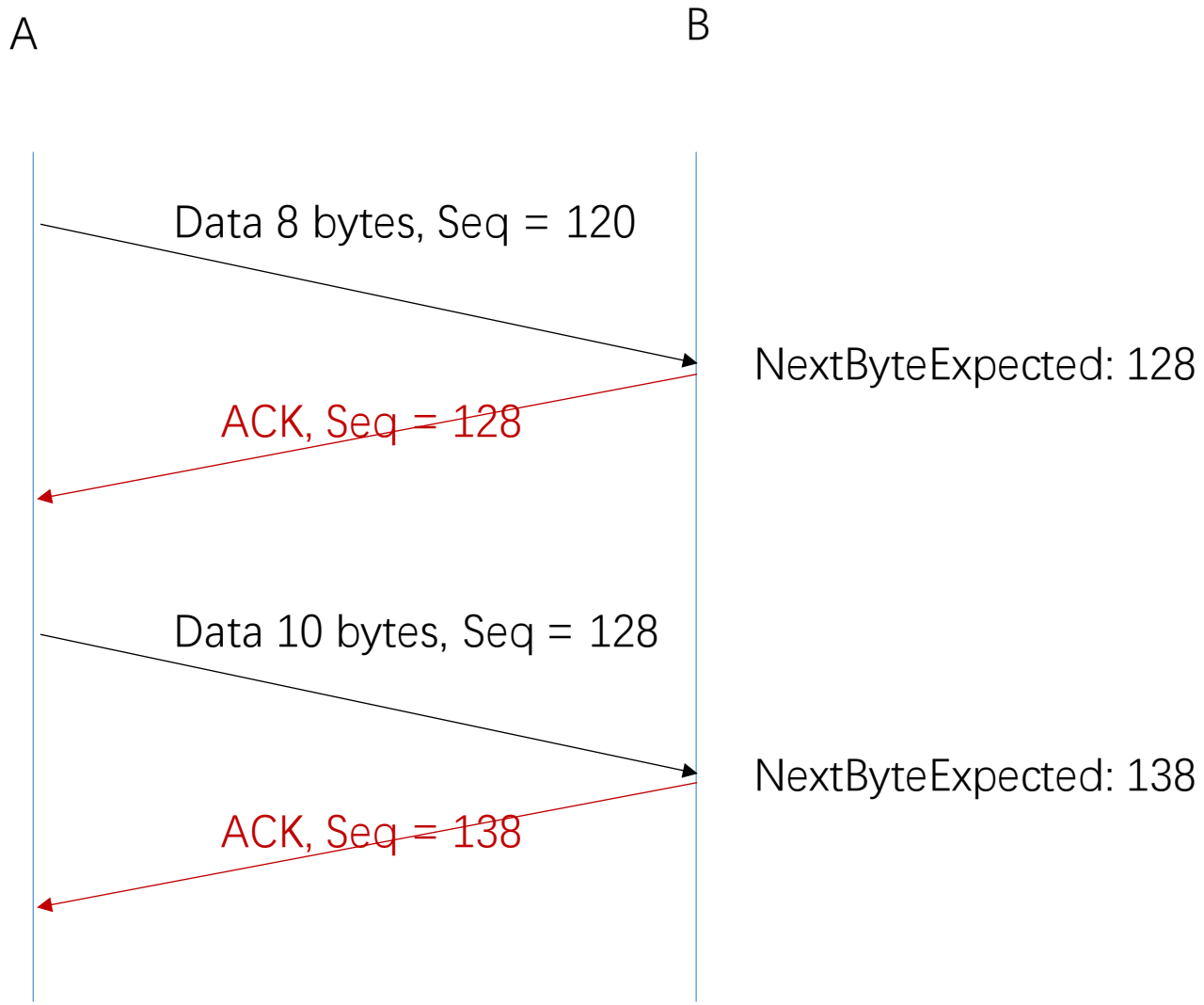
- if all bytes arrive in order, LastByteRcvd + 1
- otherwise, the first gap

NextByteExpected <= LastByteRcvd + 1

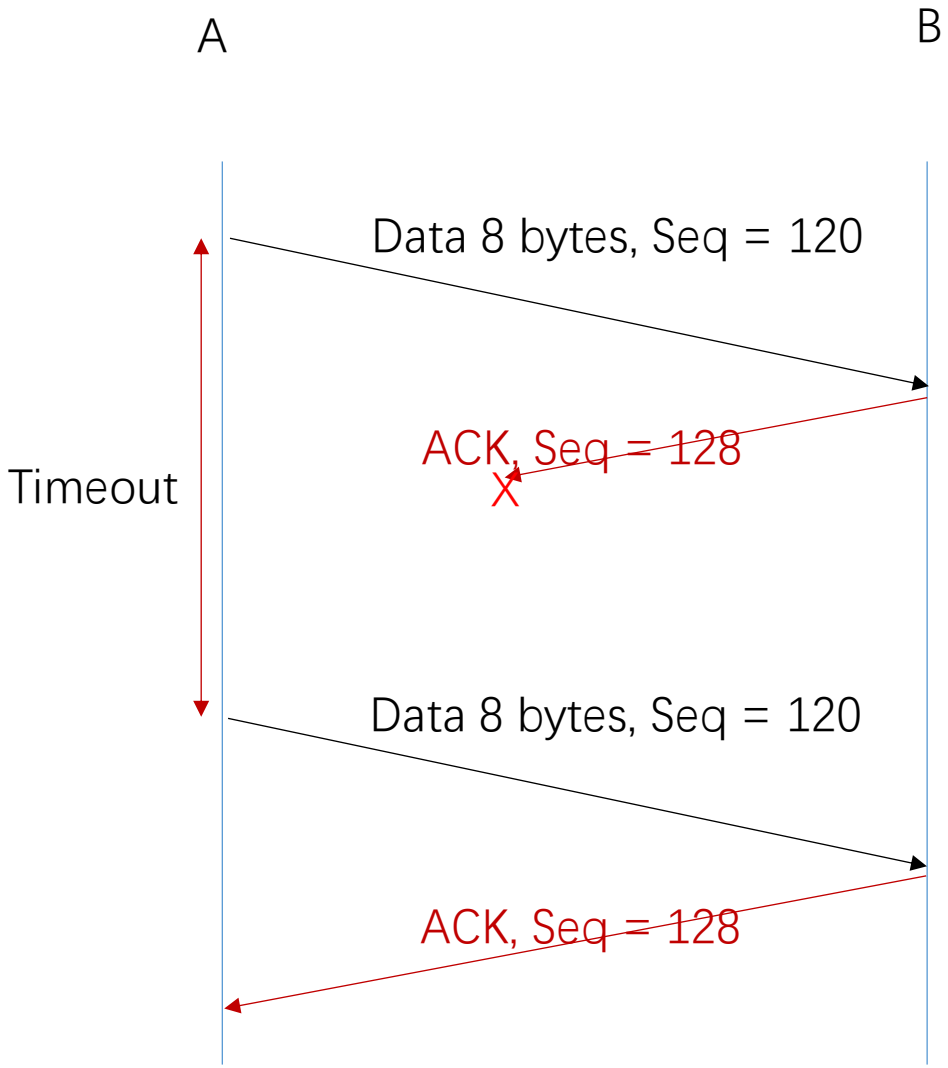


Receive buffer

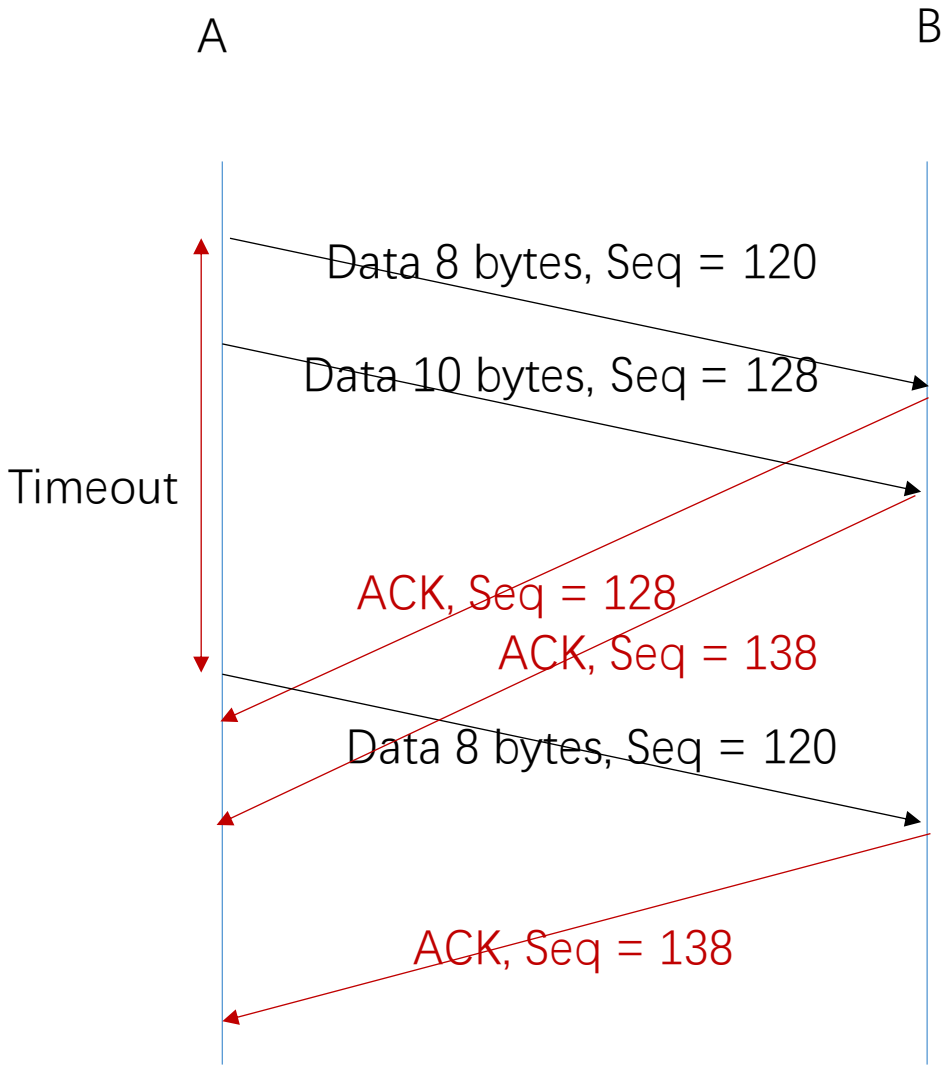
Sliding Window in TCP: Examples



Sliding Window in TCP: Examples

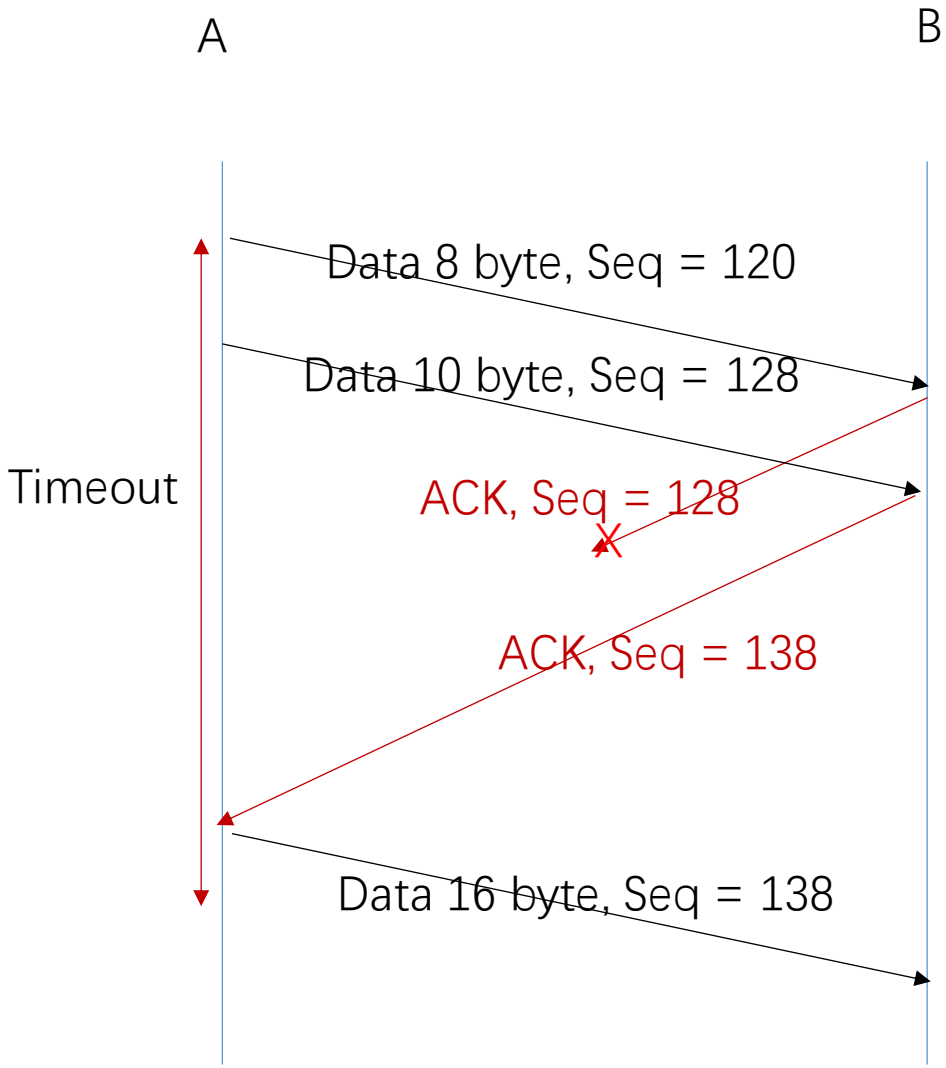


Sliding Window in TCP: Examples



Accumulative ACK

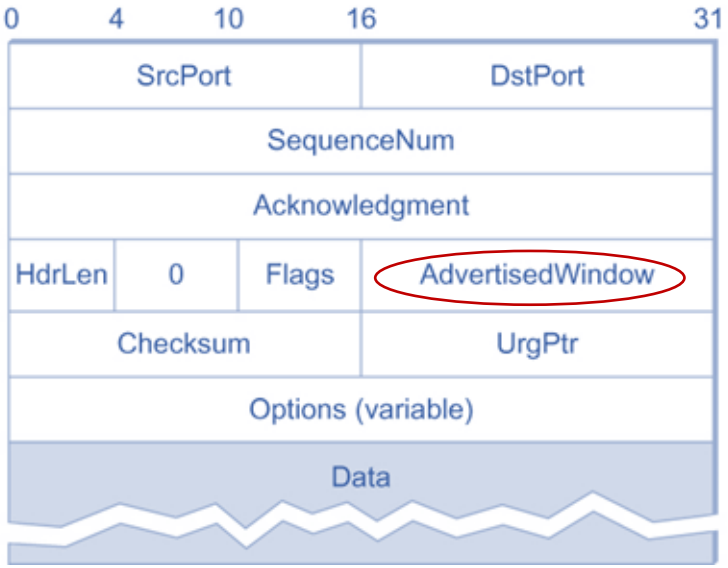
Sliding Window in TCP: Examples



Accumulative ACK

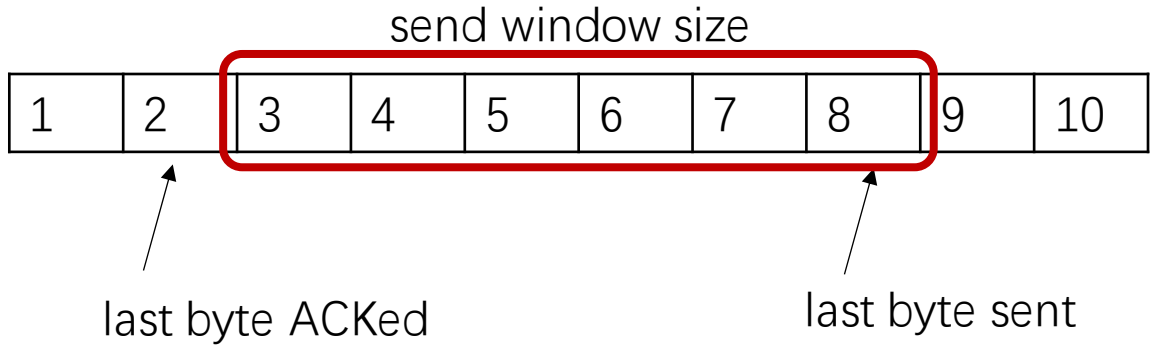
TCP Flow Control

- Goal:
  - Sender does not overrun receiver's buffer
- Approach:
  - Rx window is not fixed, adjusted according to receiving buffer
  - The receiver advertises an adjustable window size (AdvertisedWindow field in TCP header)
  - Sender is limited to having no more than AdvertisedWindow bytes of unACKed data at any time

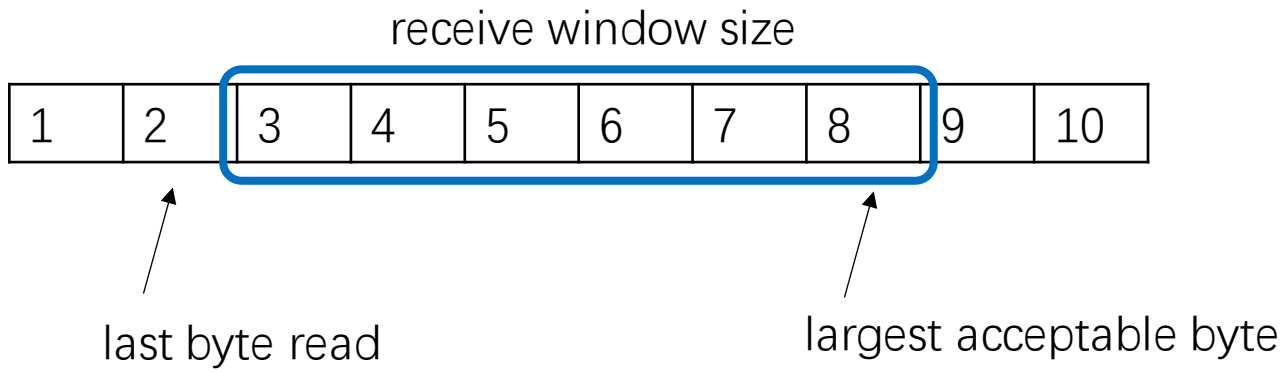


Sliding Window

Sender Buffer:



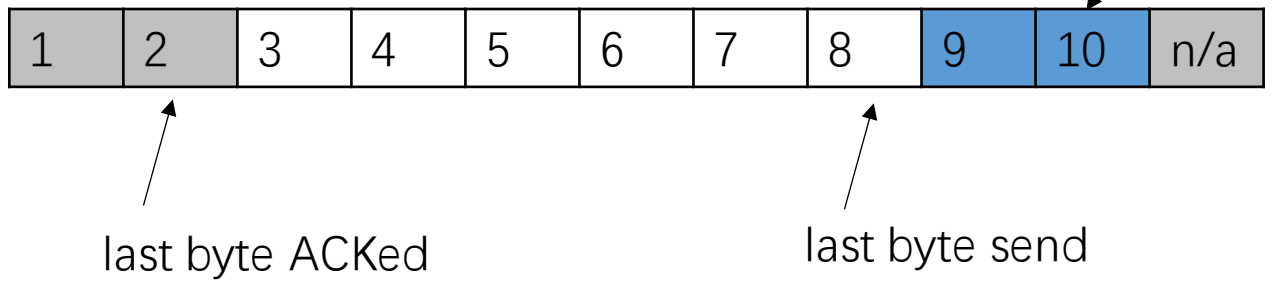
Receiver Buffer:



Sliding Window in TCP

Sender Buffer (8 bytes):

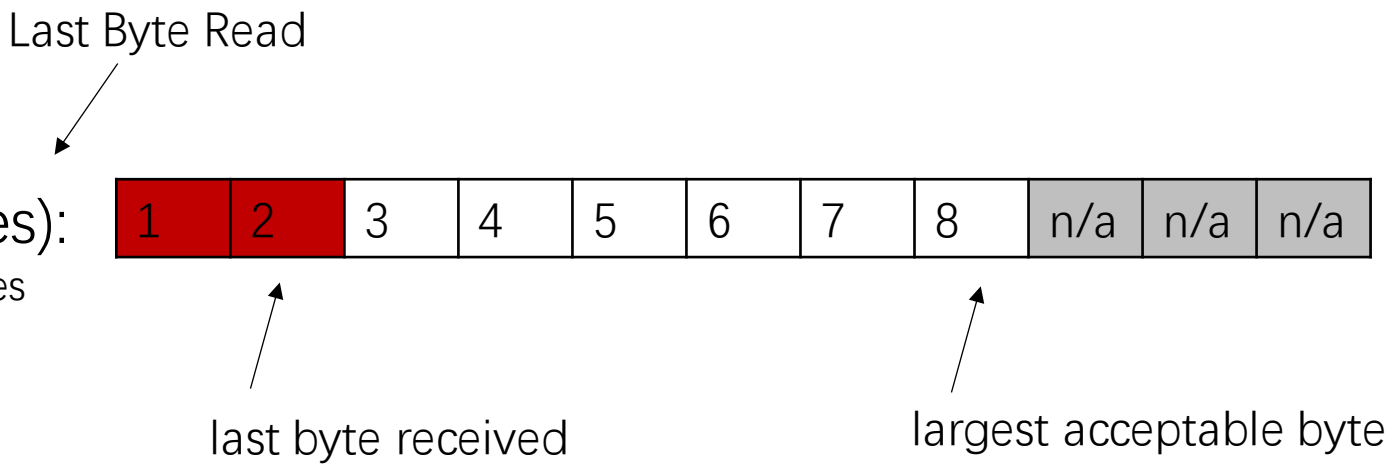
1,2 are freed buffer



Receiver Buffer (8 bytes):

1,2 are occupied by received bytes

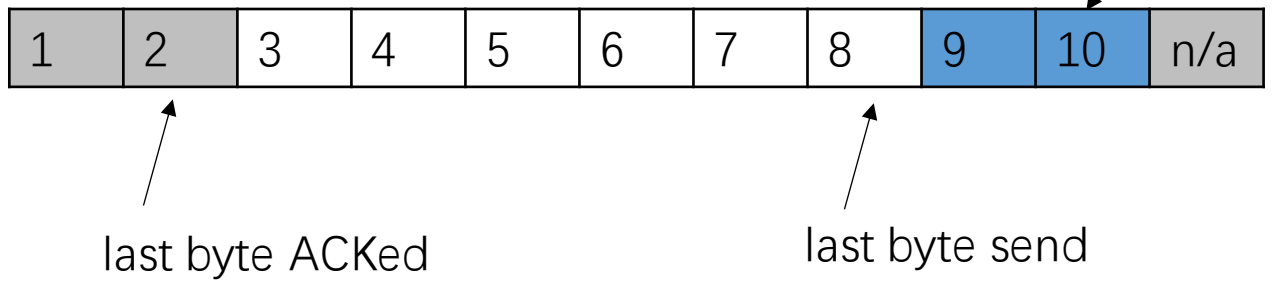
9,10 haven't allocated



Sliding Window in TCP

Sender Buffer (8 bytes):

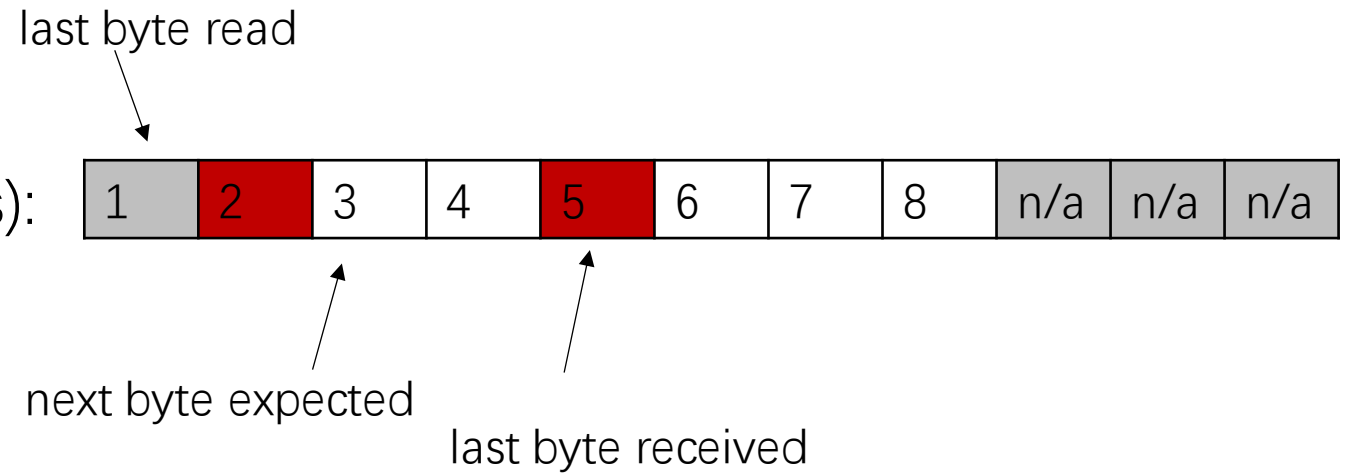
1,2 are freed buffer



Receiver Buffer (7 bytes):

2,5 are occupied by received bytes

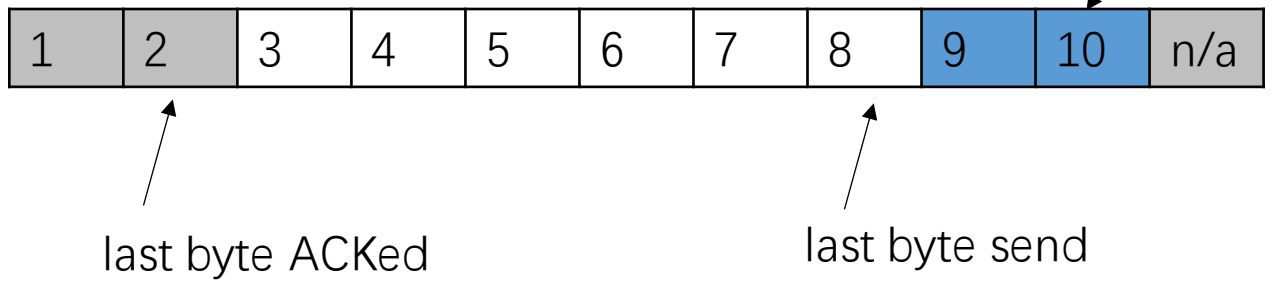
1 is read and freed by receiver



Sliding Window in TCP

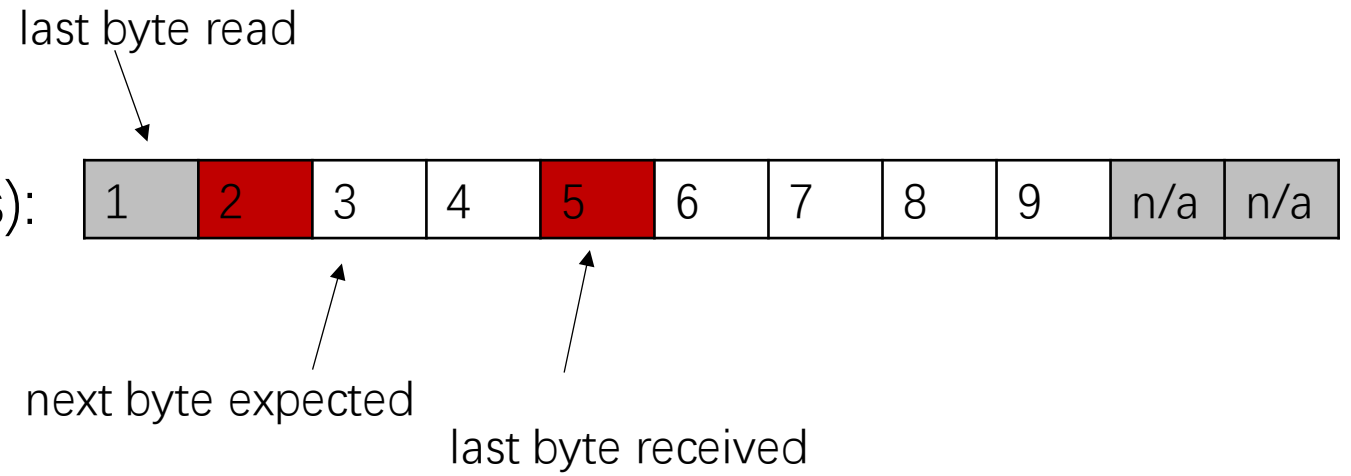
Sender Buffer (8 bytes):

1,2 are freed buffer



Receiver Buffer (8 bytes):

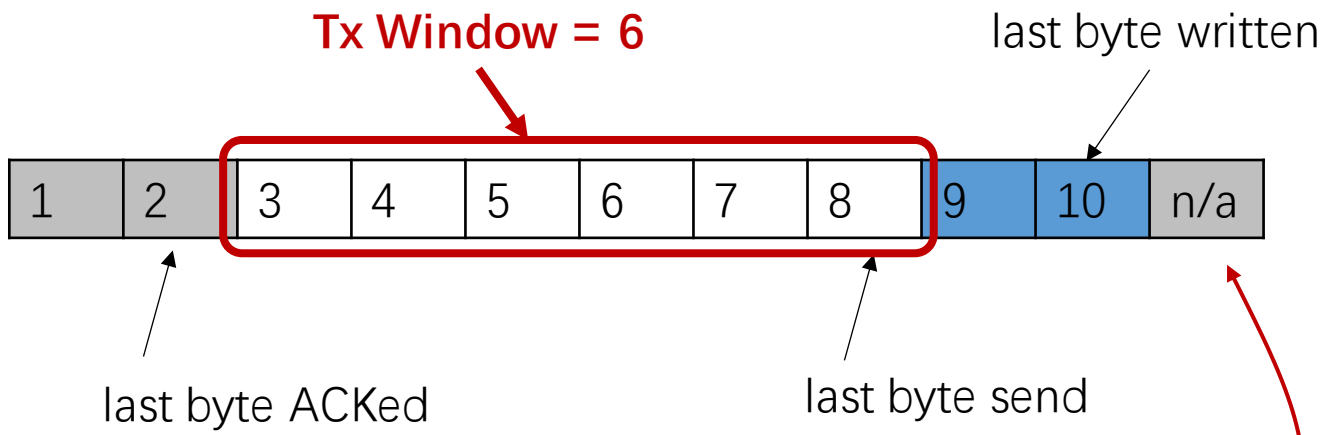
2,5 are occupied by received bytes
1 is read and freed by receiver
9 is allocated for receiving



TCP Flow Control

Sender Buffer (8 bytes):

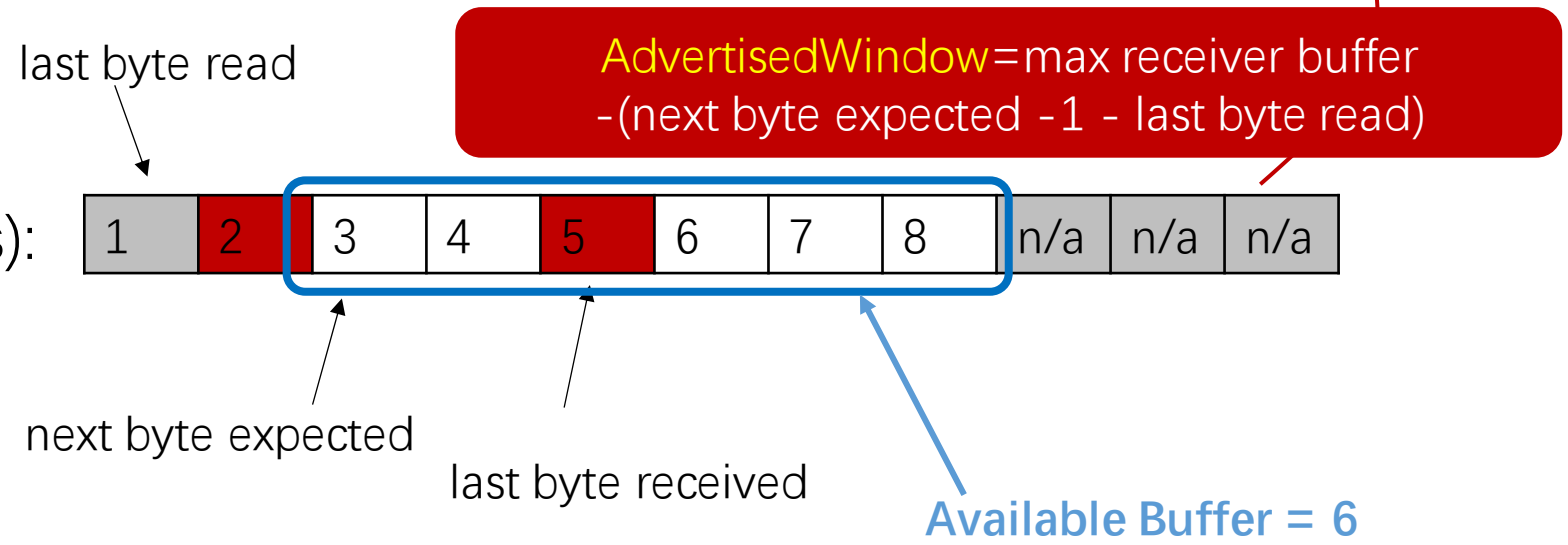
1,2 are freed buffer



Receiver Buffer (7 bytes):

2,5 are occupied by received bytes

1 is read and freed by receiver

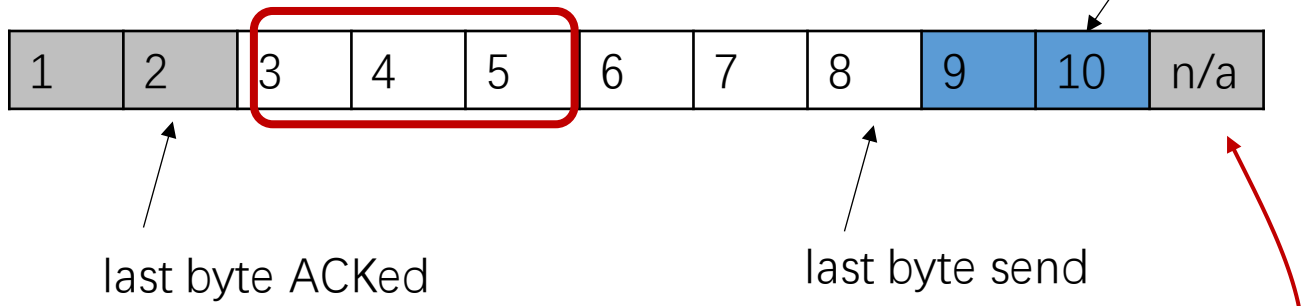


TCP Flow Control

last byte sent – last byte ACKed <= AdvertisedWindow

Sender Buffer (8 bytes):

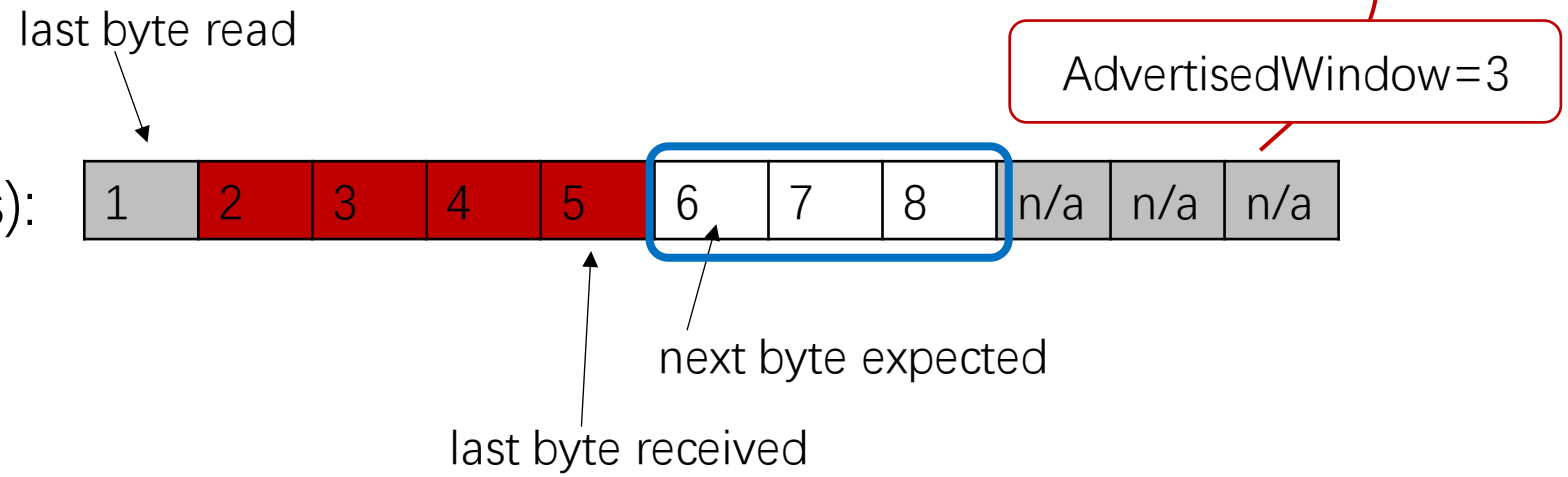
1,2 are freed buffer



Receiver Buffer (7 bytes):

2,5 are occupied by received bytes

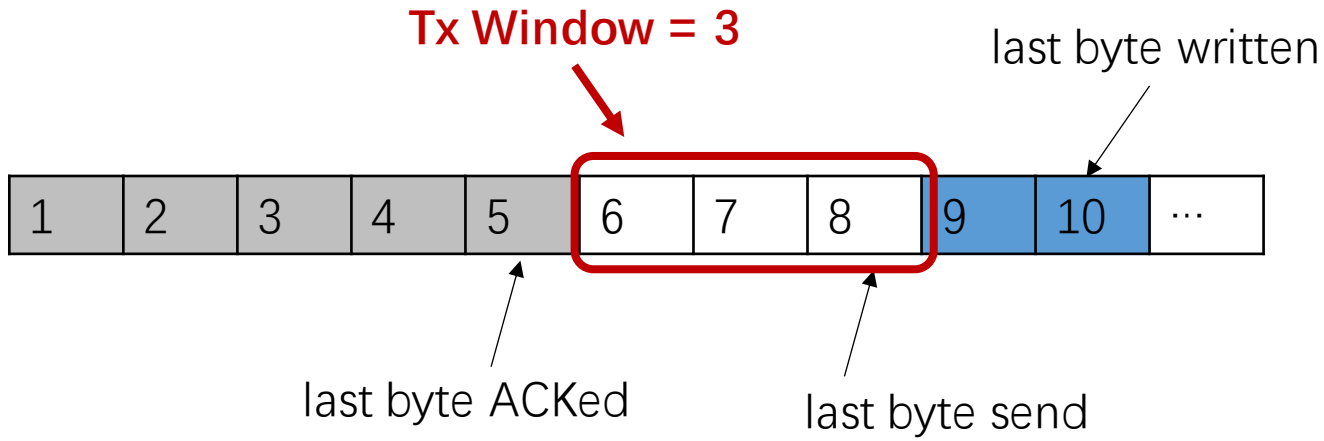
1 is read and freed by receiver



TCP Flow Control

Sender Buffer (8 bytes):

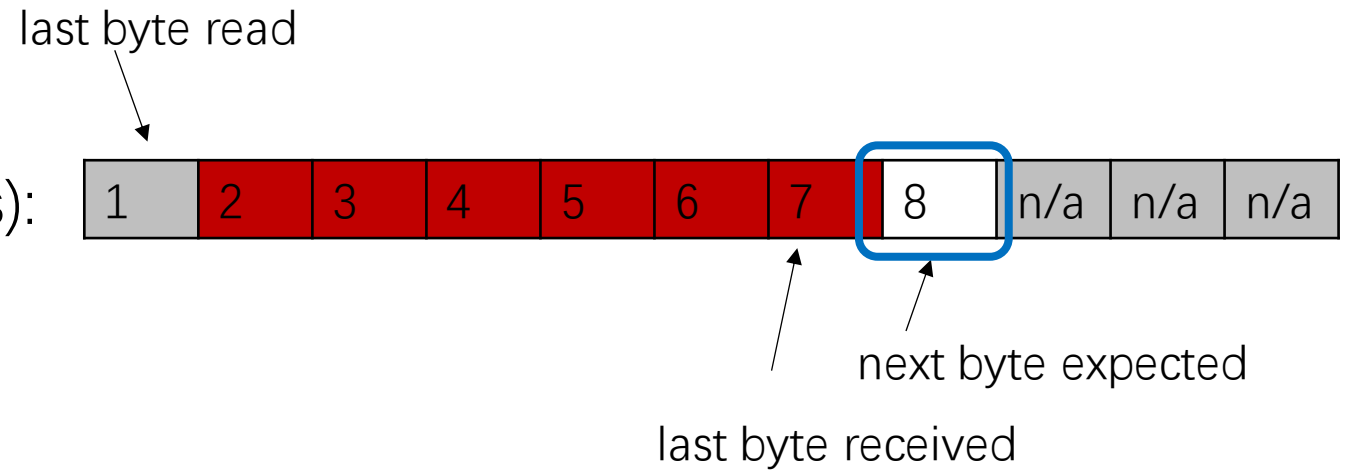
1,2 are freed buffer



Receiver Buffer (7 bytes):

2,5 are occupied by received bytes

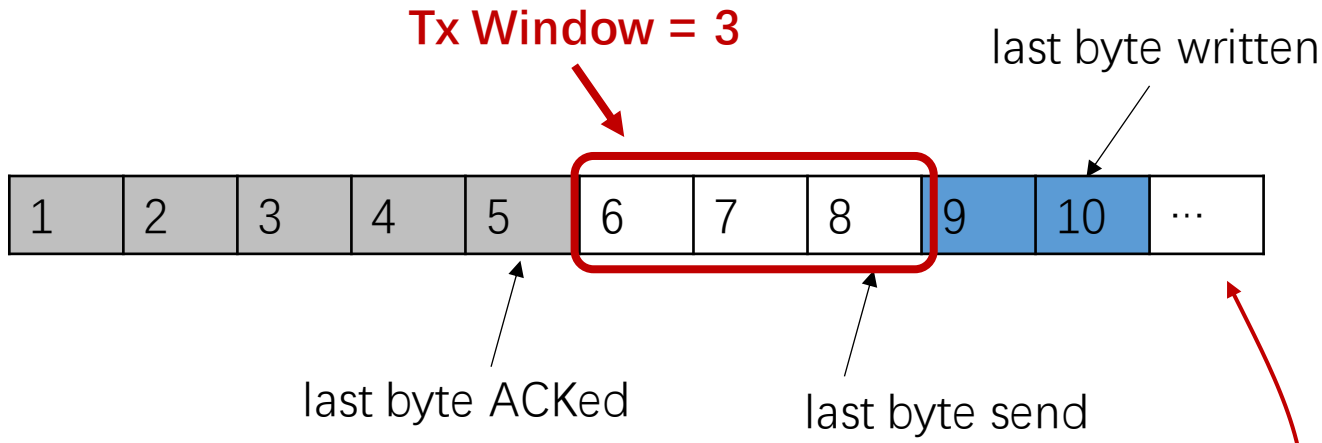
1 is read and freed by receiver



TCP Flow Control

Sender Buffer (8 bytes):

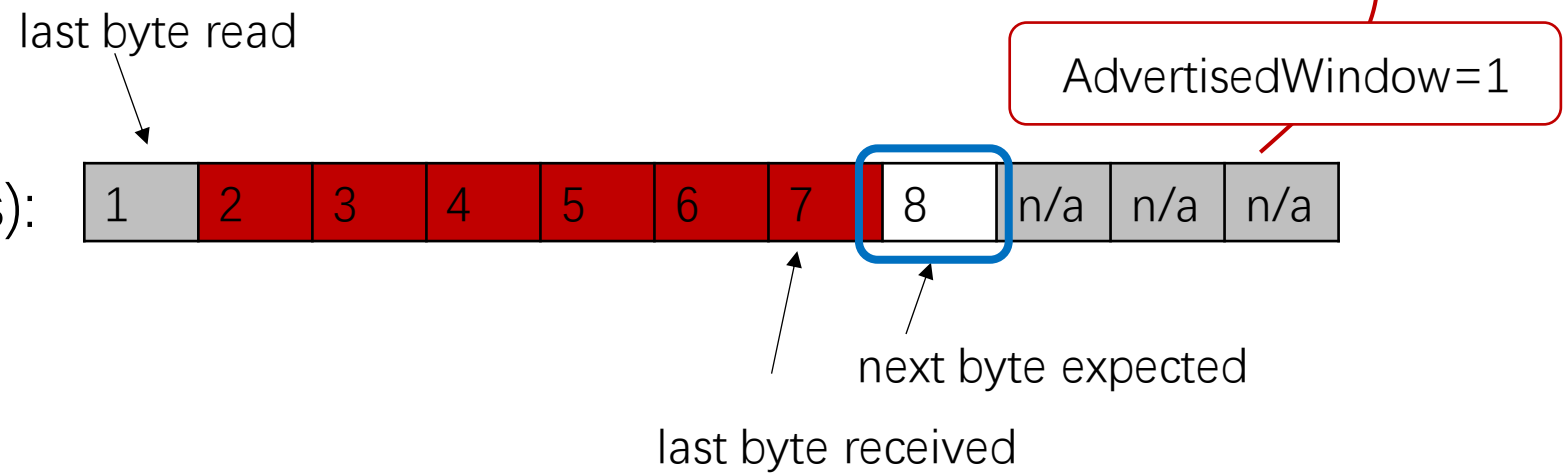
1,2 are freed buffer



Receiver Buffer (7 bytes):

2,5 are occupied by received bytes

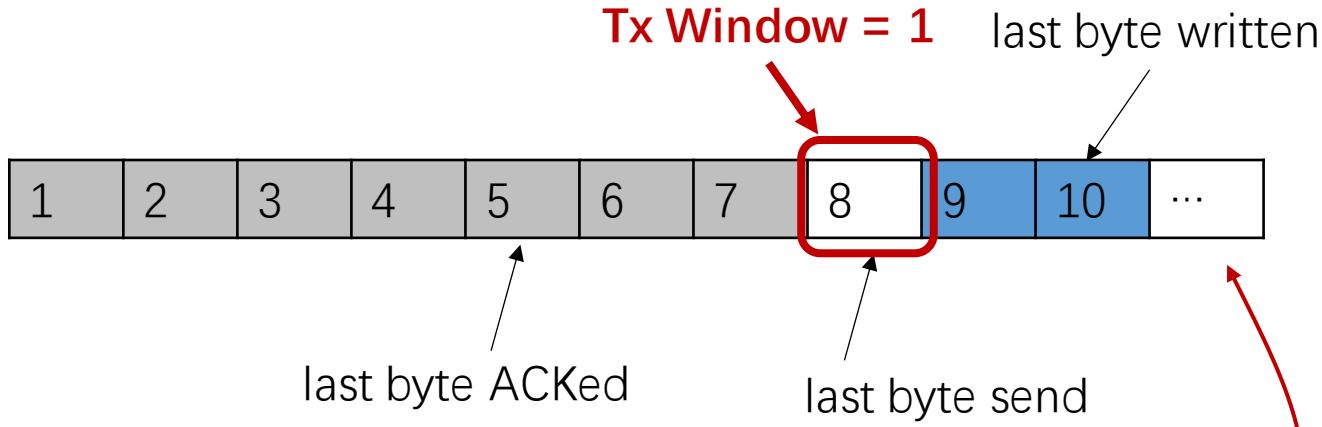
1 is read and freed by receiver



TCP Flow Control

Sender Buffer (8 bytes):

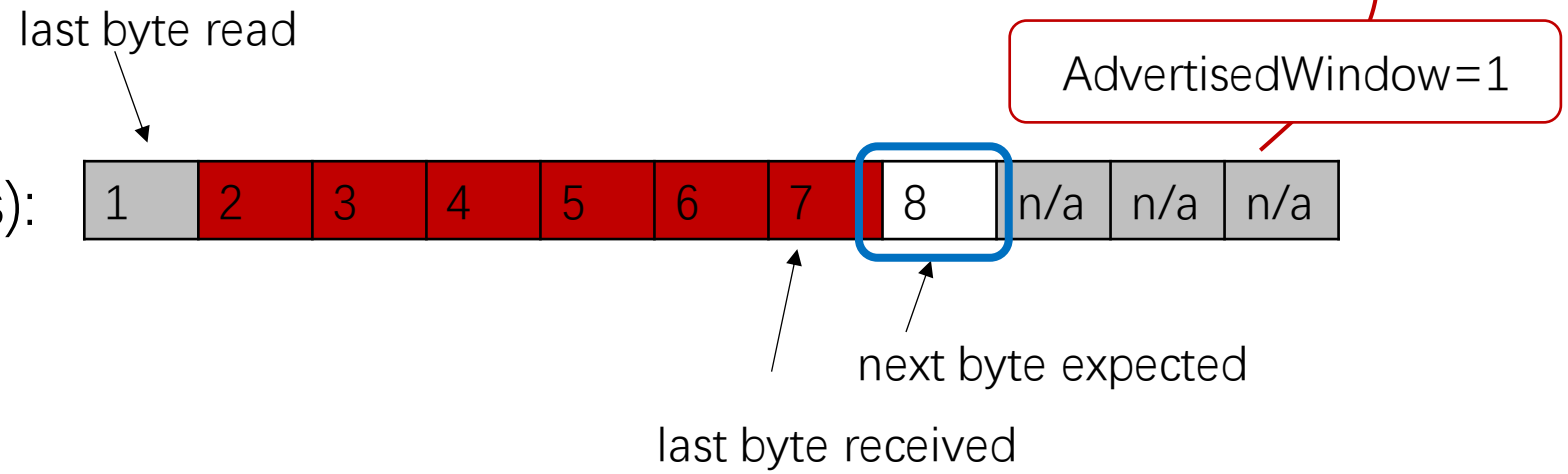
1,2 are freed buffer



Receiver Buffer (7 bytes):

2,5 are occupied by received bytes

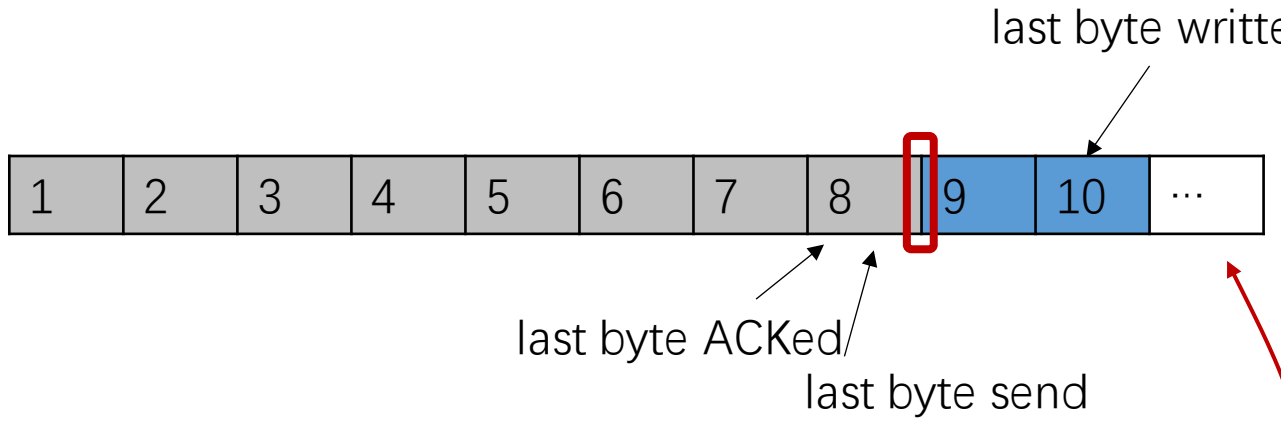
1 is read and freed by receiver



TCP Flow Control

Sender Buffer (8 bytes):

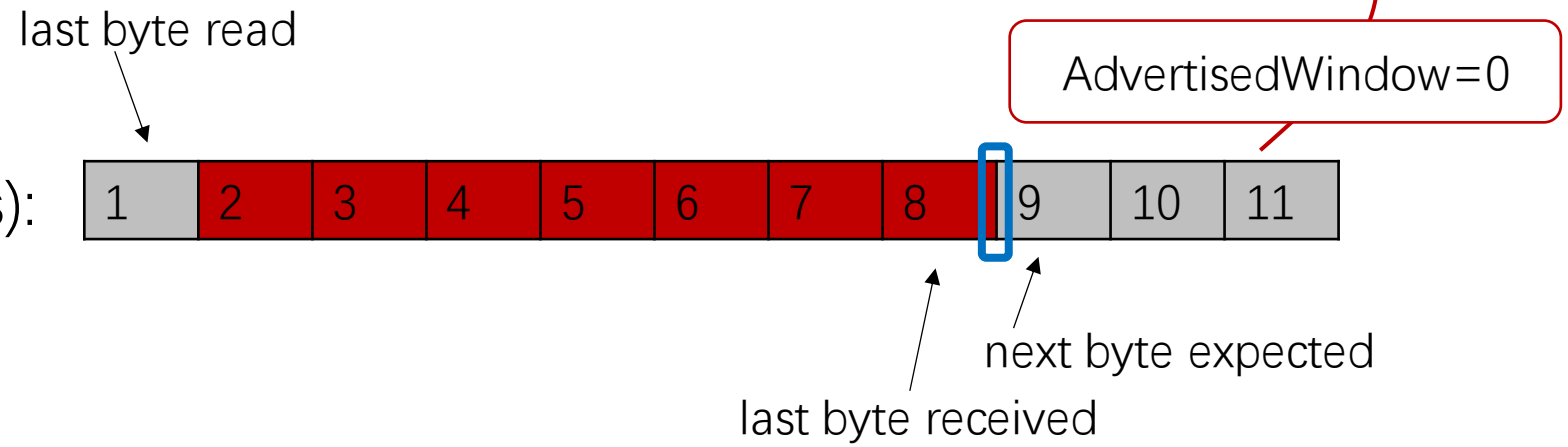
1,2 are freed buffer



Receiver Buffer (7 bytes):

2,5 are occupied by received bytes

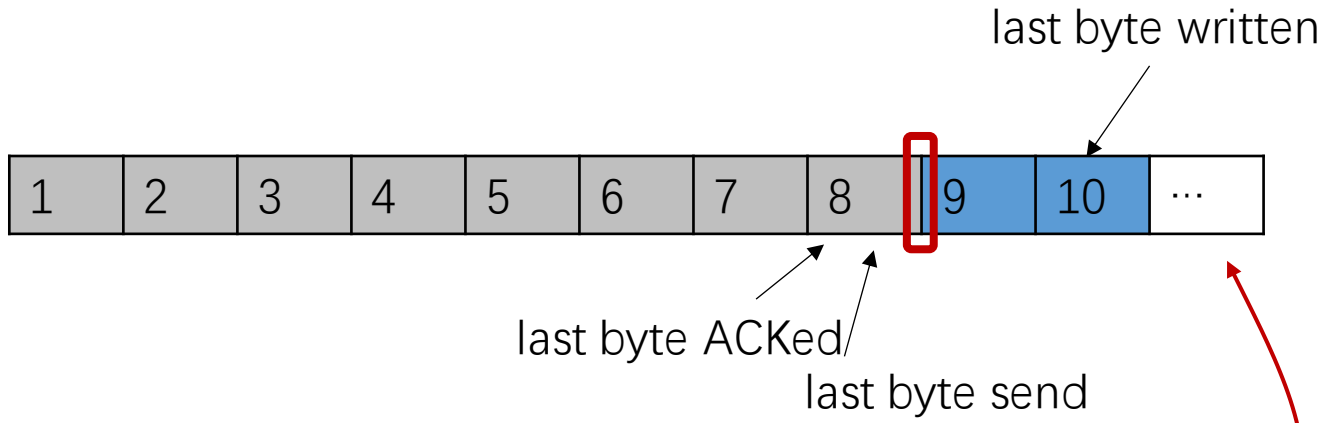
1 is read and freed by receiver



TCP Flow Control

Sender Buffer (8 bytes):

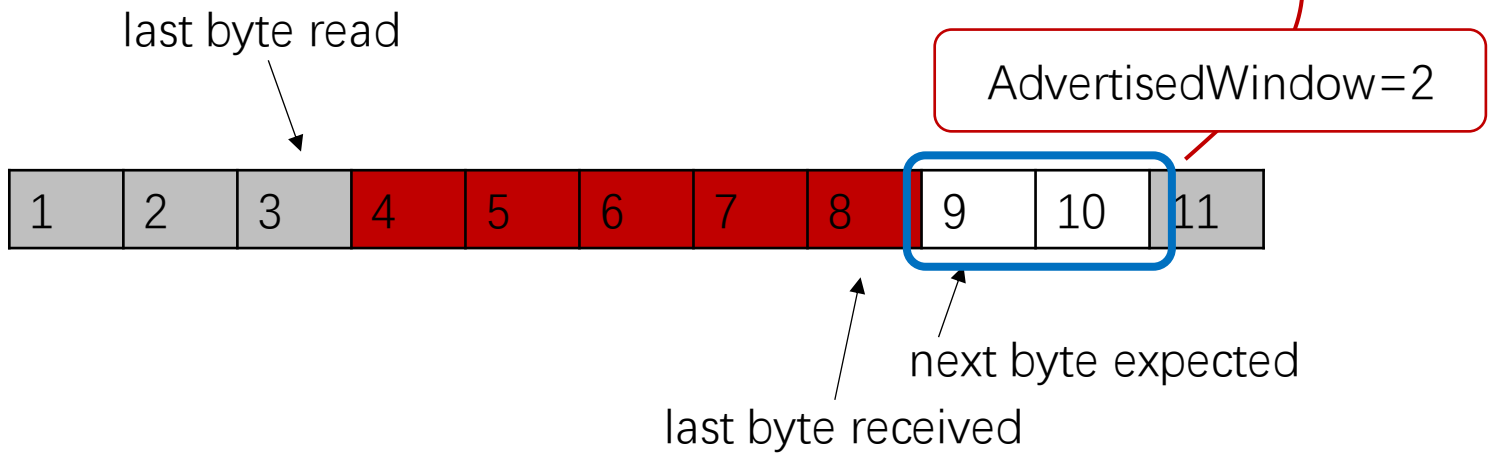
1,2 are freed buffer



Receiver Buffer (7 bytes):

2,5 are occupied by received bytes

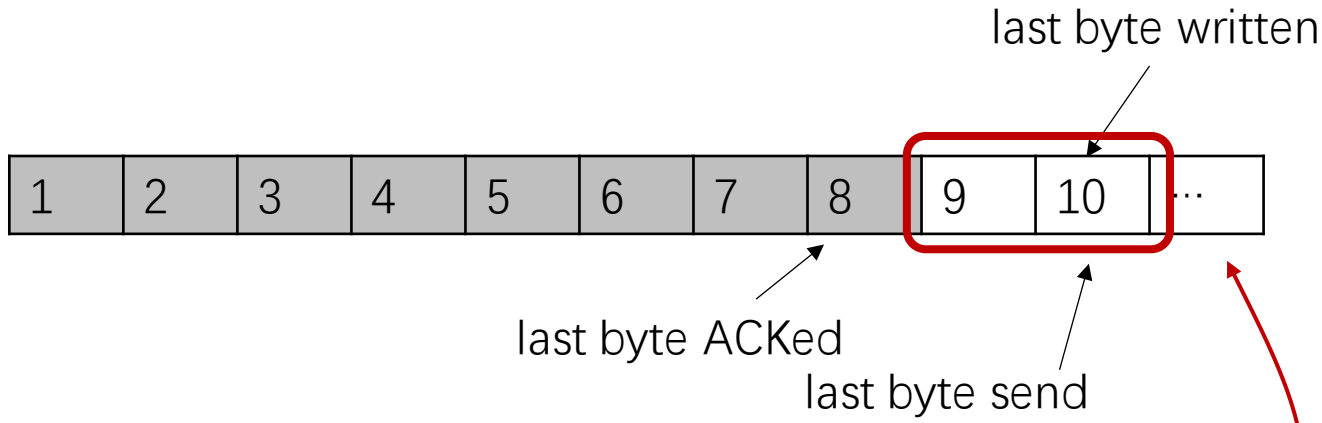
1 is read and freed by receiver



TCP Flow Control

Sender Buffer (8 bytes):

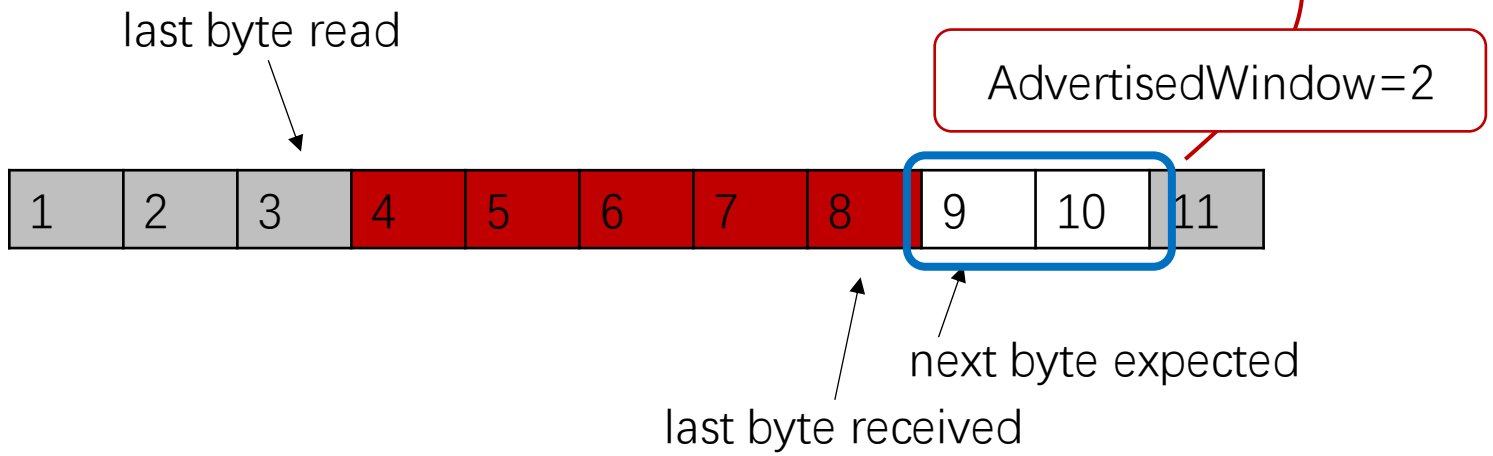
1,2 are freed buffer



Receiver Buffer (7 bytes):

2,5 are occupied by received bytes

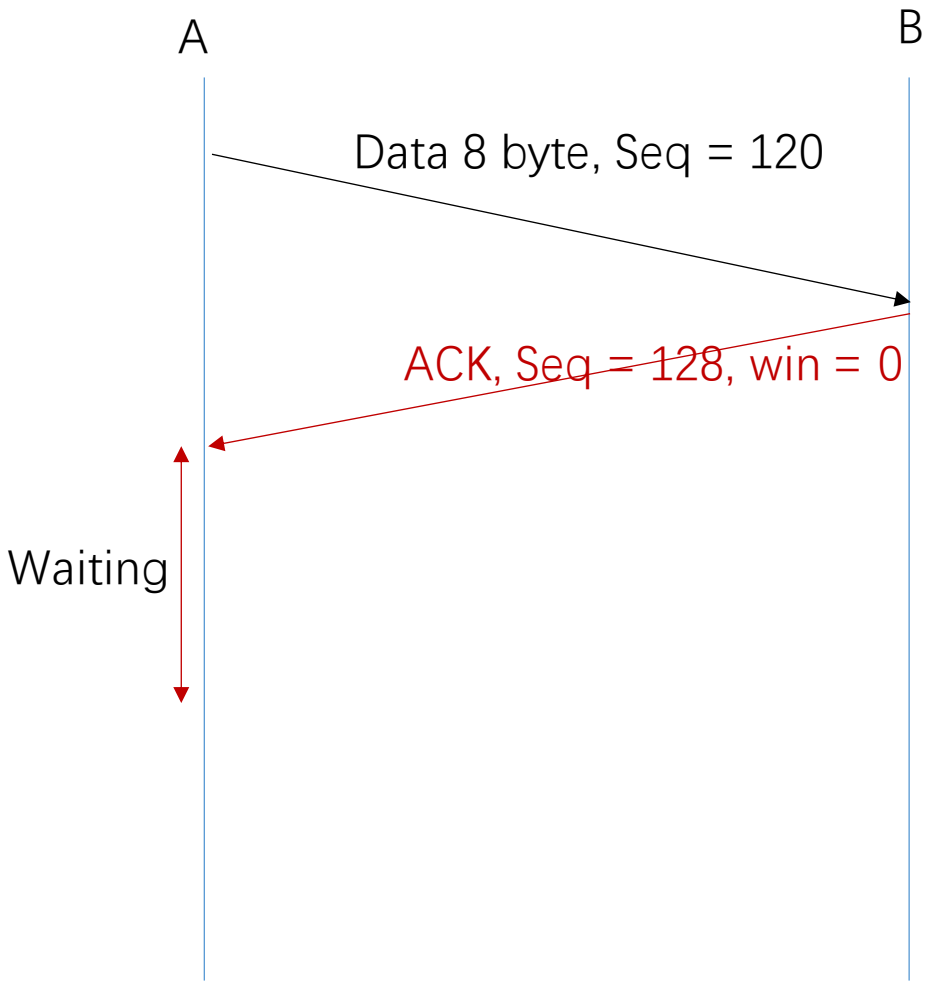
1 is read and freed by receiver



TCP Flow Control

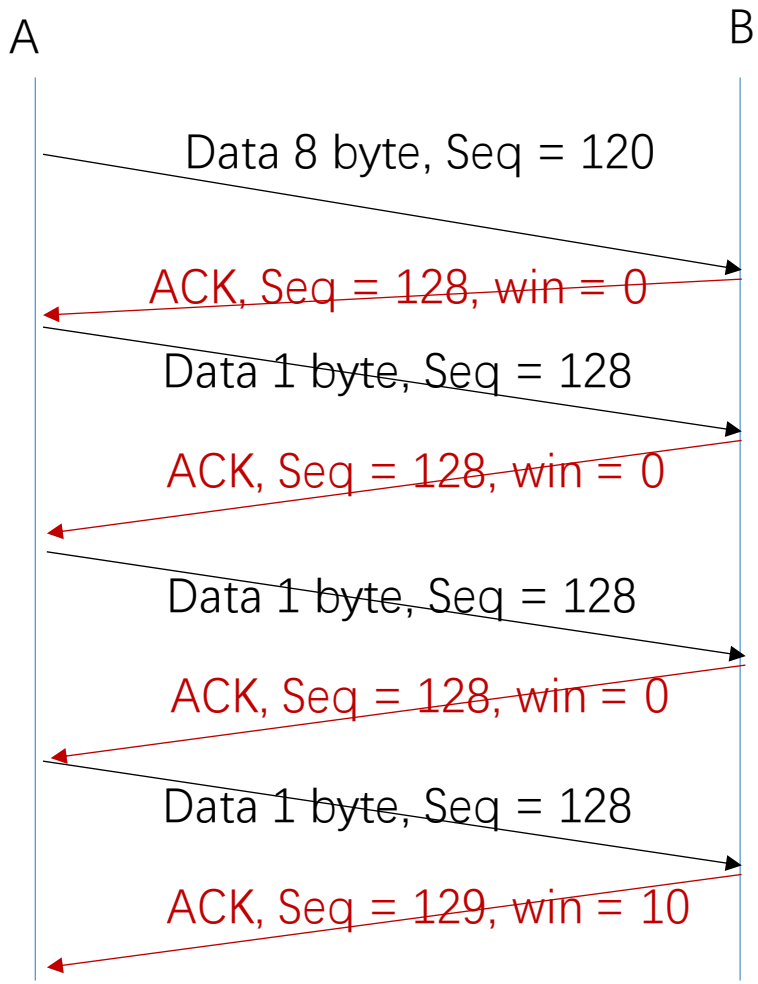
- Receiver throttles sender by advertising a window size no larger than the amount it can buffer:
  - AdvertisedWindow=max receiver buffer-(next byte expected -1 - last byte read)
  - Indicate the amount of free space available in the receive buffer
- Sender must adhere to AdvertisedWindow from the receiver such that:
  - last byte sent – last byte ACKed <= AdvertisedWindow

TCP Flow Control: 0 window case

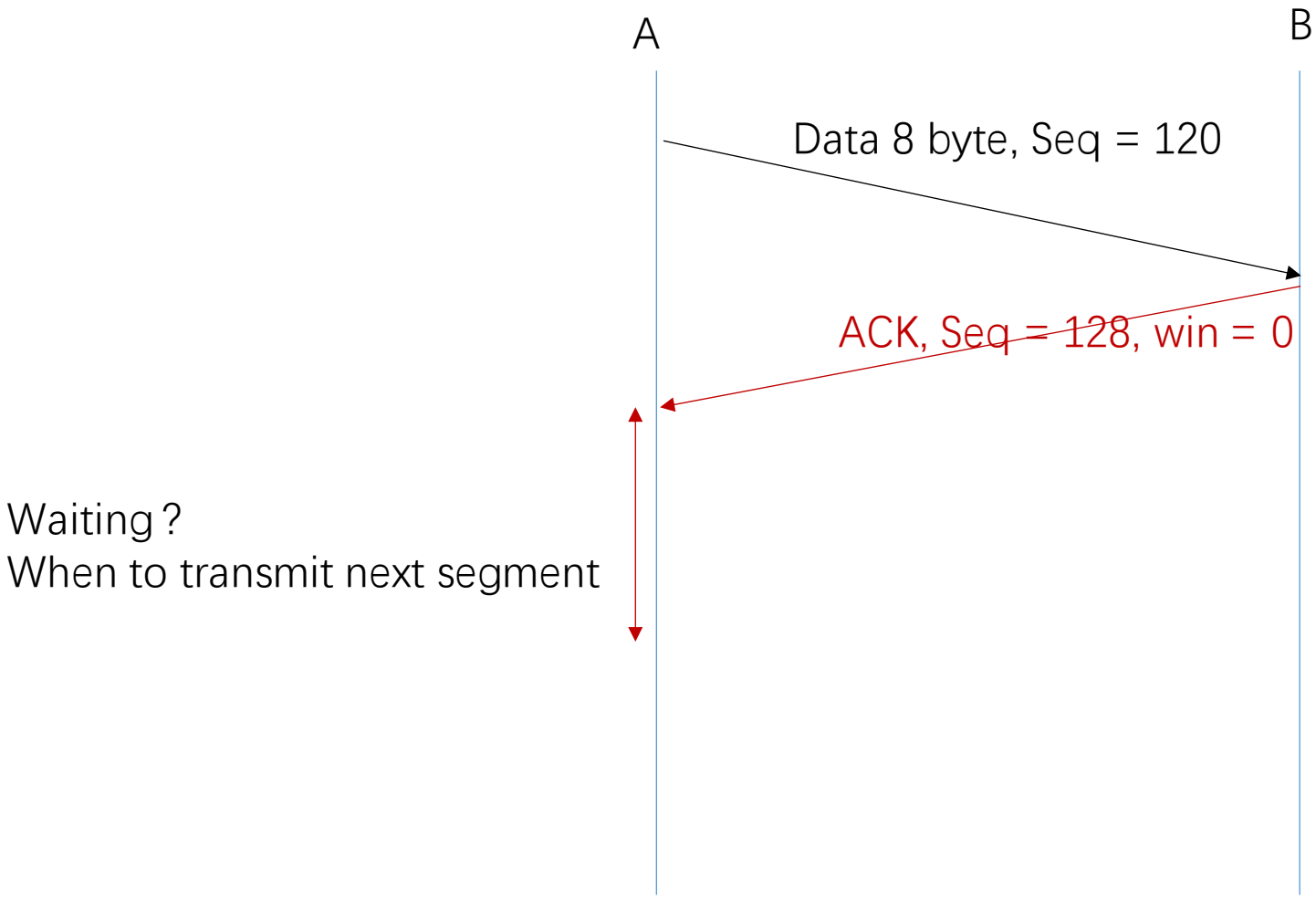


TCP Flow Control: 0 window case

Send 1 byte segment to probe



TCP Flow Control: 0 window case



TCP: triggering transmission

When to transmit data?

1. Once local buffer accumulates a certain amount, called Maximum Segment Size (MSS), and has big enough advertised window.

2. Explicit “push” operation.

3. When a ”timer” fires.

TCP Flow Control: Silly Window Syndrome

Suppose the send buffer has bytes to send, and the advertised window is only MSS / 2.

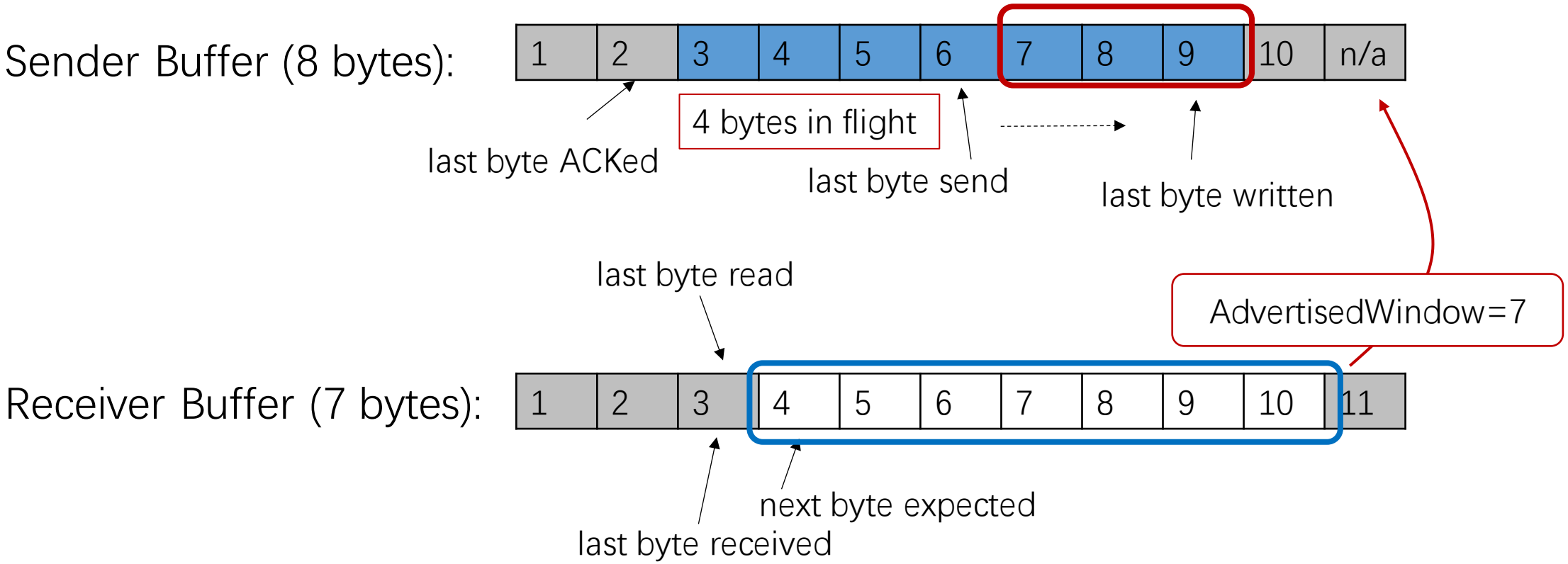
Should it:

- Send a half full segment ➔ Silly Window Syndrome
- Or wait till the window to open to a full MSS?

TCP Flow Control: Silly Window Syndrome

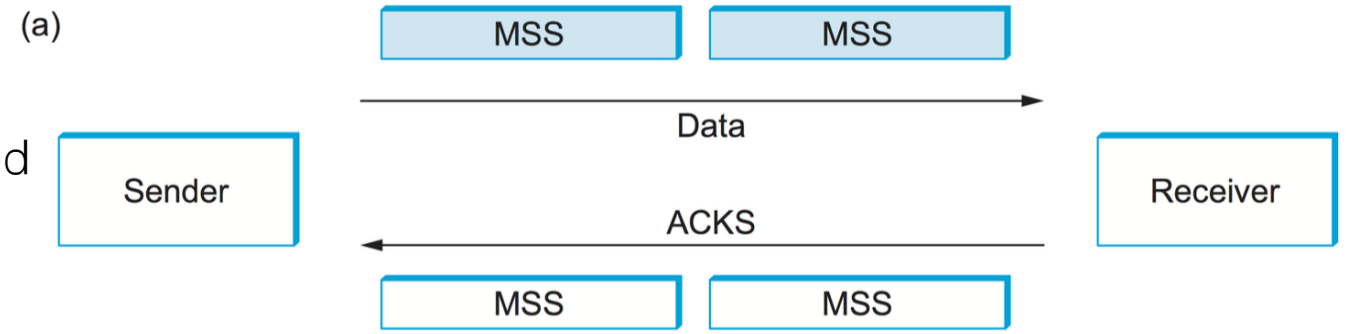
UsableWindow = AdvertisedWindow - BytesInFlight

Usable window = 3

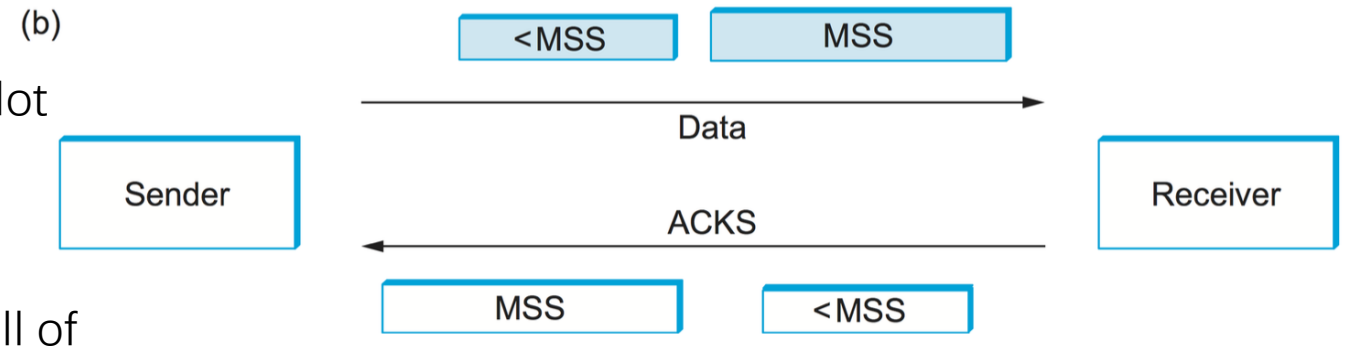


TCP Flow Control: Silly Window Syndrome

Normal case: a full MSS segment is sent and ACKed every time.



Once a smaller segment is sent or ACKed, the small slot persists, breaking up the available window.



Eventually connection is full of small segments, wasting bandwidth.

TCP: How to Pack a Segment

- Best of Effort
  - Silly Window Syndrome
- Wait Until Full
  - Less Timely
- Hybrid: Nagle's algorithm
  - Wait Until Full + Timer

TCP: Nagle's Algorithm

When the application produces data to send

    if both the available data and the window ⩾ MSS

        send a full segment

    else

        if there is unACKed data in flight

            buffer the new data until an ACK arrives

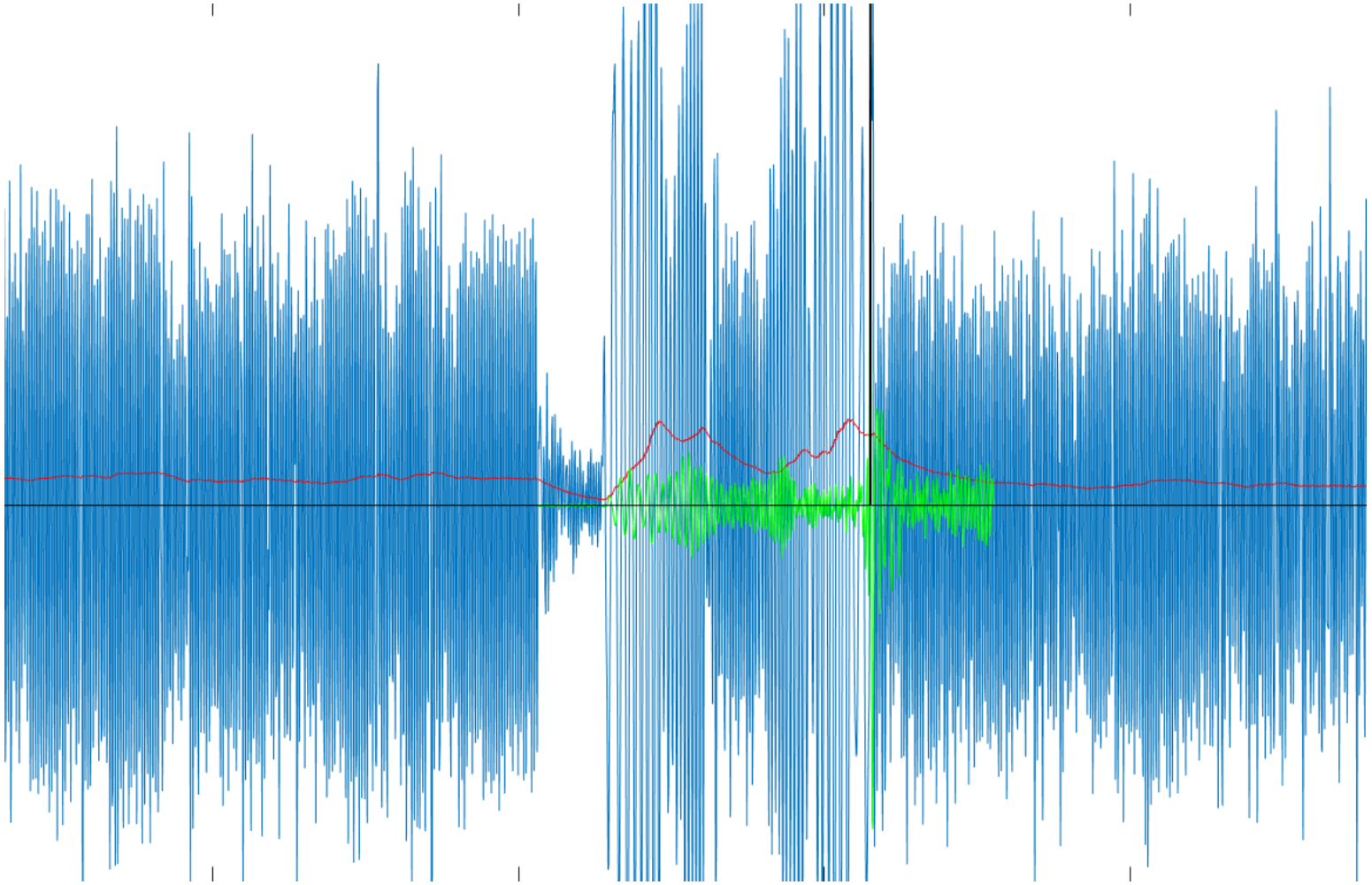
        else

            send all the new data now

Sliding Window in TCP: Adaptive Timeout

Original Algorithm: moving average of RTT

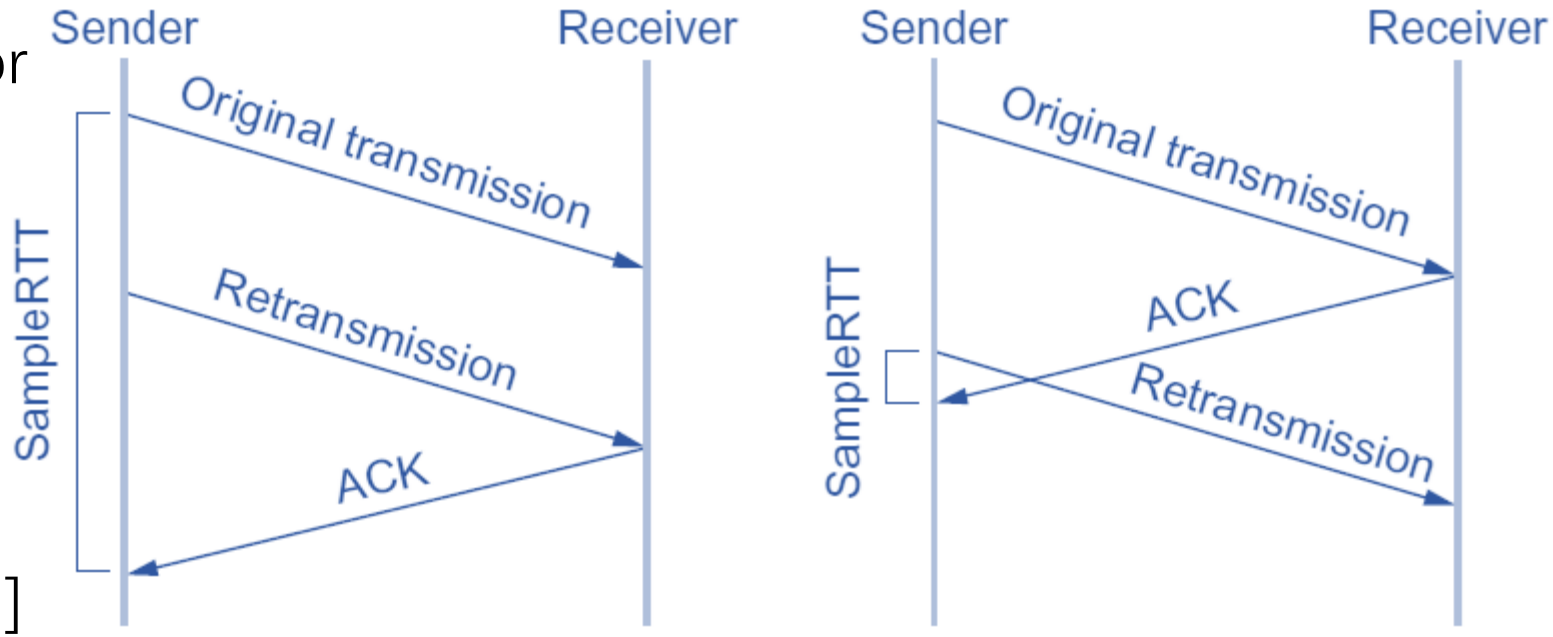
- Measure SampleRTT for each segment/ ACK pair
- Compute weighted average of RTT
  - EstimatedRTT = a * EstimatedRTT + (1 - a)* SampleRTT
  - a is between 0.8 and 0.9
- Set timeout based on EstimatedRTT
  - TimeOut = 2 * EstimatedRTT



Sliding Window in TCP: Adaptive Timeout

Problem:
cannot distinguish ACK for original transmission or retransmission

Solution:
Do not sample RTT when retransmitting.
[Karn/Partridge Algorithm]



Sliding Window in TCP: Adaptive Timeout

Karn/Partridge Algorithm:

- Do not sample RTT when retransmitting

- Exponential backoff: each time TCP retransmits, double the timeout.

  After successful retransmission (ACK is received), the timeout threshold is reset to the RTT-based value.

Rationale: congestion is the most likely cause of lost segments, meaning that the TCP source should not react too aggressively to a timeout.

Sliding Window in TCP: Adaptive Timeout

Jacobson/Karels Algorithm: take RTT variance into account

$$\text{Difference} = \text{SampleRTT} - \text{EstimatedRTT}$$

$$\text{EstimatedRTT} = \text{EstimatedRTT} + (\delta * \text{Difference})$$

$$\text{Deviation} = \text{Deviation} + \delta * (|\text{Difference}| - \text{Deviation})$$

- δ is typically set to 1/8

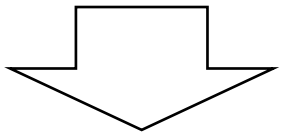
$$\text{TimeOut} = \mu * \text{EstimatedRTT} + \varphi * \text{Deviation}$$

- μ is typically set to 1 and φ is set to 4

Sliding Window in TCP: Adaptive Timeout

- Jacobson/Karels Algorithm Implementation

Difference = SampleRTT − EstimatedRTT
EstimatedRTT = EstimatedRTT + (δ∗Difference)
Deviation = Deviation + δ∗(|Difference| − Deviation)
TimeOut = μ ∗ EstimatedRTT + φ ∗ Deviation



```
SampleRTT -= (EstimatedRTT >> 3);
EstimatedRTT += SampleRTT;
if (SampleRTT < 0)
        SampleRTT = -SampleRTT;
SampleRTT -= (Deviation >> 3);
Deviation += SampleRTT;
TimeOut = (EstimatedRTT >> 3) + (Deviation >> 1);
```

Transmission Control Protocol (TCP)

- RFC: 793,1122,1323, 2018, 2581
- Goal: Reliable, In-order Delivery
  - Connection oriented
  - ➢ Flow control
  - Congestion control
- ➢ Core Algorithm: Sliding Window

Transmission Control Protocol (TCP)

- RFC: 793,1122,1323, 2018, 2581
  - ➢TCP patches
- Goal: Reliable, In-order Delivery
  - Connection oriented
  - Flow control
  - Congestion control
- Core Algorithm: Sliding Window

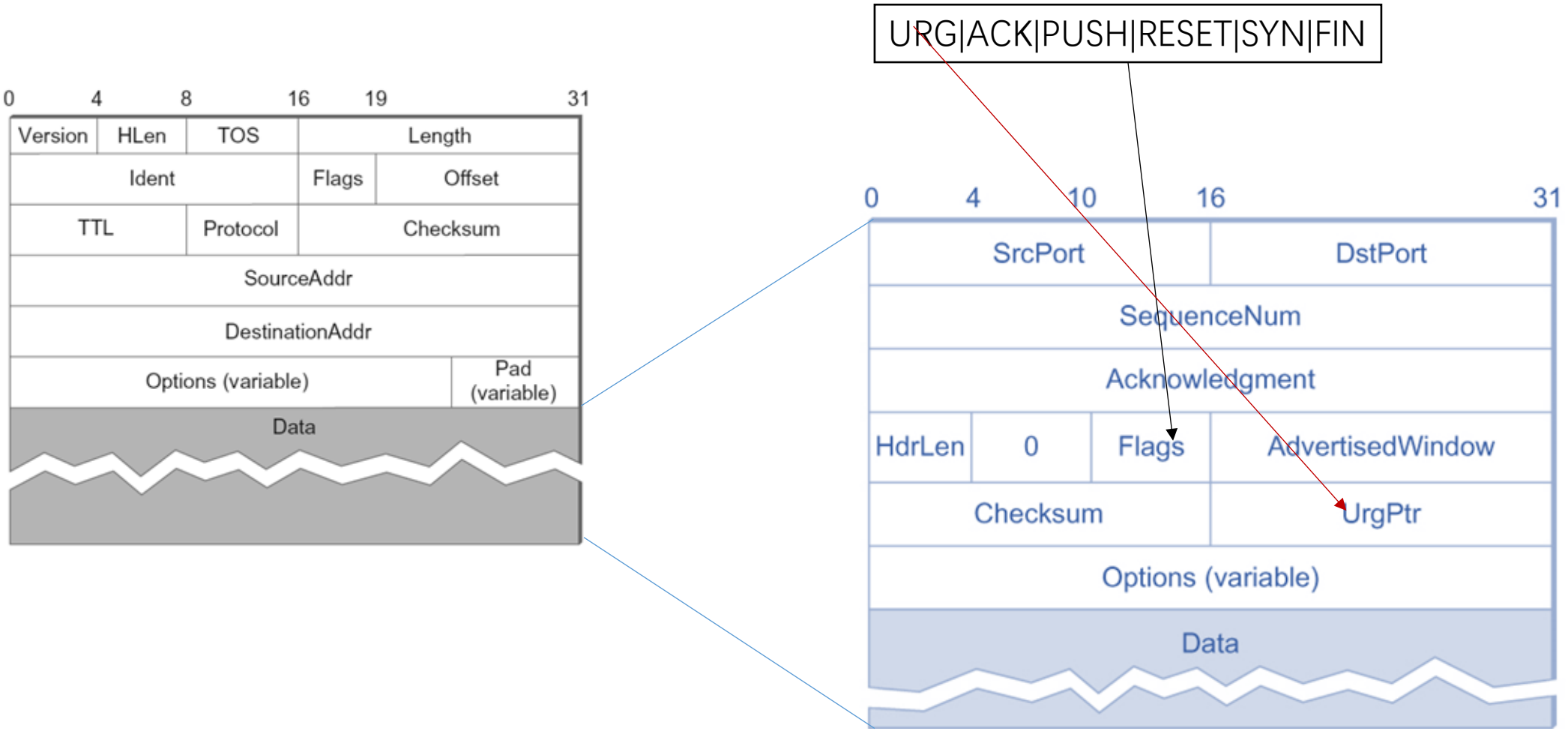
TCP Record Boundaries

- Problem: TCP is a byte stream, how to specify a message in the stream
  - UDP: one message per frame

- Solution
  - Sentinel-based Checking
  - URG flag
  - PUSH flag

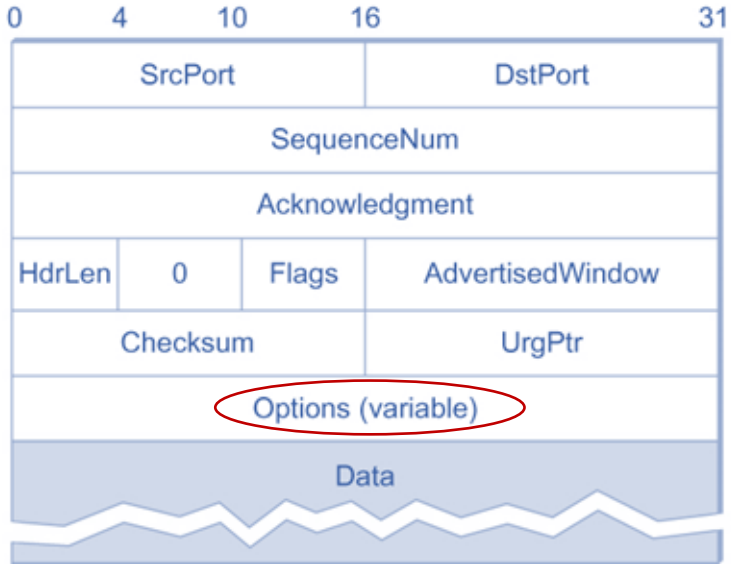
TCP will inform the application that a URG/PUSH segment is received

TCP: Header


URG|ACK|PUSH|RESET|SYN|FIN
0 4 8 16 19 31
Version
HLen
TOS
Length
Ident
Flags
Offset
TTL
Protocol
Checksum
SourceAddr
DestinationAddr
Options (variable)
Pad (variable)
Data
0 4 10 16 31
SrcPort
DstPort
SequenceNum
Acknowledgment
HdrLen
0
Flags
AdvertisedWindow
Checksum
UrgPtr
Options (variable)
Data

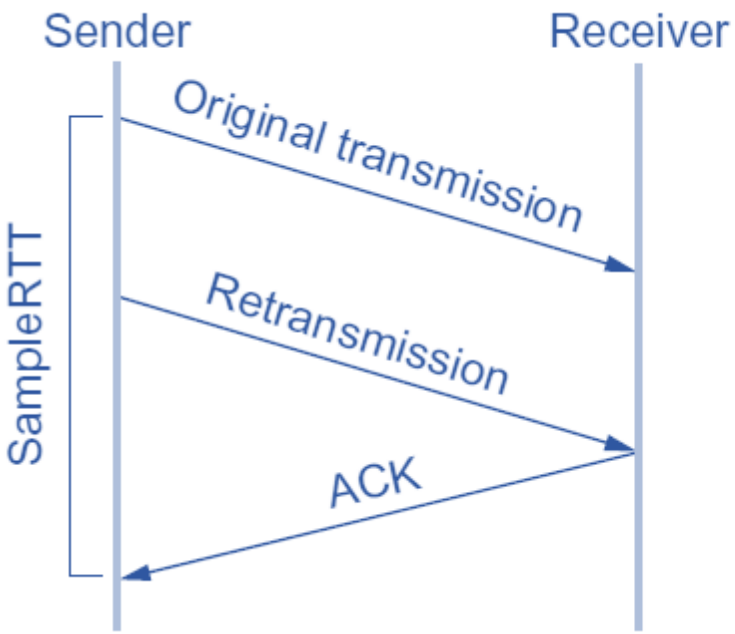
TCP Extensions

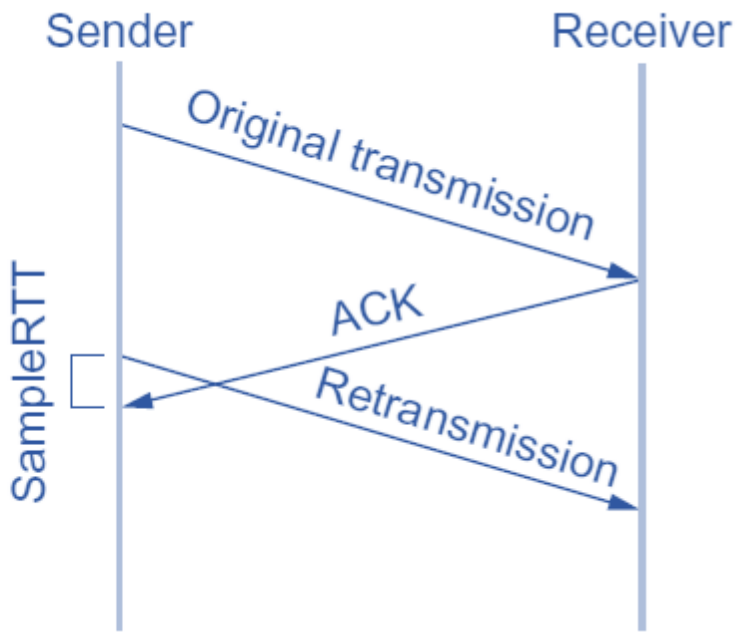
- TCP keeps evolving to incorporate high performance network



Sliding Window in TCP: Adaptive Timeout

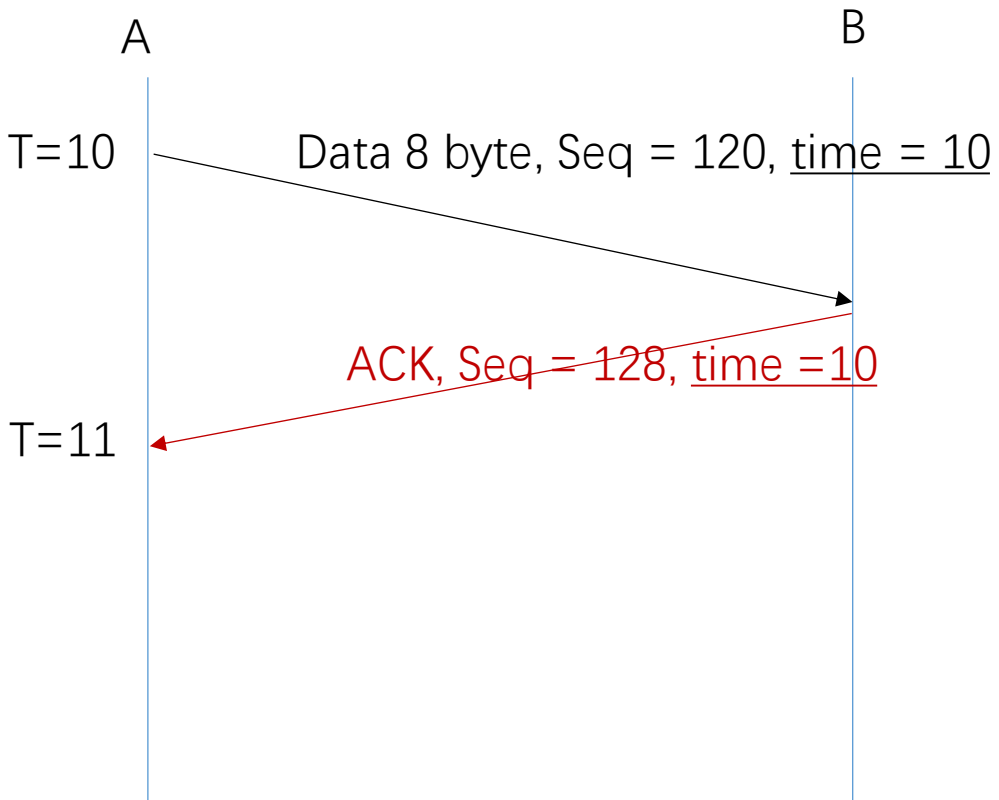
- Original Algorithm
  - Mea
  - Con
    -
    -
  - Set
    -





TCP Extensions

1. Better Estimation of RTT
   - Introduce 32-bit timestamp



TCP Extensions

2. Sequence Number Wraparound:

sequence number can be easily used up in high speed network (too many bytes are out per unit time)

- Use Seq + Timestamp to identify wraparound

Table 5.1 Time Until 32-Bit Sequence Number Space Wraps Around

Bandwidth	Time until Wraparound
T1 (1.5 Mbps)	6.4 hours
Ethernet (10 Mbps)	57 minutes
T3 (45 Mbps)	13 minutes
Fast Ethernet (100 Mbps)	6 minutes
OC-3 (155 Mbps)	4 minutes
OC-12 (622 Mbps)	55 seconds
OC-48 (2.5 Gbps)	14 seconds

TCP Extensions

3. Scaling advertised window

Advertised Window is not large enough for the Rx buffer

- Advertised Window (16bits) + Scaling Factor (0 to 14)
  - Negotiated in SYN packet

Table 5.2 Required Window Size for 100-ms RTT

Bandwidth	Delay × Bandwidth Product
T1 (1.5 Mbps)	18 KB
Ethernet (10 Mbps)	122 KB
T3 (45 Mbps)	549 KB
Fast Ethernet (100 Mbps)	1.2 MB
OC-3 (155 Mbps)	1.8 MB
OC-12 (622 Mbps)	7.4 MB
OC-48 (2.5 Gbps)	29.6 MB

TCP Extensions

4. Selective ACK

- Uses optional fields in the header to acknowledge any additional blocks of received data.

- This allows the sender to retransmit just the missing segments.

Reference

- Textbook 5.2
- Silly window syndrome https://datatracker.ietf.org/doc/html/rfc813
- Karn/Partridges algorithm: https://datatracker.ietf.org/doc/html/rfc2988
- TCP Extenstions https://www.ietf.org/rfc/rfc1323.txt