CS120: Computer Networks

Lecture 16. TCP 2

Zhice Yang

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

Delay × Bandwidth

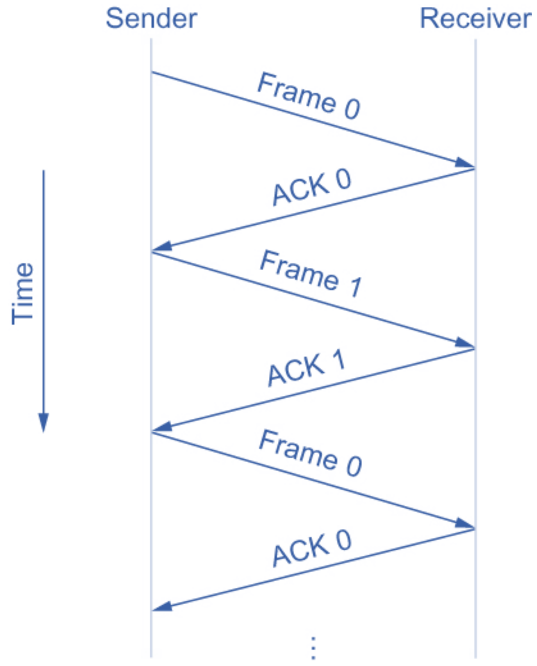
- Quantify the utilization of the link

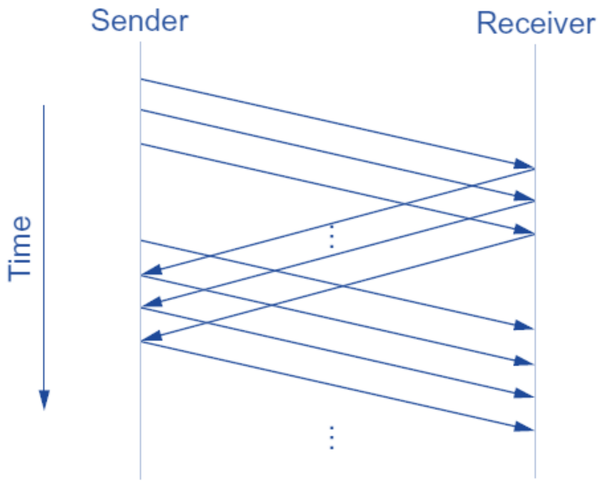
Table 1.1 Sample Delay × Bandwidth Products

Link type	Bandwidth (typical)	One-way distance (typical)	Round-trip delay	RTT × Bandwidth
Dial-up	56 kbps	10 km	87 μs	5 bits
Wireless LAN	54 Mbps	50 m	0.33 μs	18 bits
Satellite	45 Mbps	35,000 km	230 ms	10 Mb
Cross-country fiber	10 Gbps	4,000 km	40 ms	400 Mb

Delay × Bandwidth

- Efficiency Problem
  - 1.5Mbps bandwidth
  - 45ms RTT
  - 1KB per packet
    - Effective Rate = 1024*8/(1024*8/1.5M+45ms)=162kbps
- Solution
  - Pipeline
  - Full pipe
    - 1.5Mbps*45ms/1kB = 8 packets

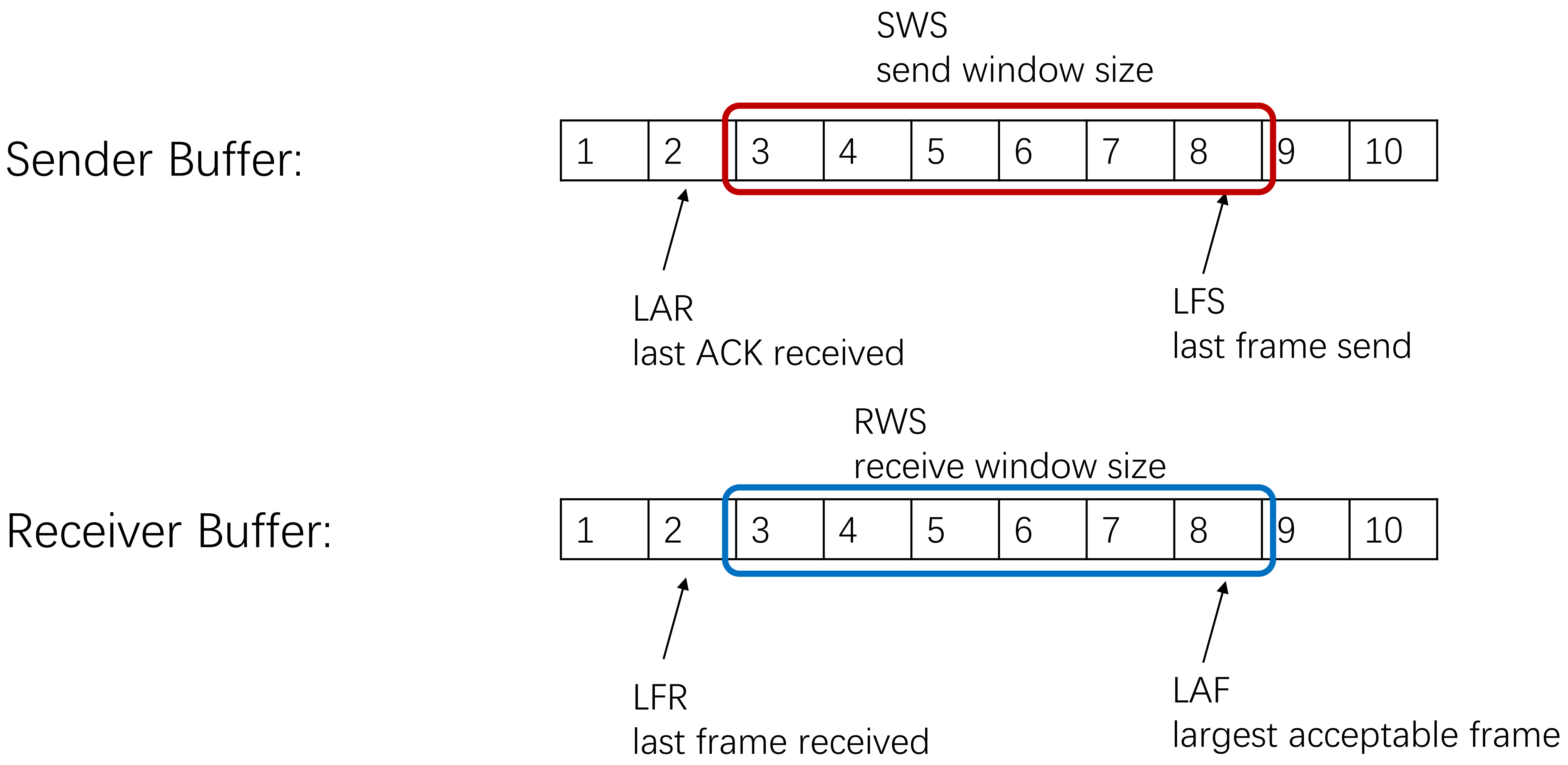




Sliding Window

- Sender Buffer
  - Retransmit
- Receiver Buffer
  - Frame order

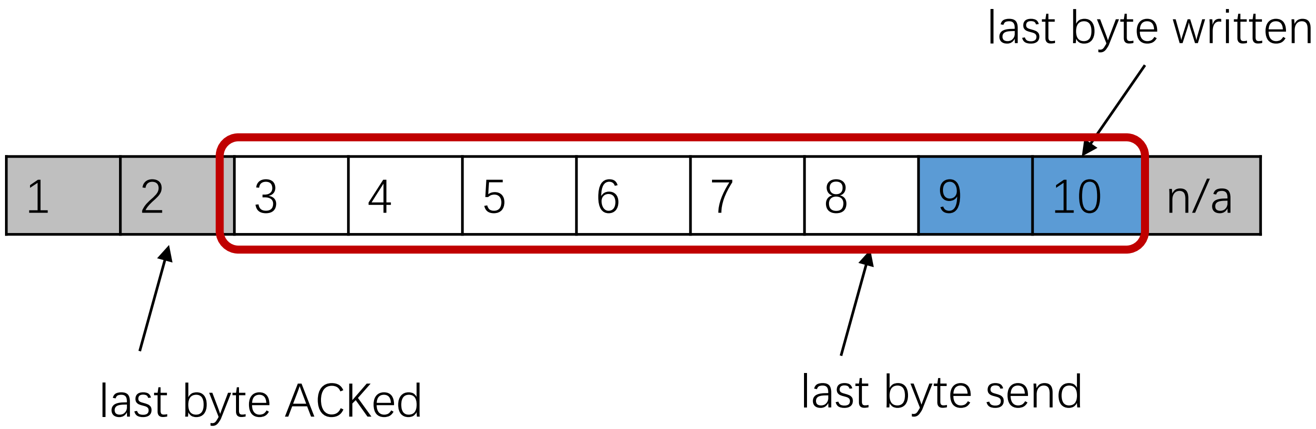
Sliding Window



Sliding Window in TCP

Sender Buffer (8 bytes):

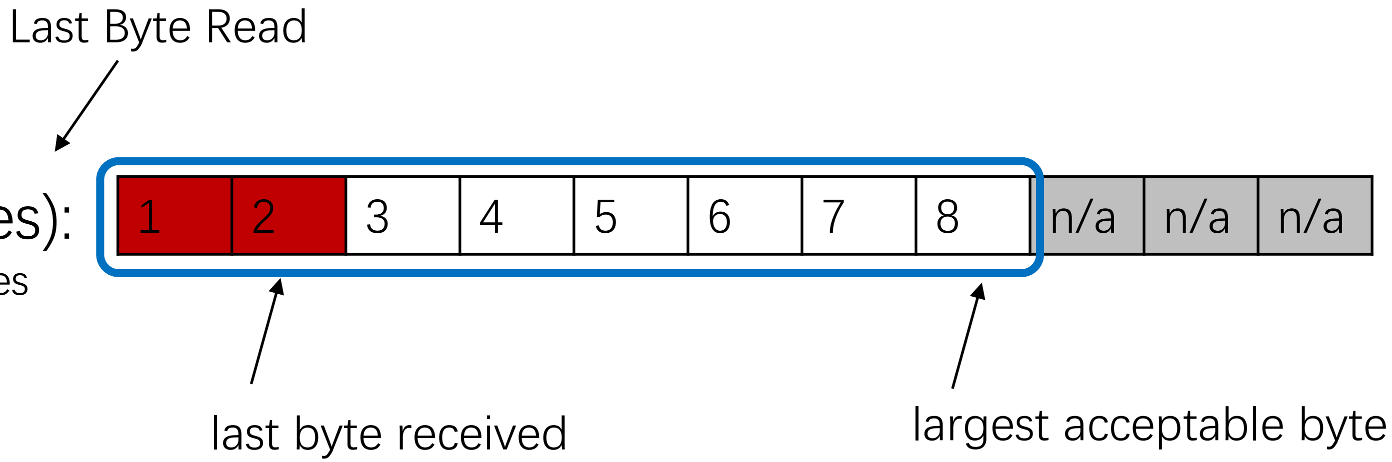
1,2 are freed buffer



Receiver Buffer (8 bytes):

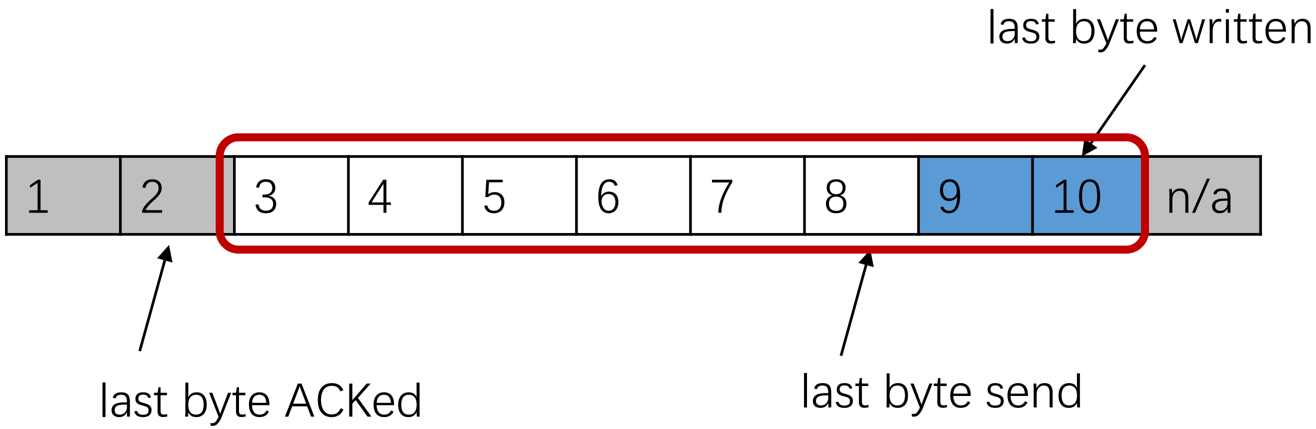
1,2 are occupied by received bytes

9,10 haven't allocated

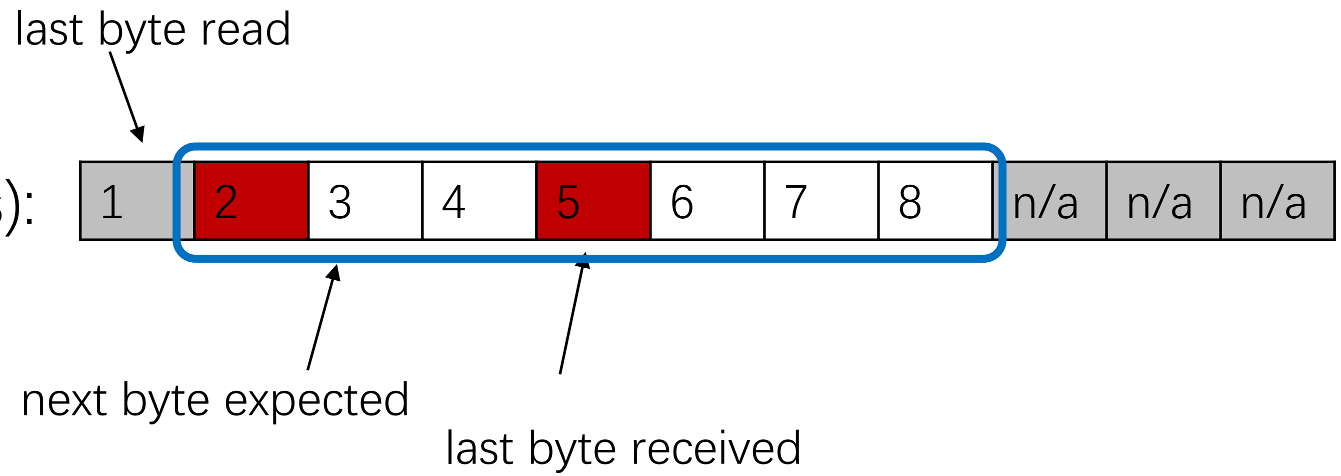


Sliding Window in TCP

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



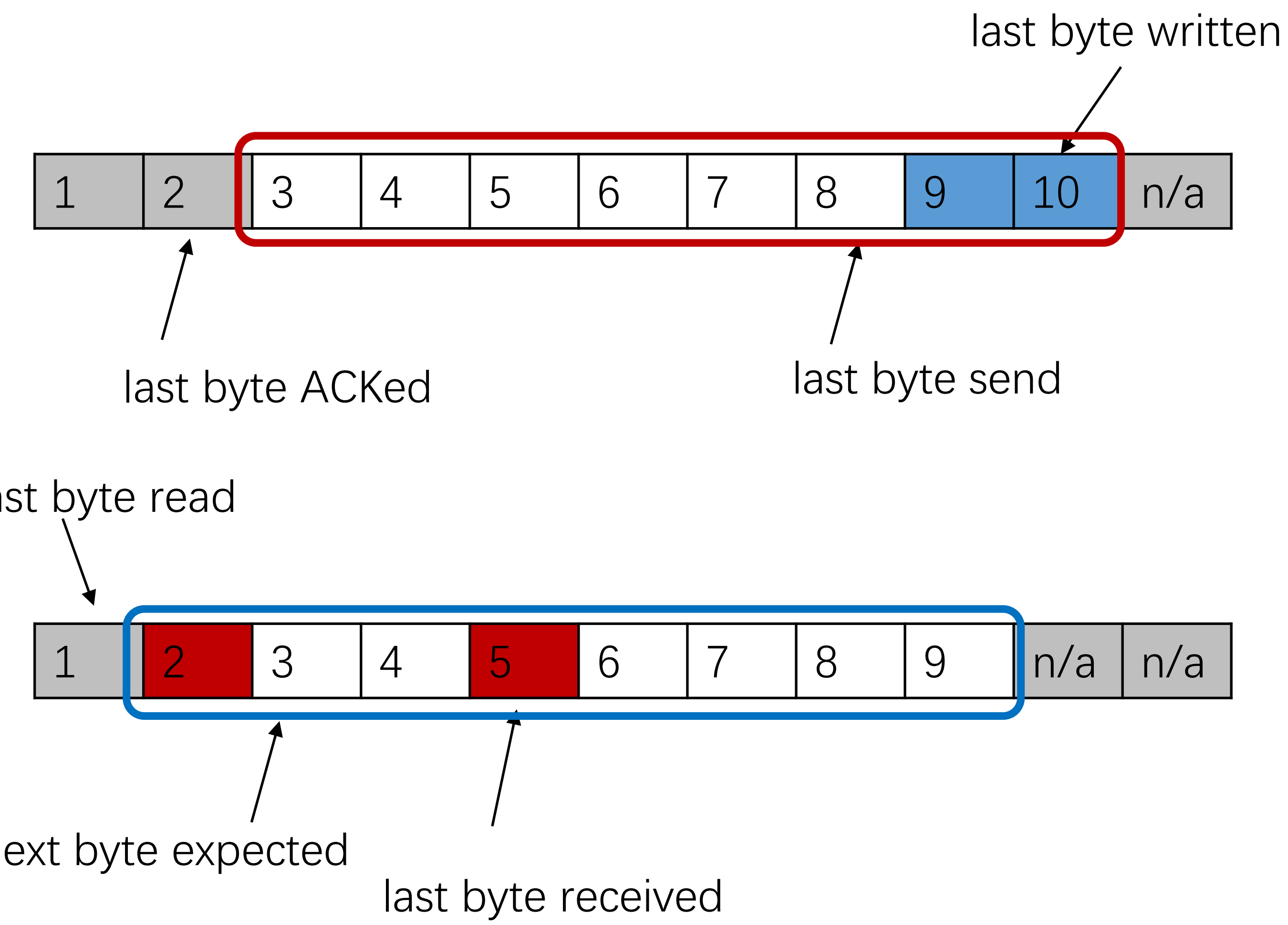
Sliding Window in TCP

Sender Buffer (8 bytes):

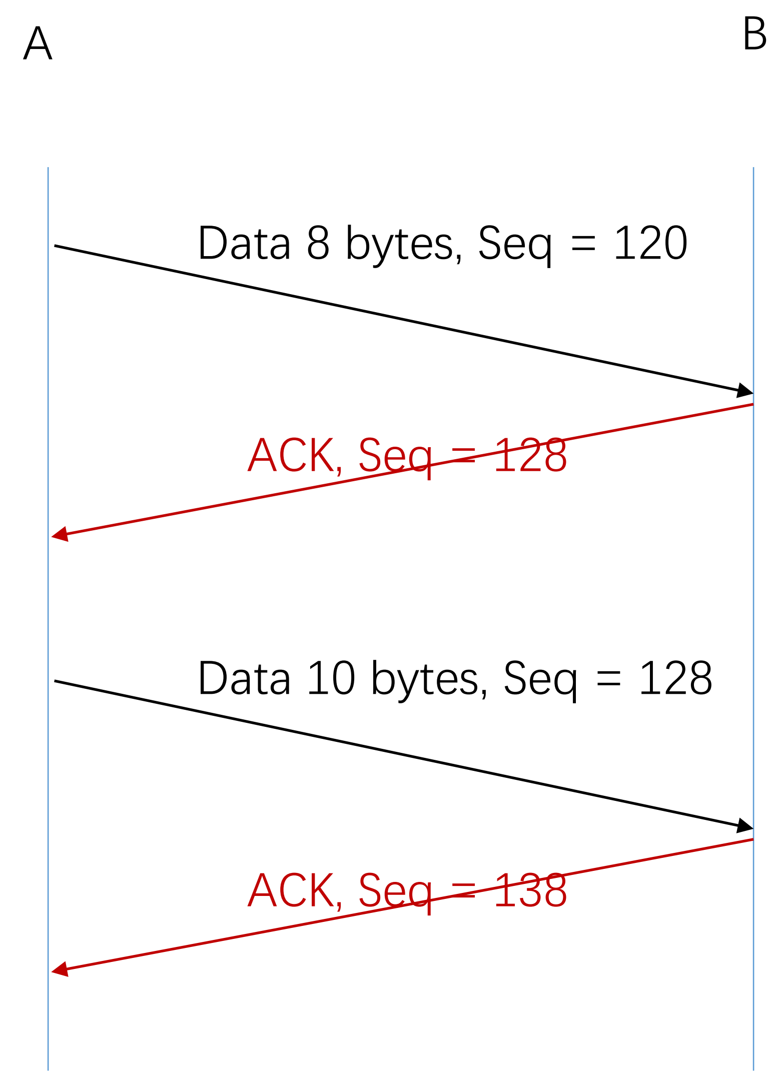
1,2 are freed buffer

Receiver Buffer (8 bytes):

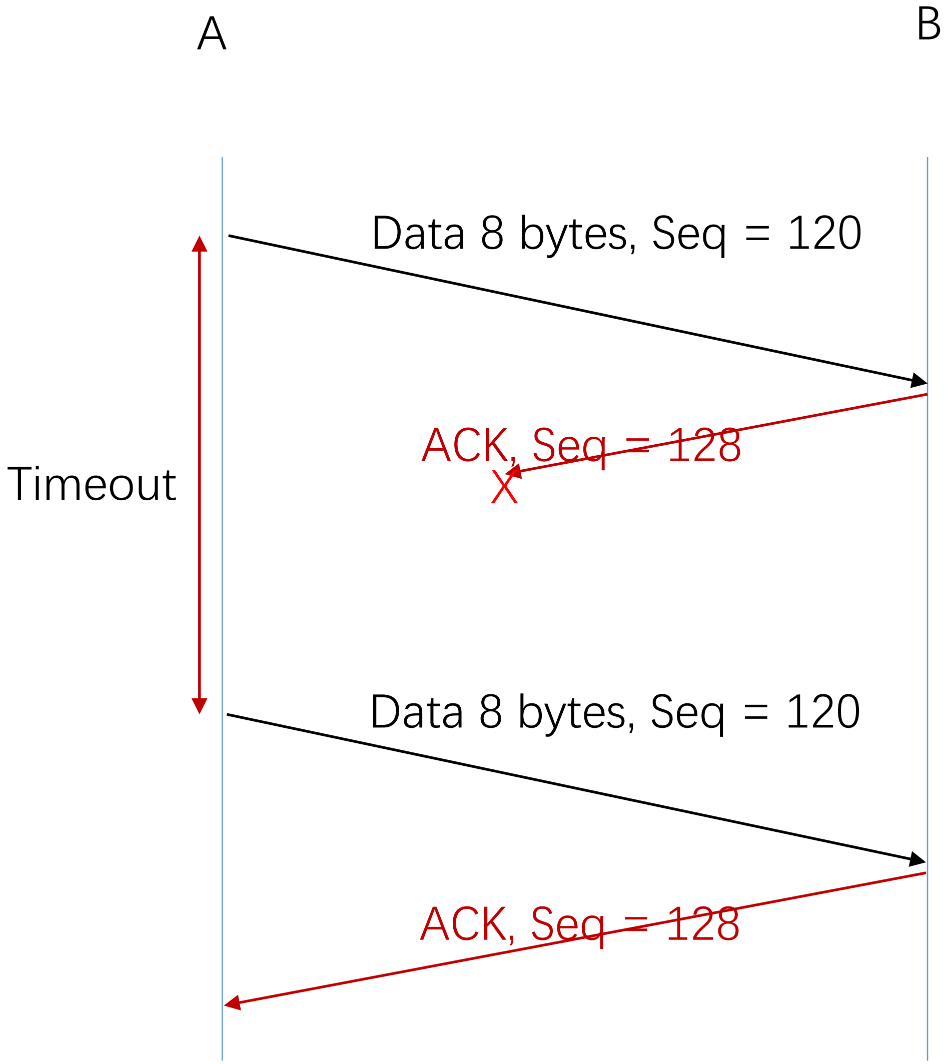
2,5 are occupied by received bytes
1 is read and freed by receiver
9 is allocated for receiving



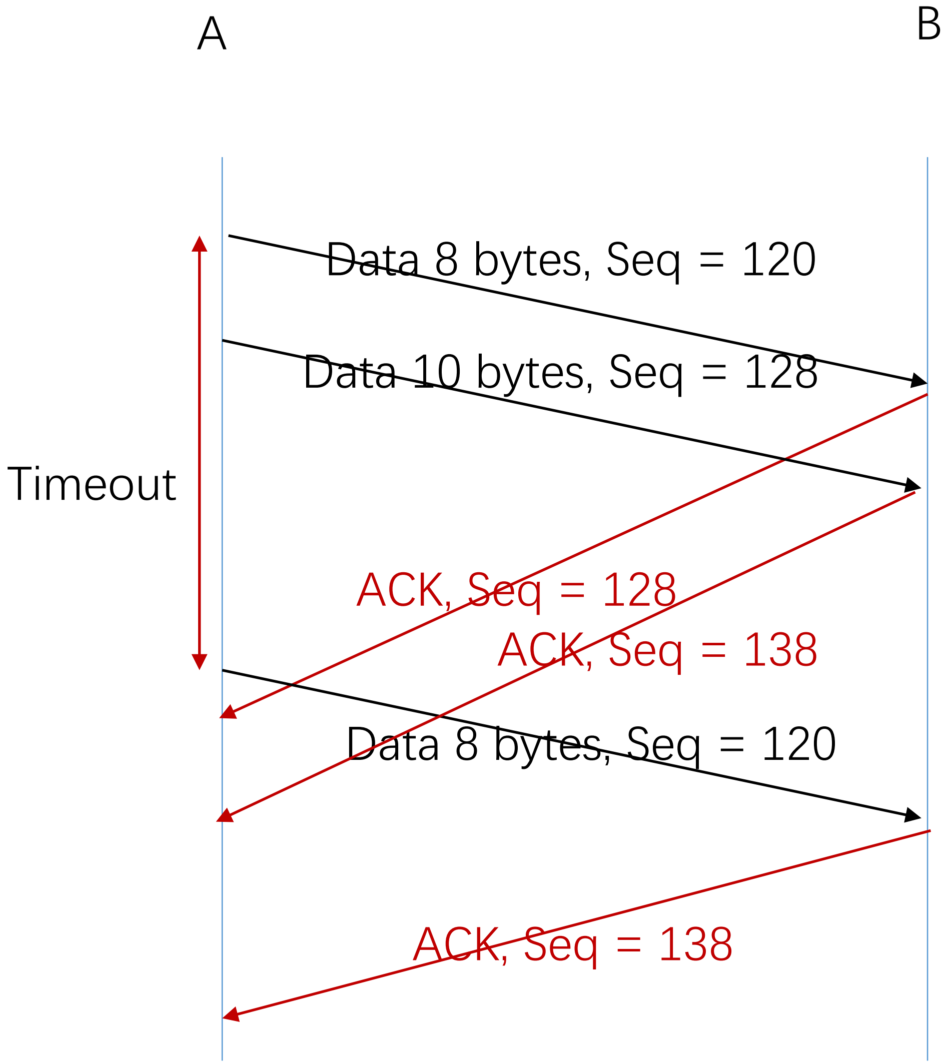
Sliding Window in TCP: Examples



Sliding Window in TCP: Examples

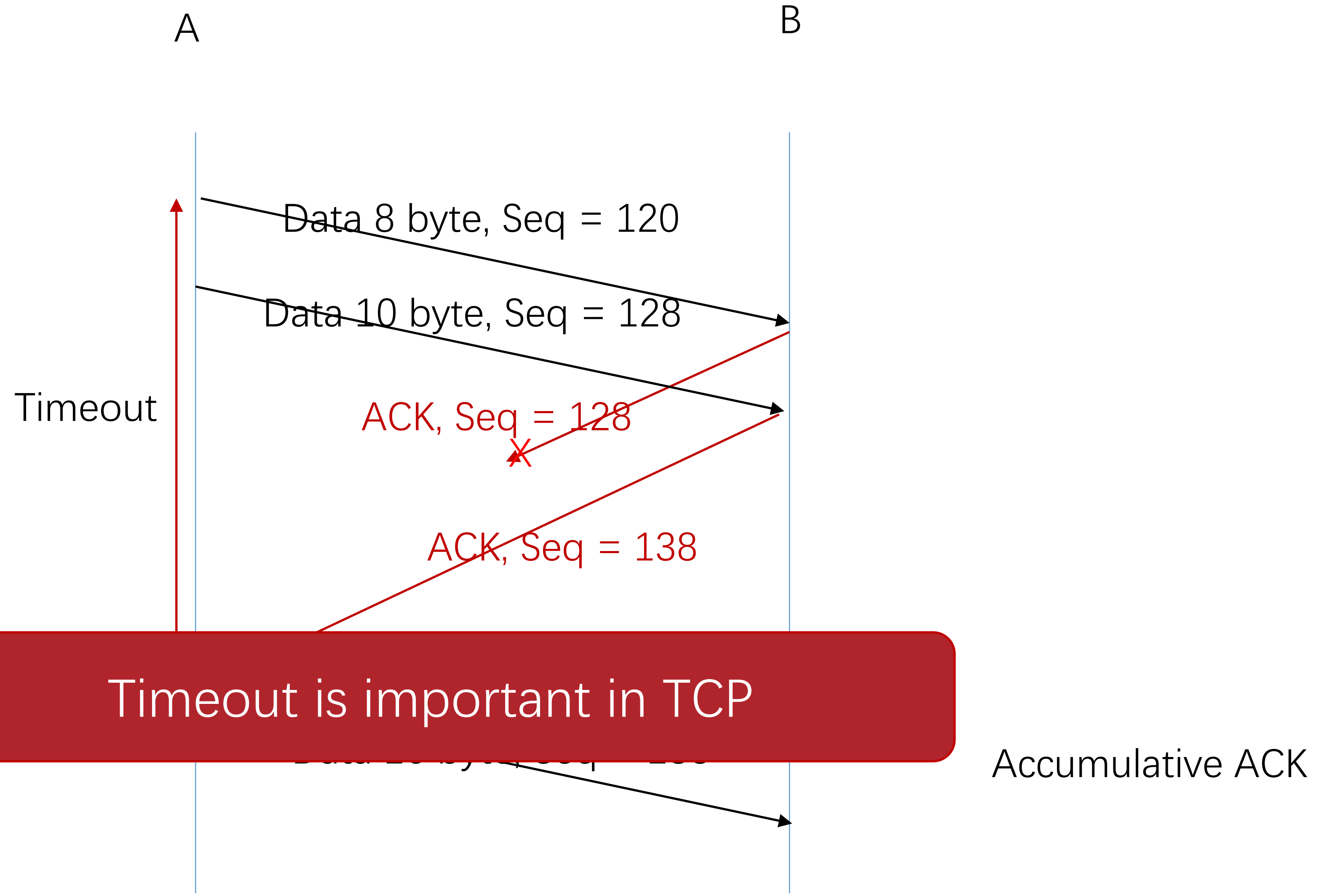


Sliding Window in TCP: Examples



Accumulative ACK

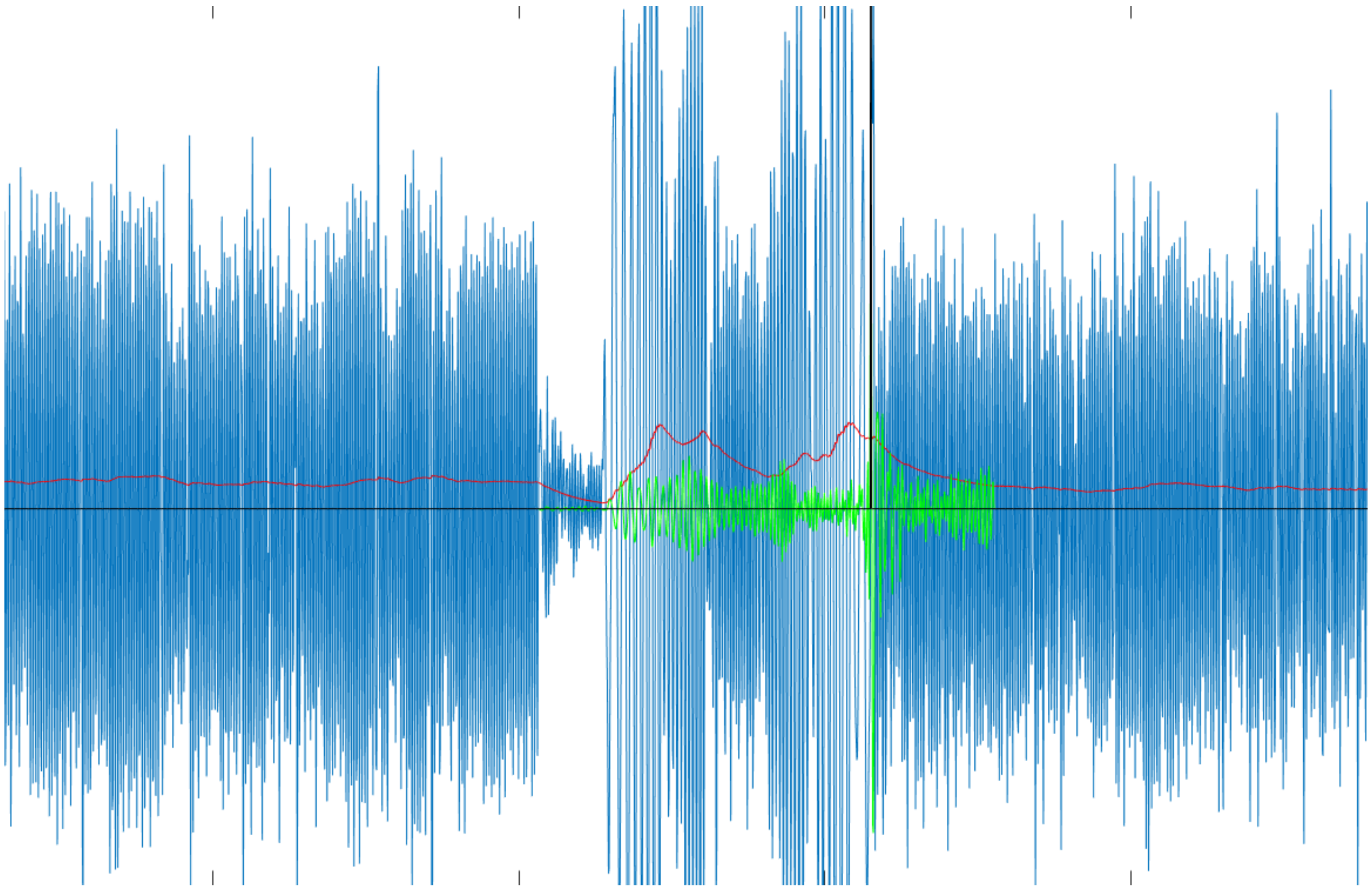
Sliding Window in TCP: Examples



Timeout is important in TCP

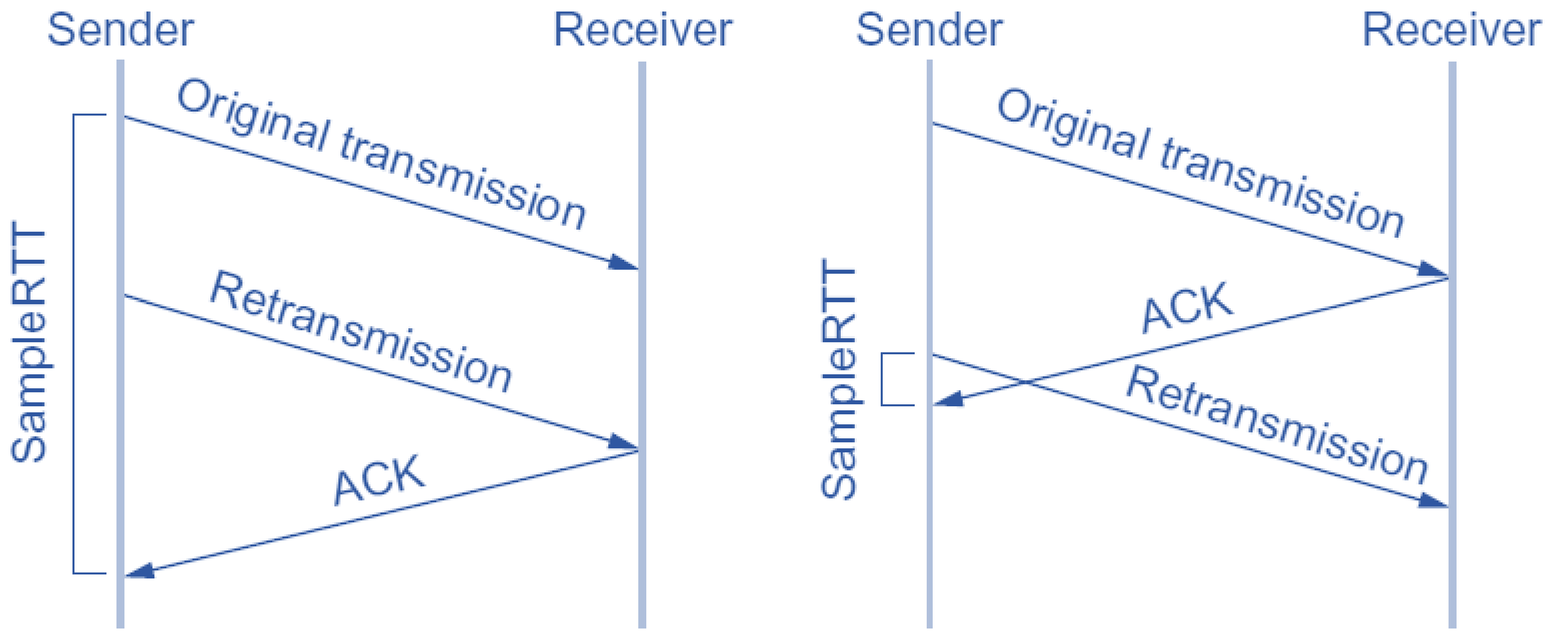
Sliding Window in TCP: Adaptive Timeout

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



Sliding Window in TCP: Adaptive Timeout

- Original Algorithm
  - Mea
  - Cor
    -
    -
  - Set
    -



Sliding Window in TCP: Adaptive Timeout

- Karn/Partridge Algorithm
  - Do not sample RTT when retransmitting
  - Double timeout after each retransmission
    - (But do not increase sliding window)
    - Retransmission => network is congested, should slow down
    - Congestion Control (will introduce in next few lectures)
- The estimation for RTT is not accurate
  - RTT is a critical signal for identifying network congestion

Sliding Window in TCP: Adaptive Timeout

- Jacobson/Karels Algorithm
  - Take RTT variance into account

  Difference = SampleRTT − EstimatedRTT

  EstimatedRTT = EstimatedRTT + (δ∗Difference)

  Deviation = Deviation + δ∗(|Difference| − Deviation)
    - δ is typically set to 1/8

  TimeOut = μ ∗ EstimatedRTT + φ ∗ Deviation
    - μ is typically set to 1 and φ is set to 4

Sliding Window in TCP: Adaptive Timeout

- Jacobson/Karels Algorithm Implementation

$$\text{Difference} = \text{SampleRTT} - \text{EstimatedRTT}$$
$$\text{EstimatedRTT} = \text{EstimatedRTT} + (\delta * \text{Difference})$$
$$\text{Deviation} = \text{Deviation} + \delta * (|\text{Difference}| - \text{Deviation})$$
$$\text{TimeOut} = \mu * \text{EstimatedRTT} + \varphi * \text{Deviation}$$

$$\Downarrow$$

```
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

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

TCP Flow Control

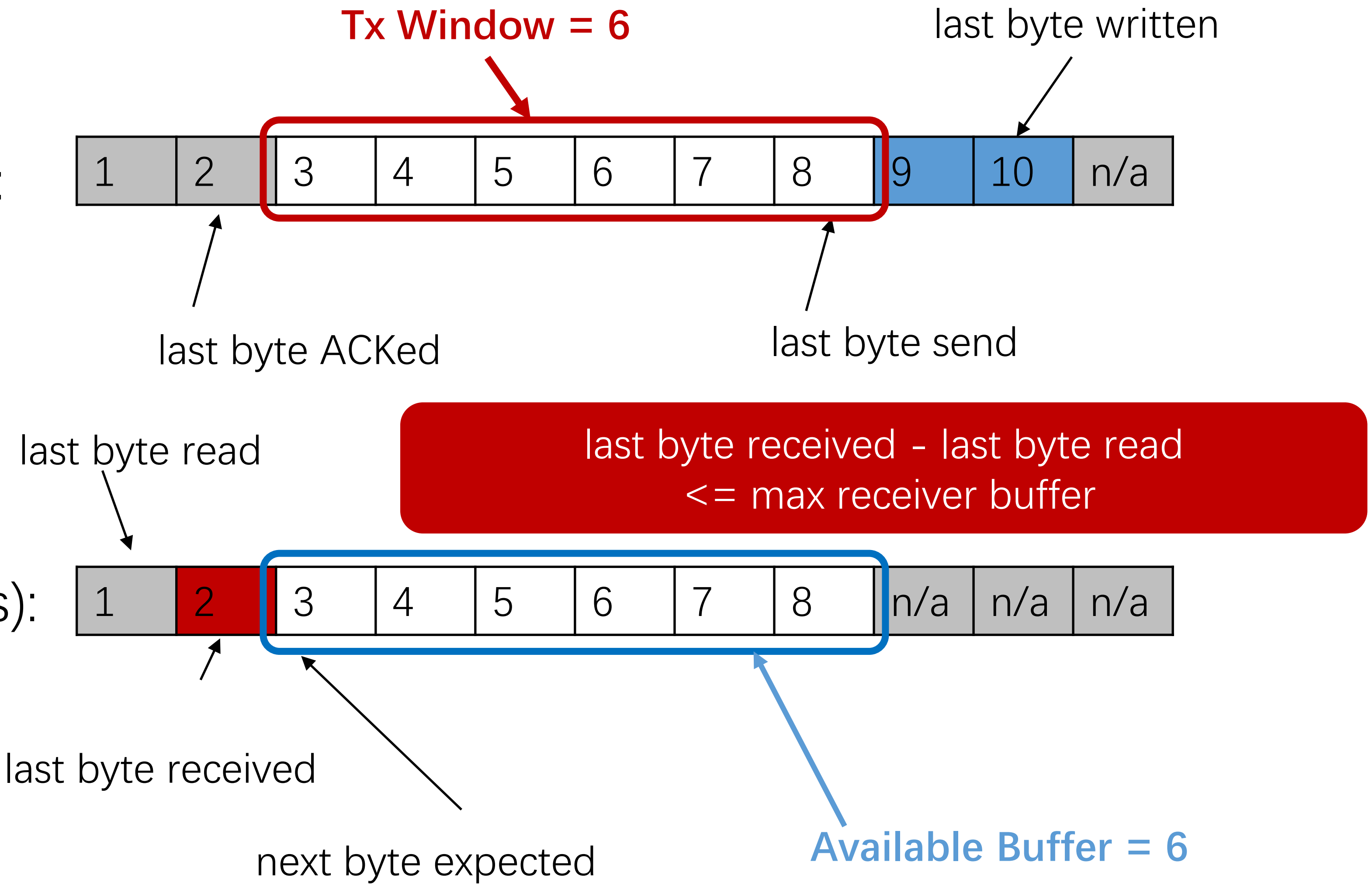
Sender Buffer (8 bytes):

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

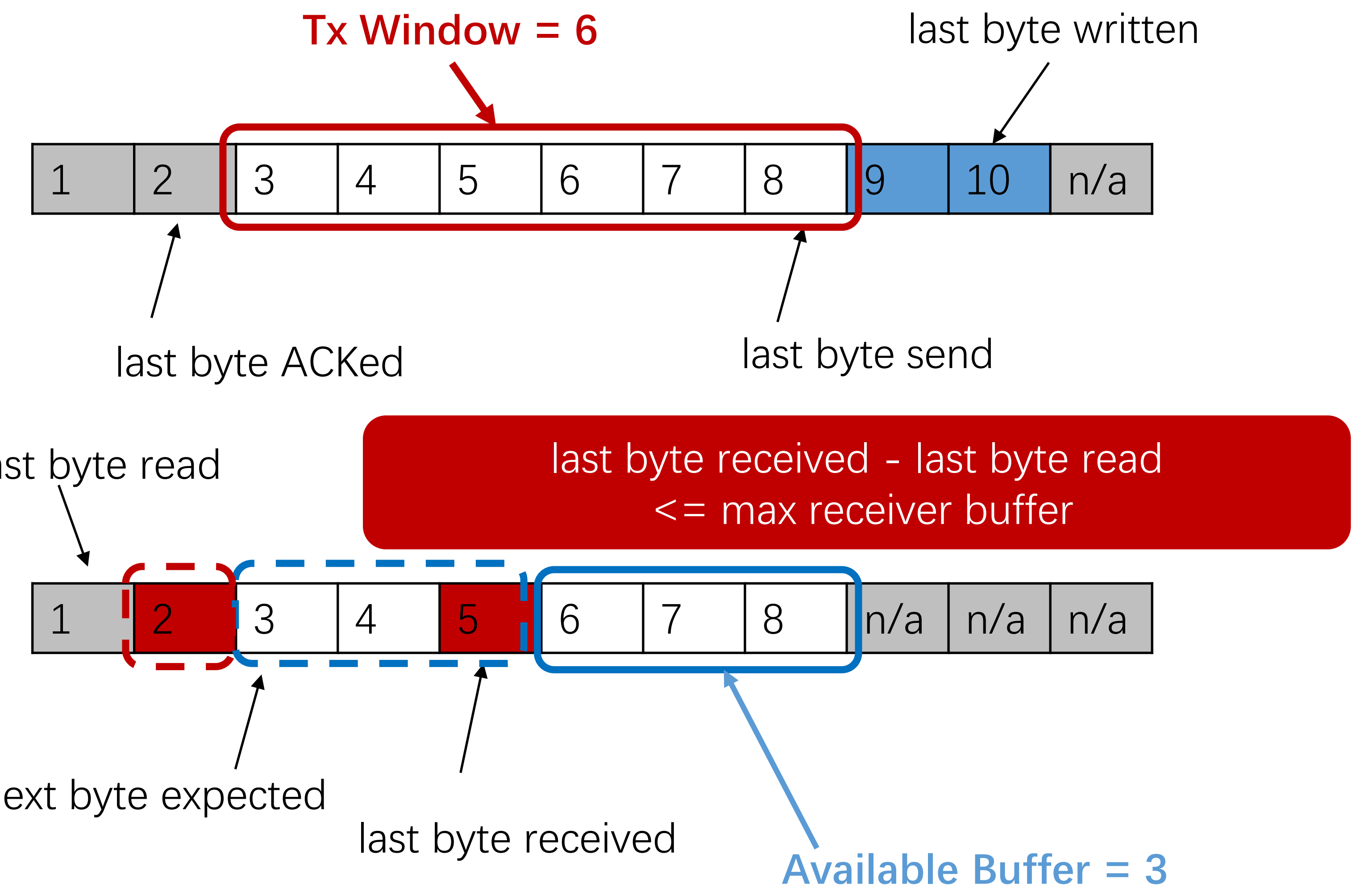
Sender Buffer (8 bytes):

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

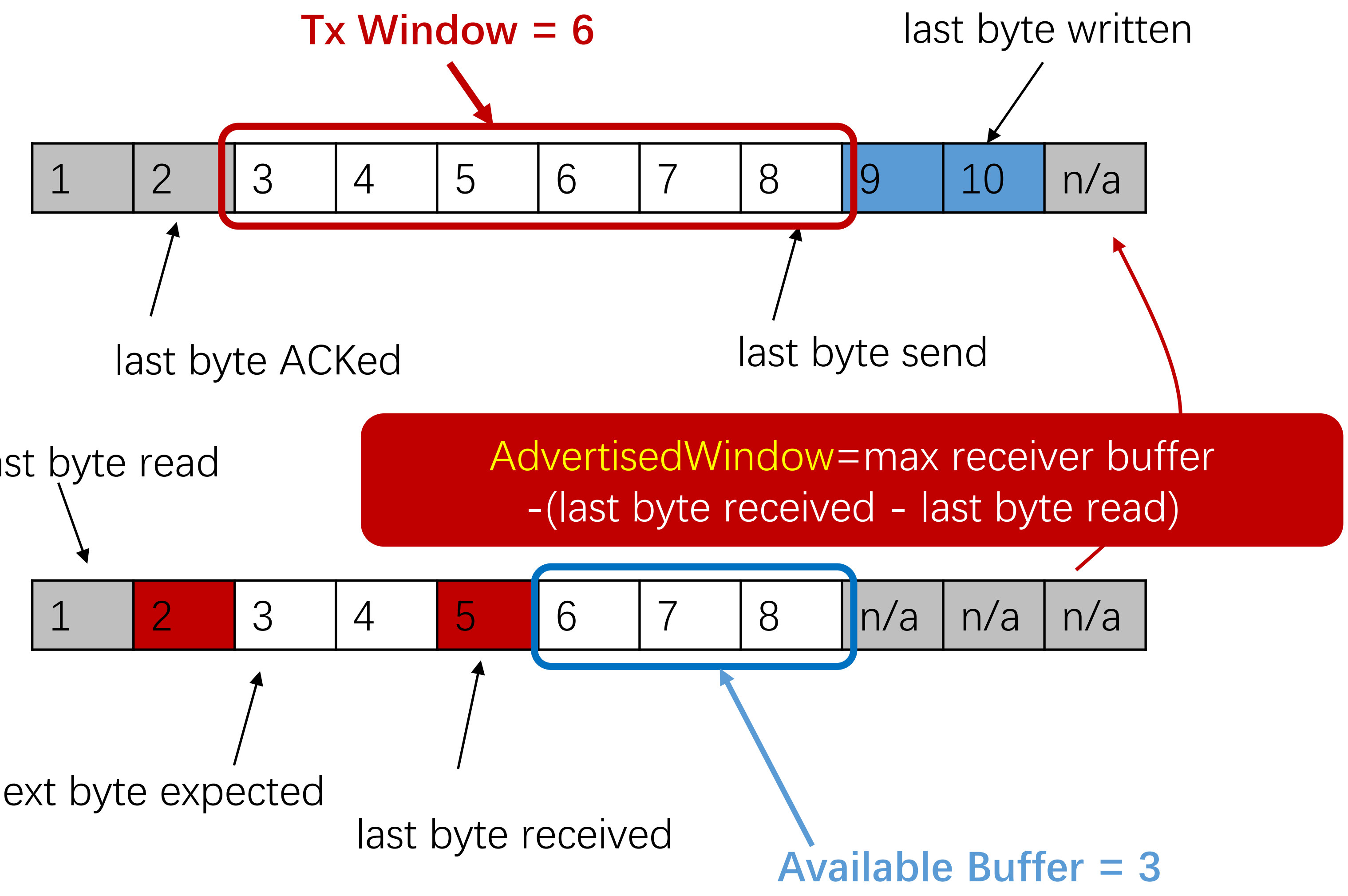
Sender Buffer (8 bytes):

1,2 are freed buffer

Receiver Buffer (7 bytes):

2,5 are occupied by received bytes

1 is read and freed by receiver

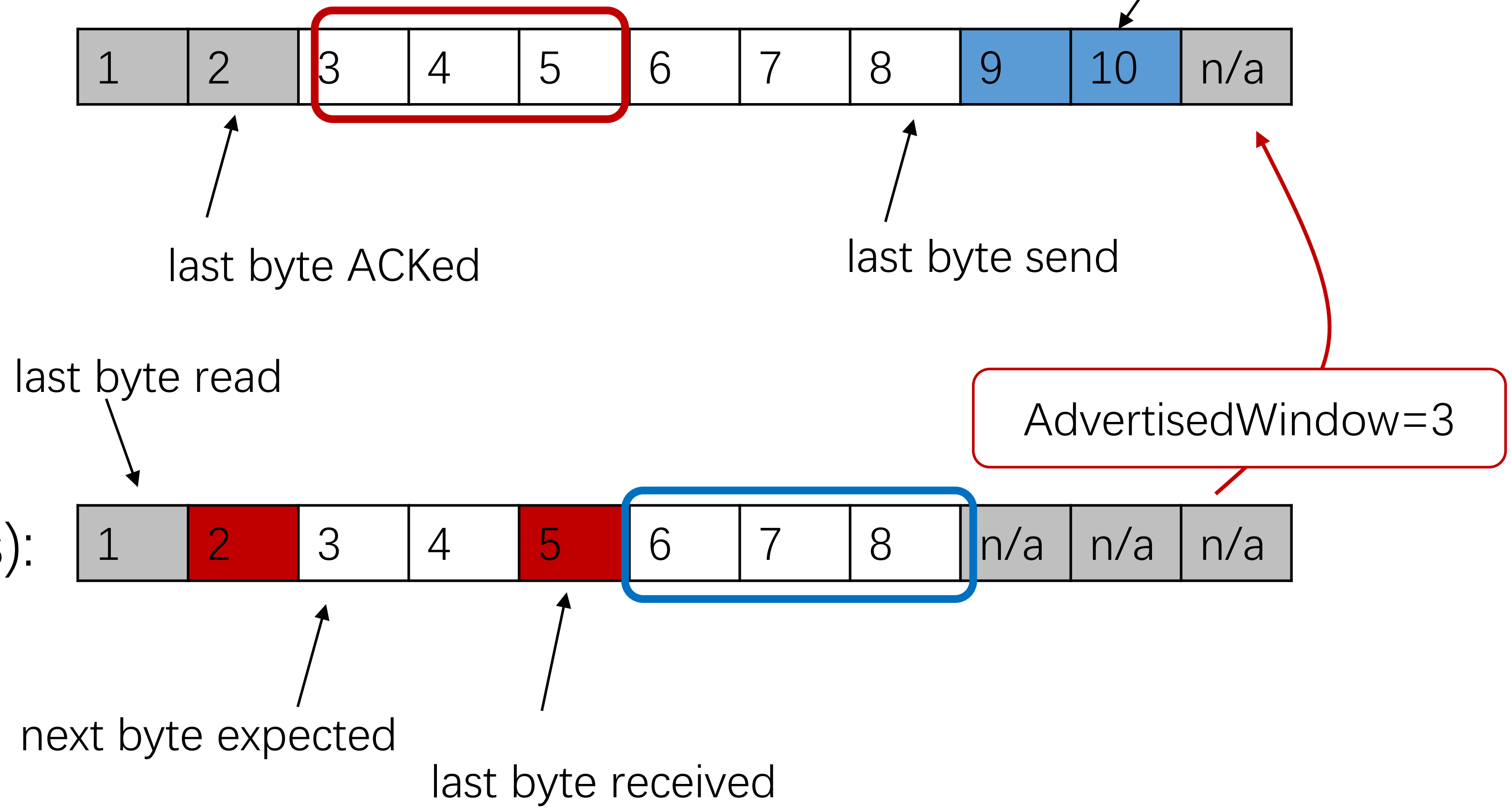


TCP Flow Control

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

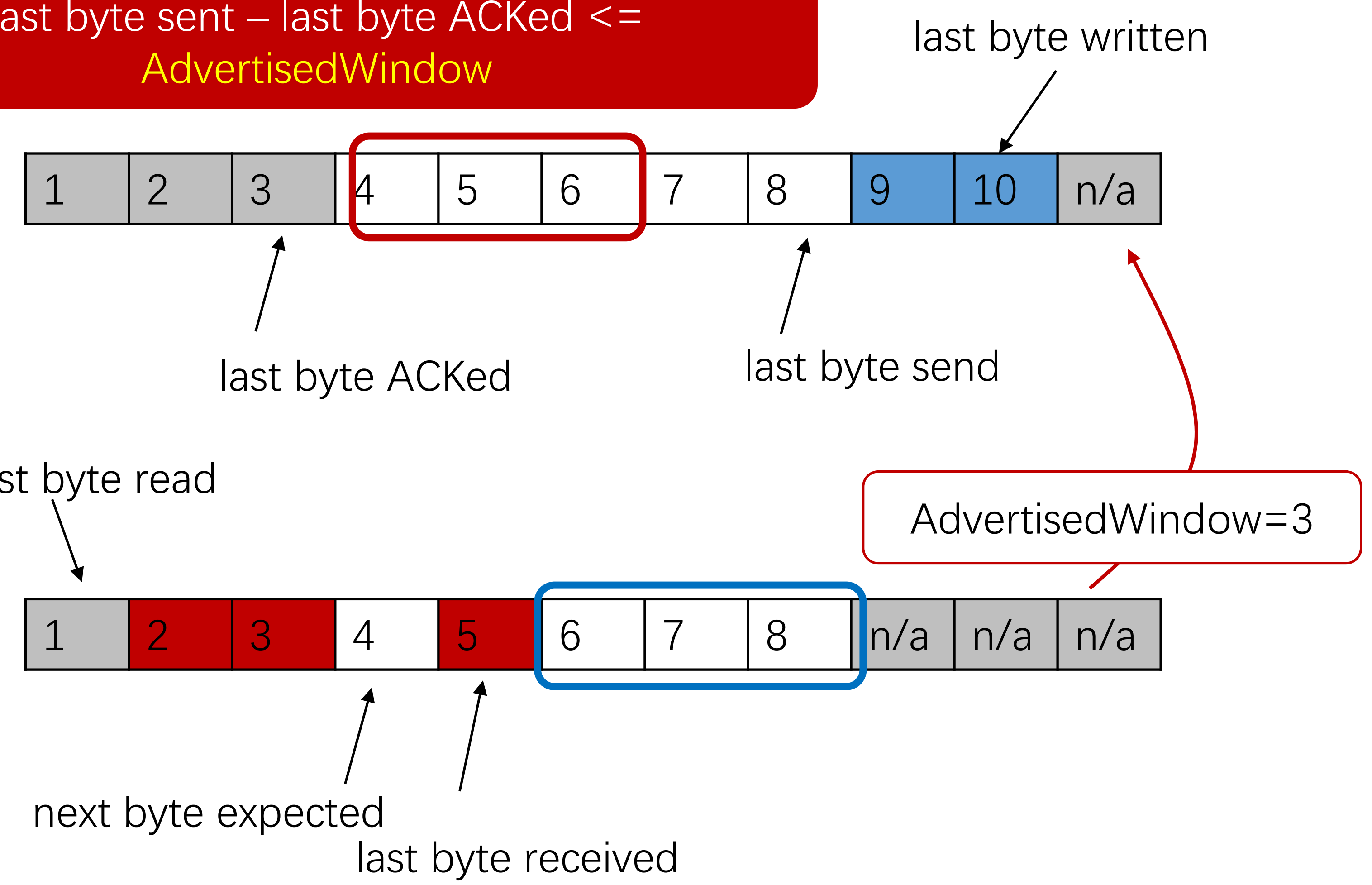


TCP Flow Control

last byte sent – last byte ACKed <= AdvertisedWindow

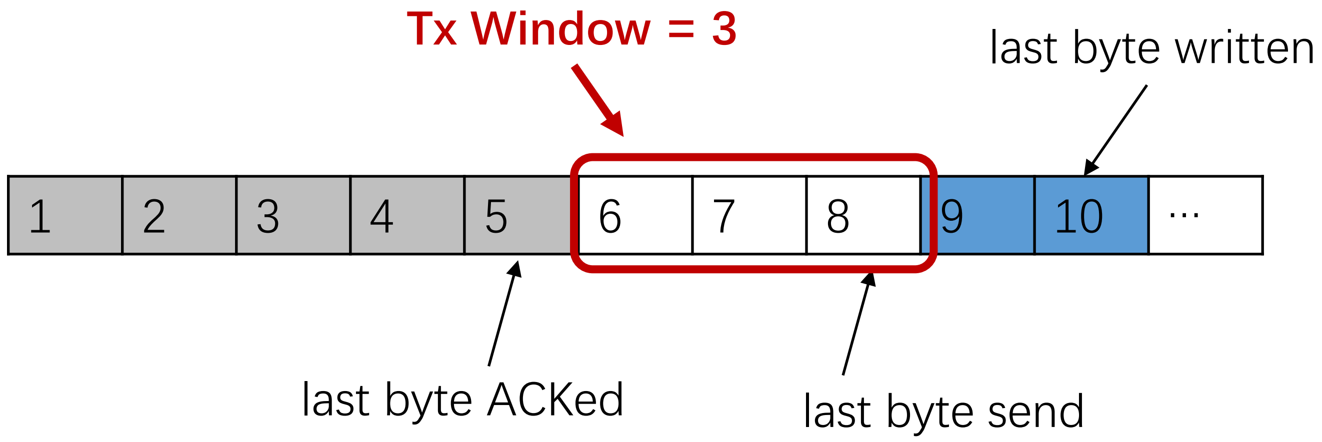
Sender Buffer (8 bytes):
1,2 are freed buffer

Receiver Buffer (7 bytes):
2,5 are occupied by received bytes
1 is read and freed by receiver

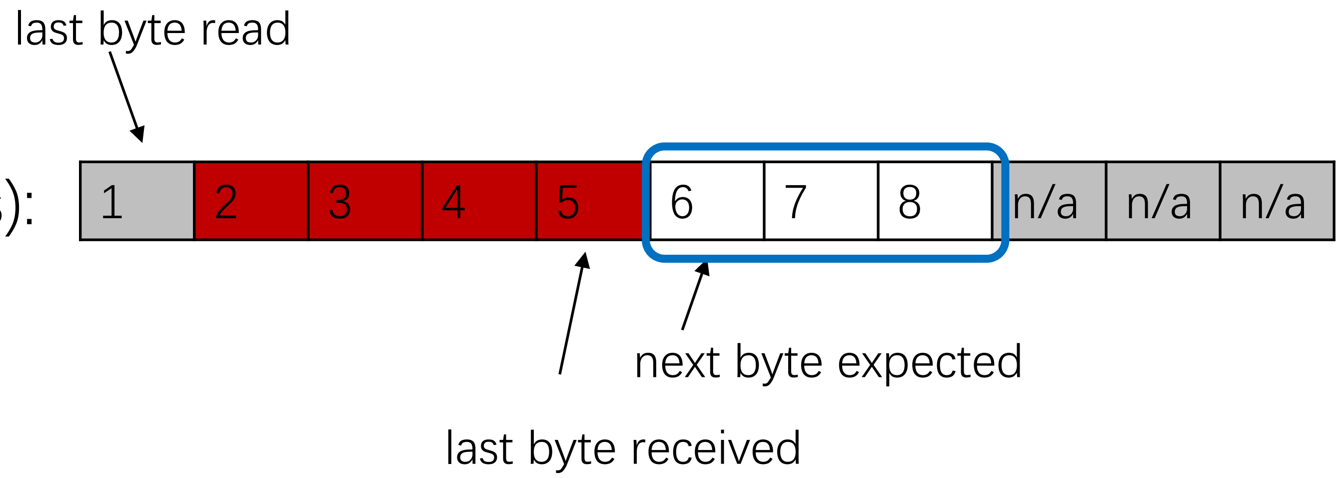


TCP Flow Control

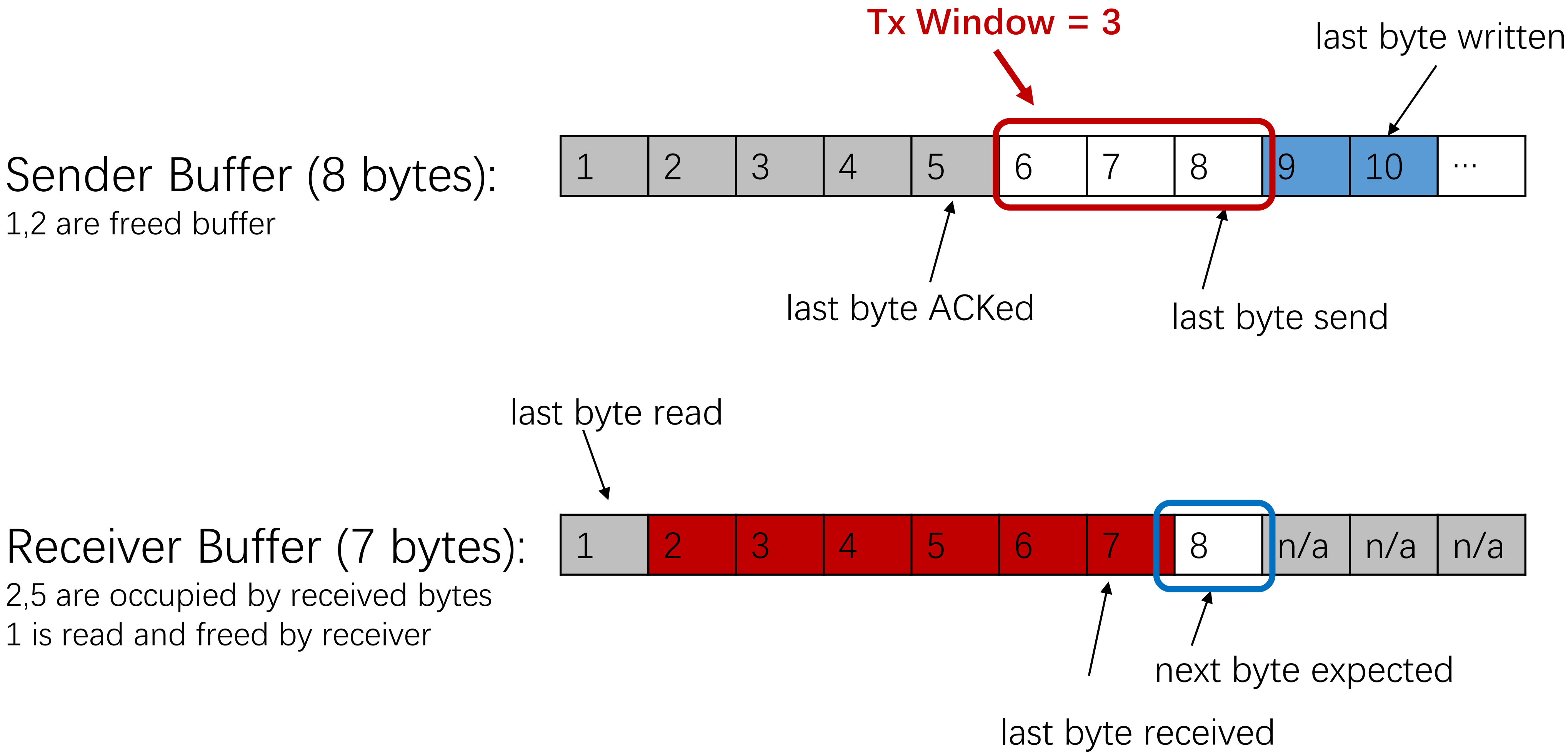
Sender Buffer (8 bytes):
1,2 are freed buffer



Receiver Buffer (7 bytes):
2,5 are occupied by received bytes
1 is read and freed by receiver



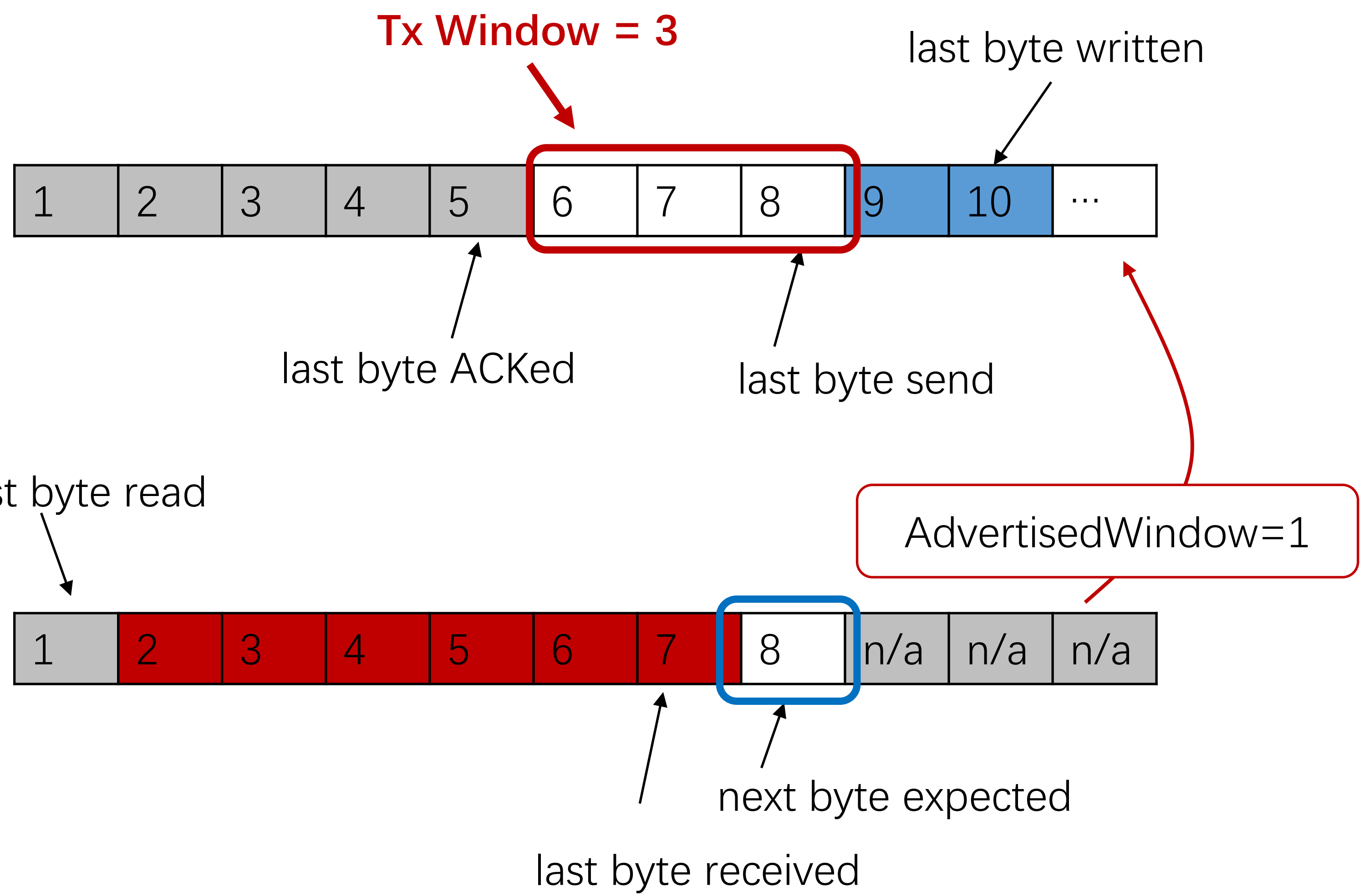
TCP Flow Control



TCP Flow Control

Sender Buffer (8 bytes):
1,2 are freed buffer

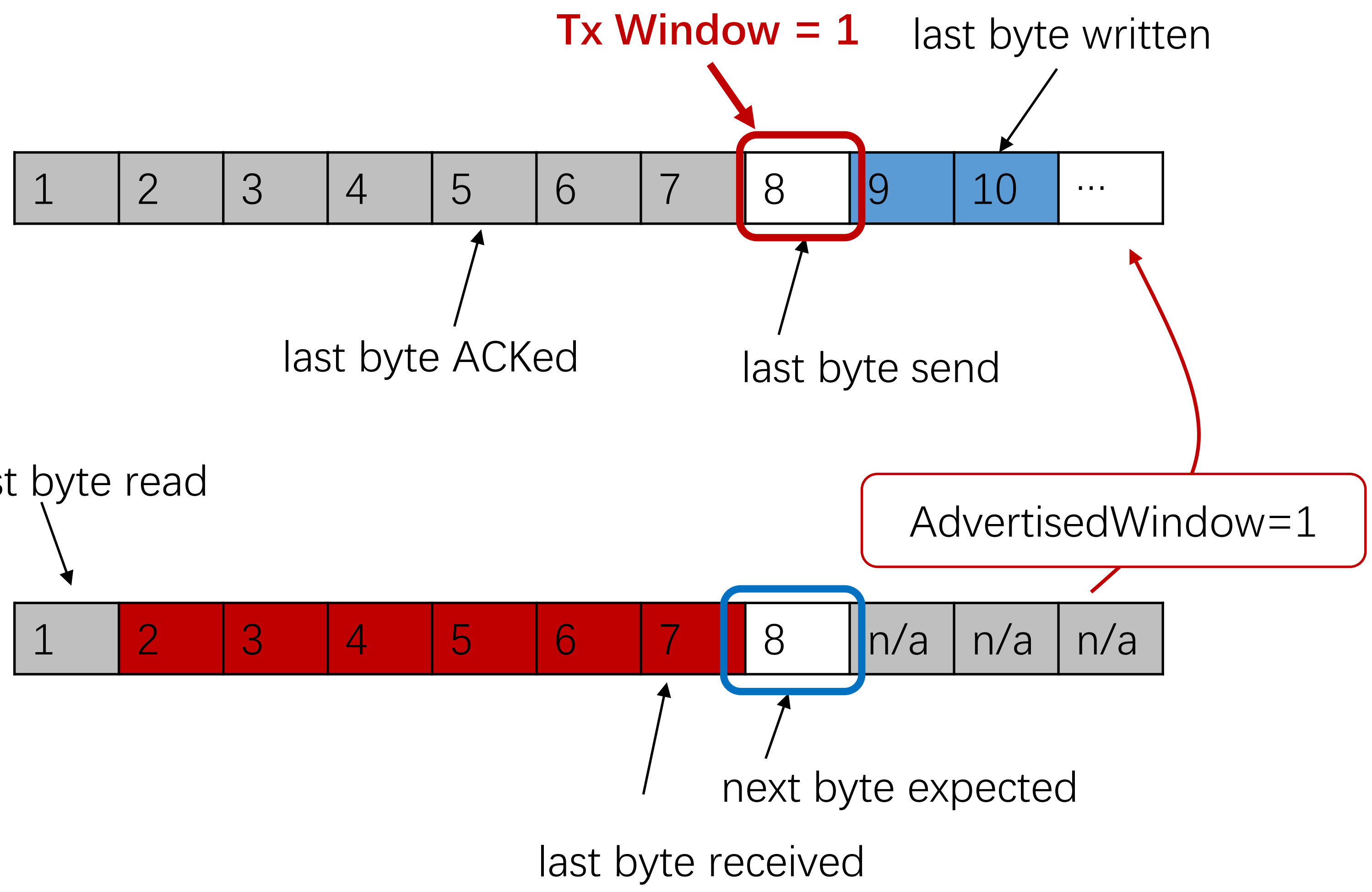
Receiver Buffer (7 bytes):
2,5 are occupied by received bytes
1 is read and freed by receiver



TCP Flow Control

Sender Buffer (8 bytes):
1,2 are freed buffer

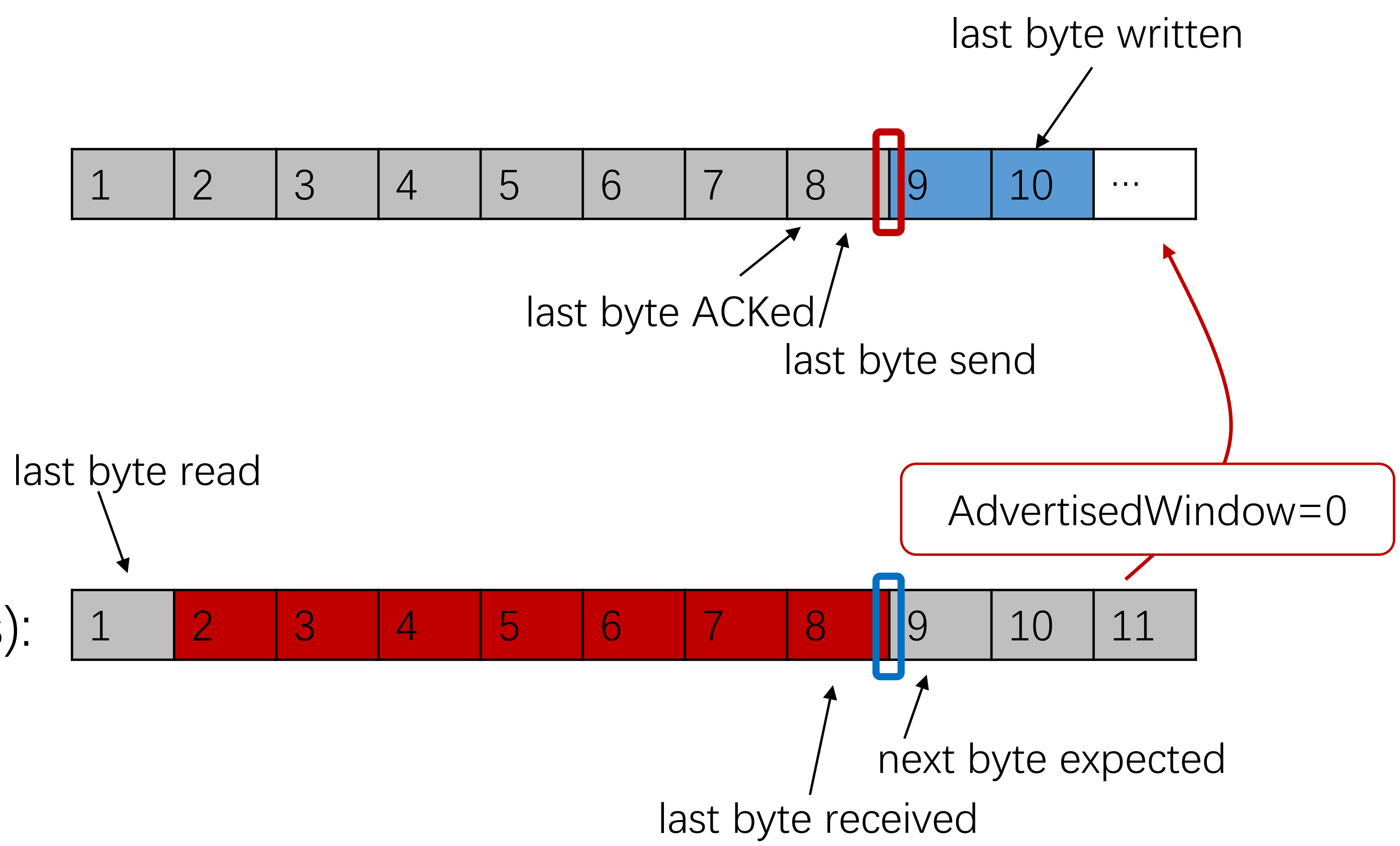
Receiver Buffer (7 bytes):
2,5 are occupied by received bytes
1 is read and freed by receiver



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

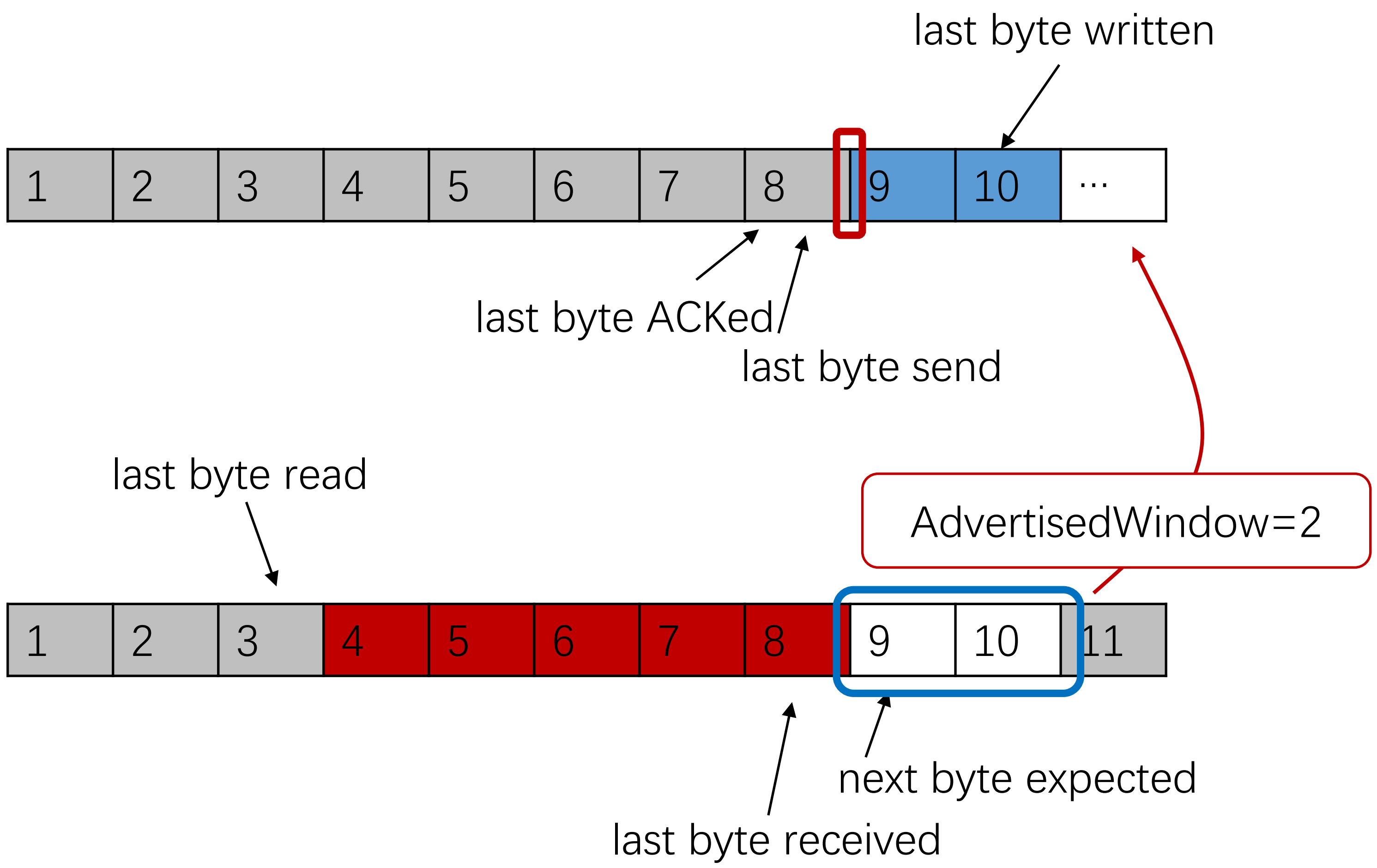
Sender Buffer (8 bytes):

1,2 are freed buffer

Receiver Buffer (7 bytes):

2,5 are occupied by received bytes

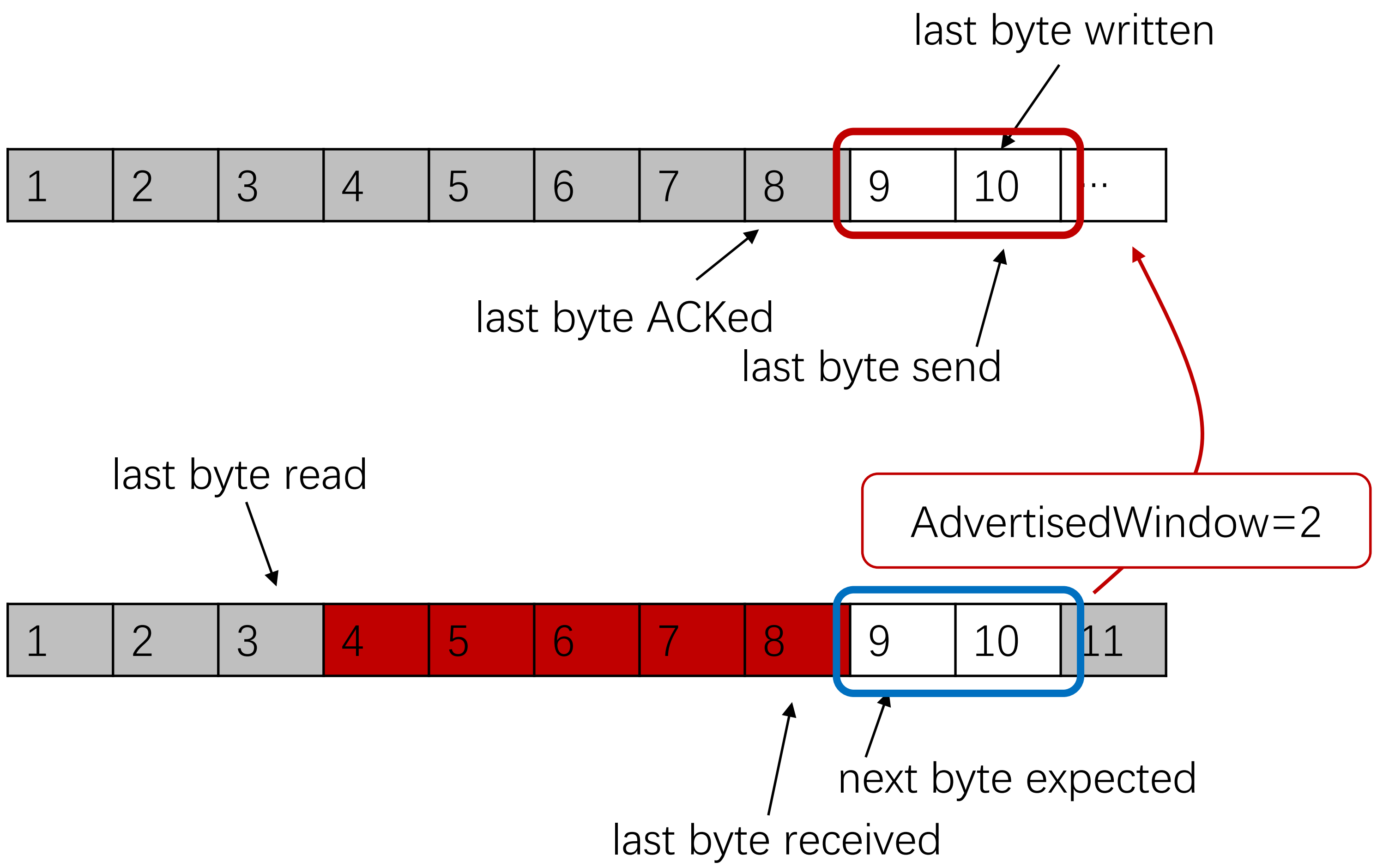
1 is read and freed by receiver



TCP Flow Control

Sender Buffer (8 bytes):
1,2 are freed buffer

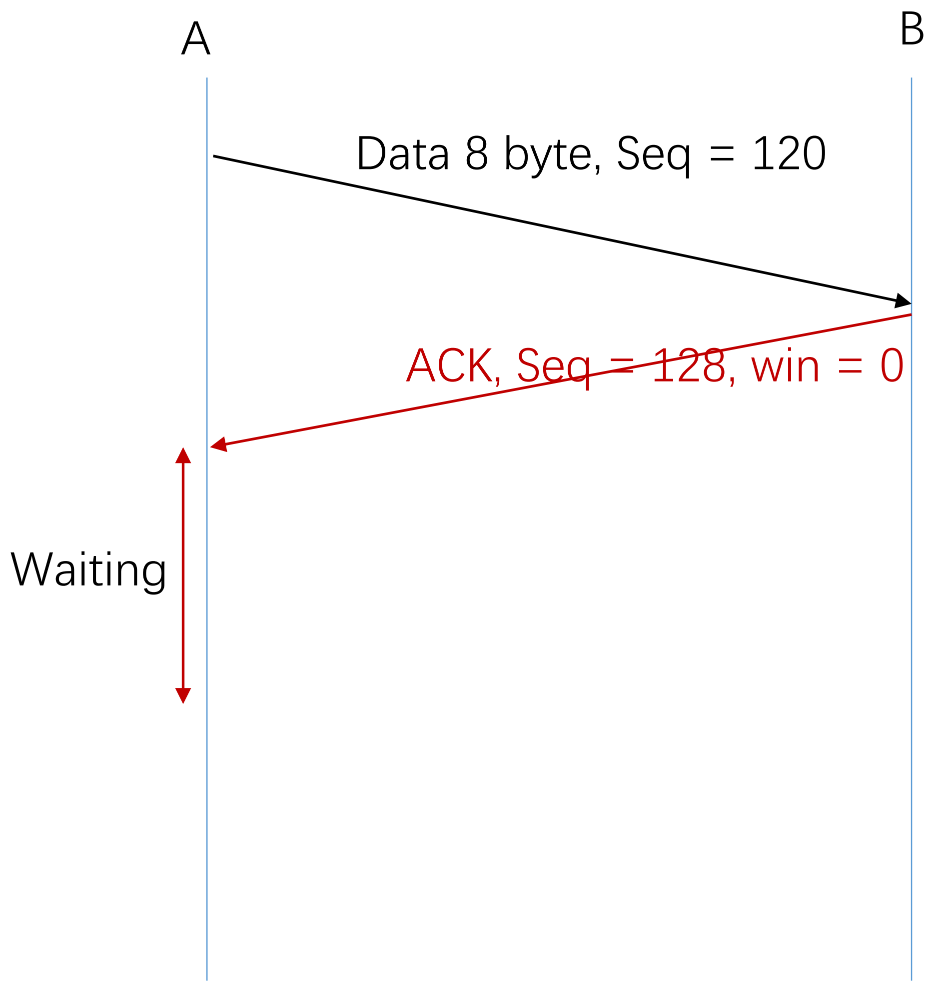
Receiver Buffer (7 bytes):
2,5 are occupied by received bytes
1 is read and freed by receiver



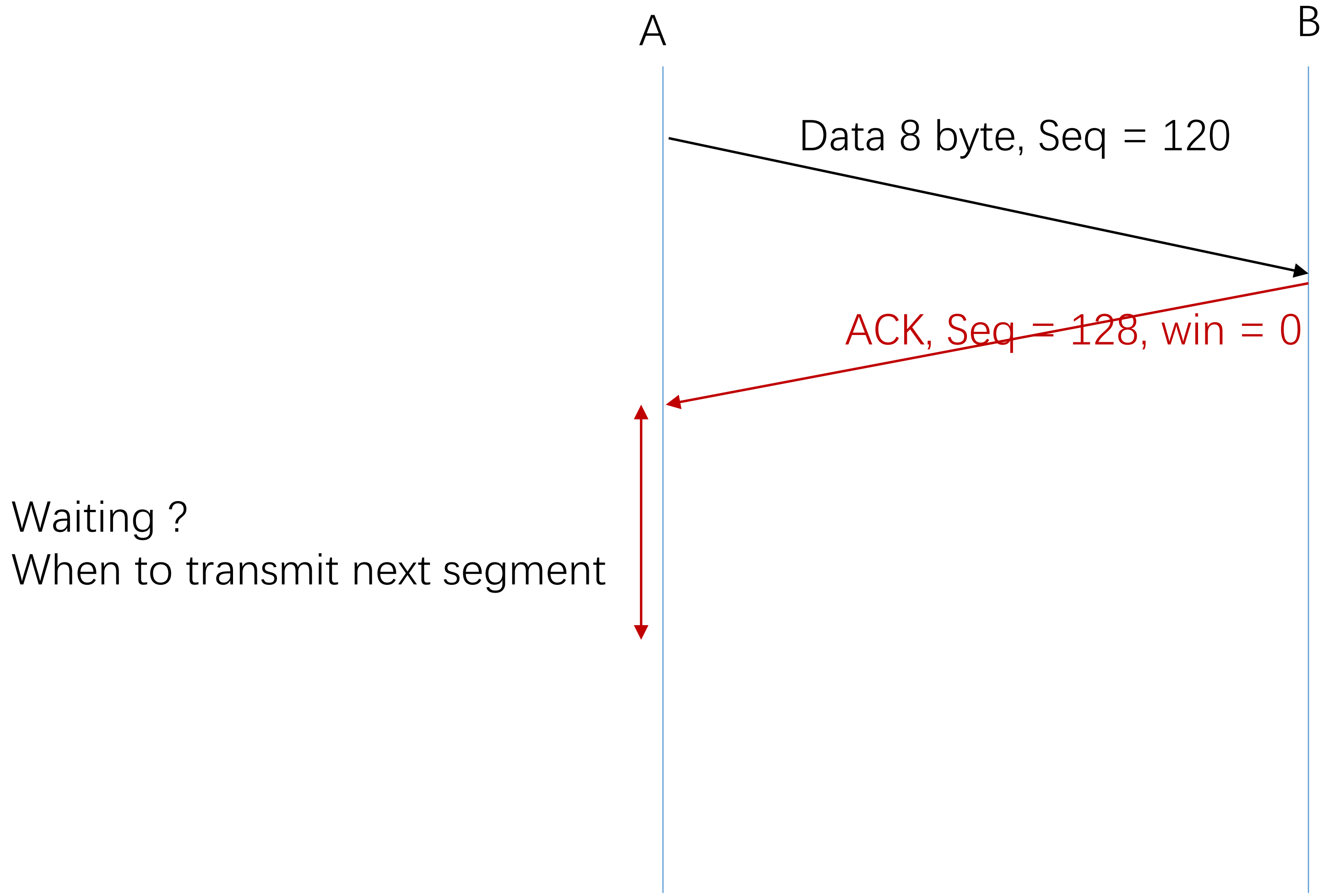
TCP Flow Control

- Receiver throttles sender by advertising a window size no larger than the amount it can buffer:
  - AdvertisedWindow=max receiver buffer-(last byte received - 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

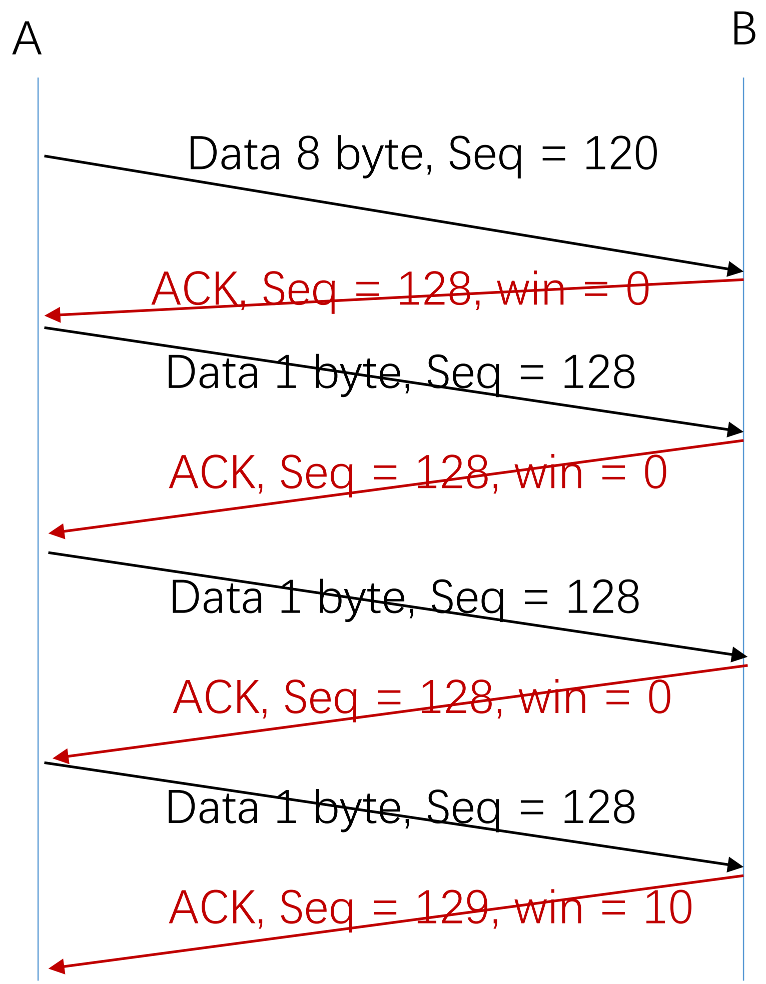
TCP Flow Control: 0 window case



TCP Flow Control: 0 window case

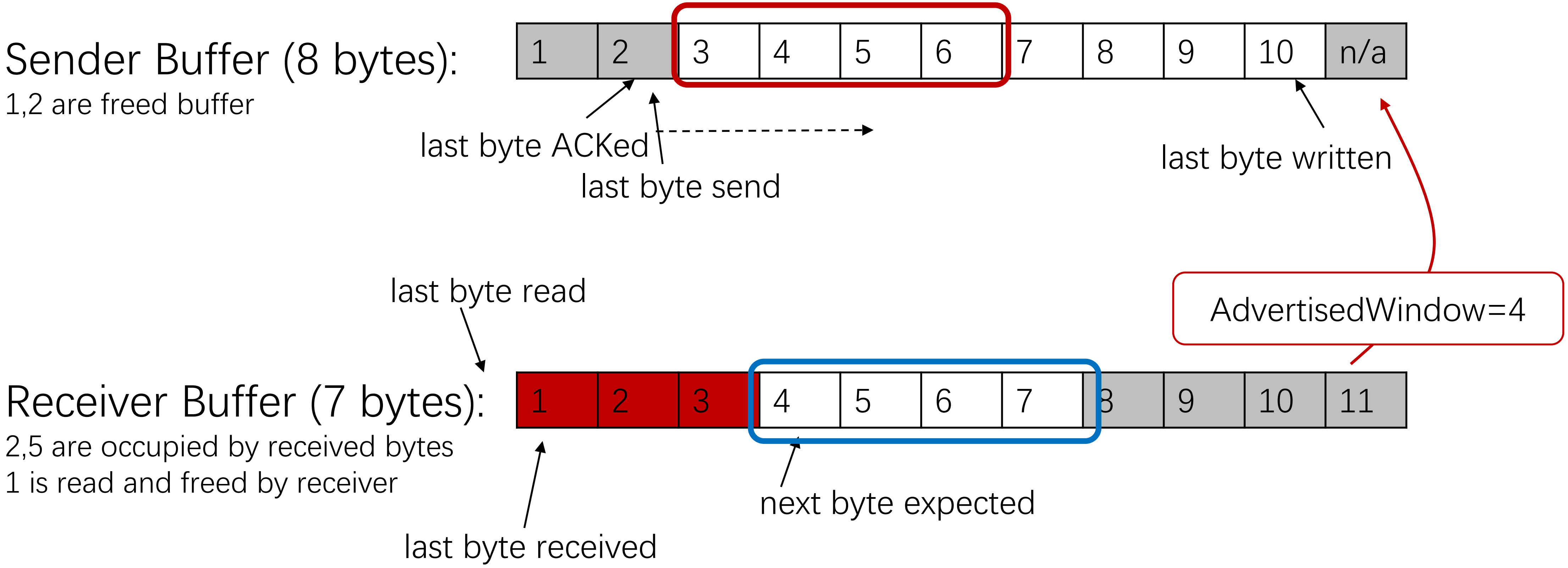


TCP Flow Control: 0 window case



TCP Flow Control: Silly Window Syndrome

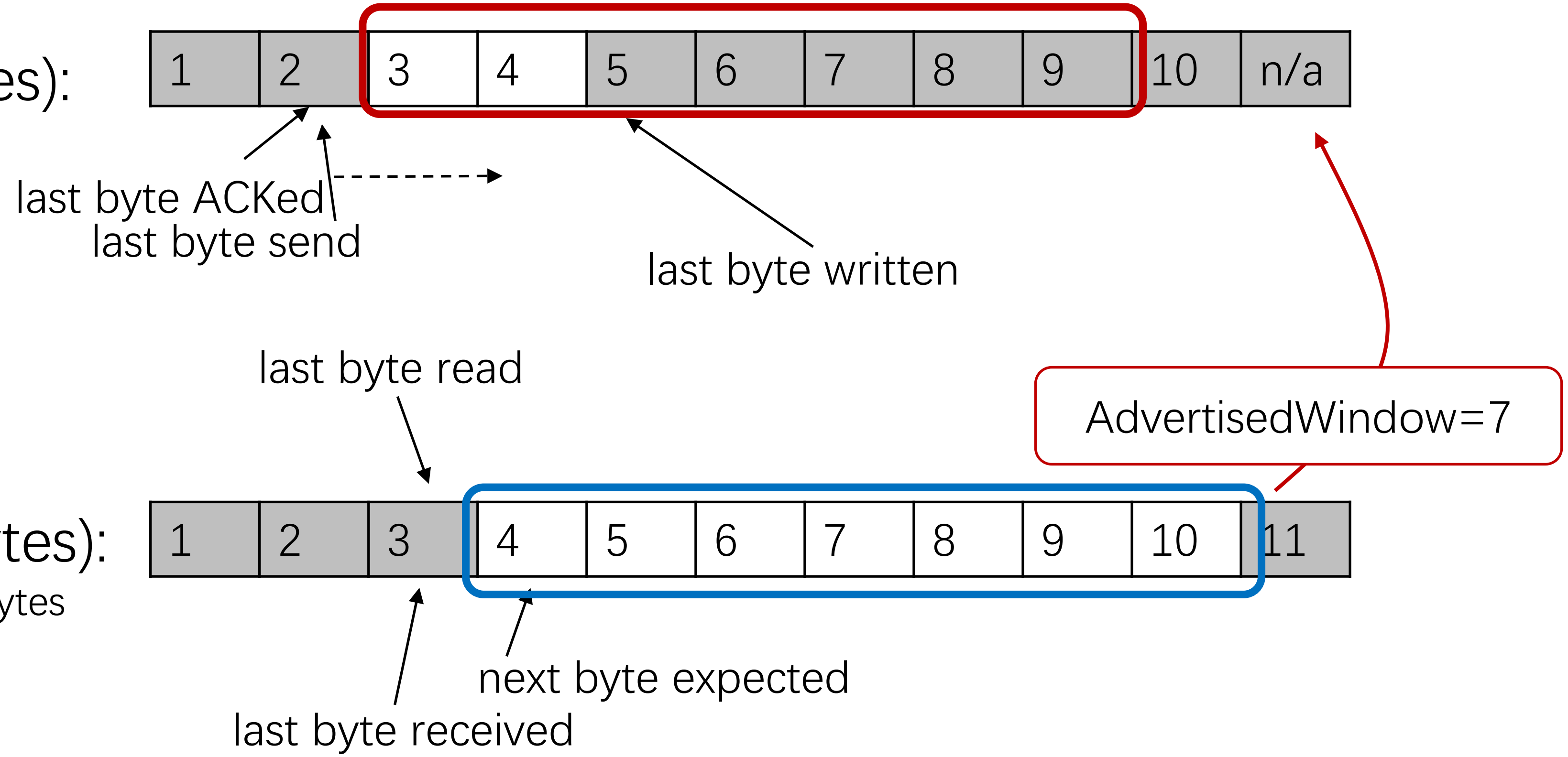
Case1: window Size < maximum segment size (6)



TCP Flow Control: Silly Window Syndrome

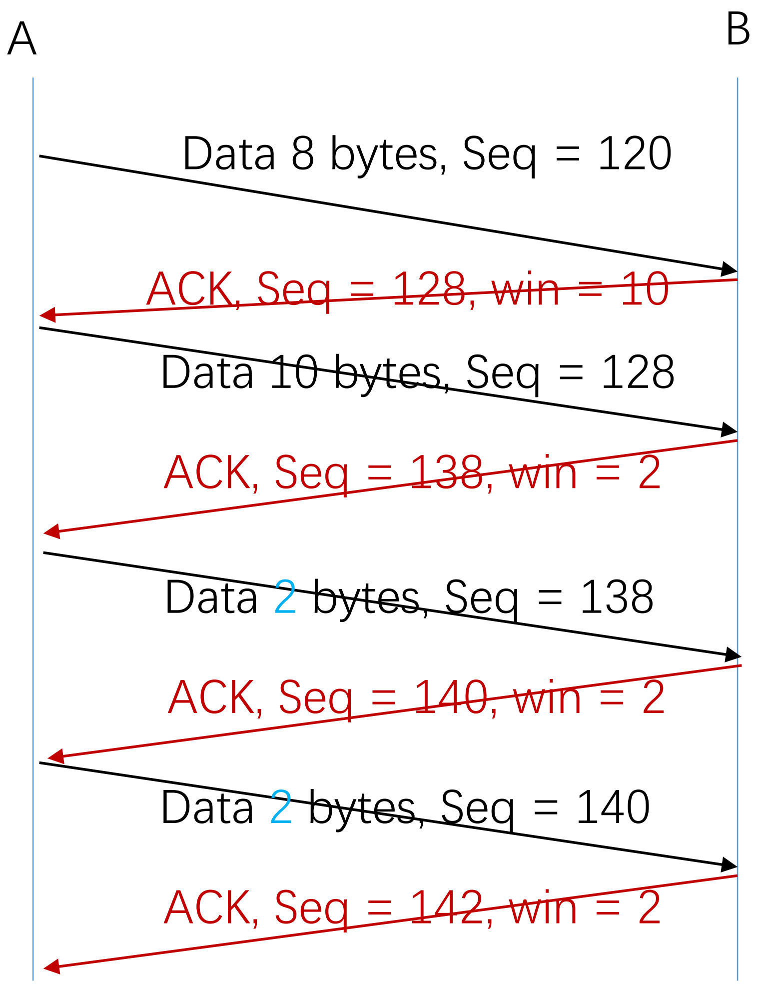
Case2: available data < maximum segment size (6)

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: Silly Window Syndrome



TCP: How to Trigger Transmission

- Best of Effort
  - Silly Window Syndrome
- Wait Until Full
  - Less Timely
- ➢Hybrid
  - 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

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

Some Patches

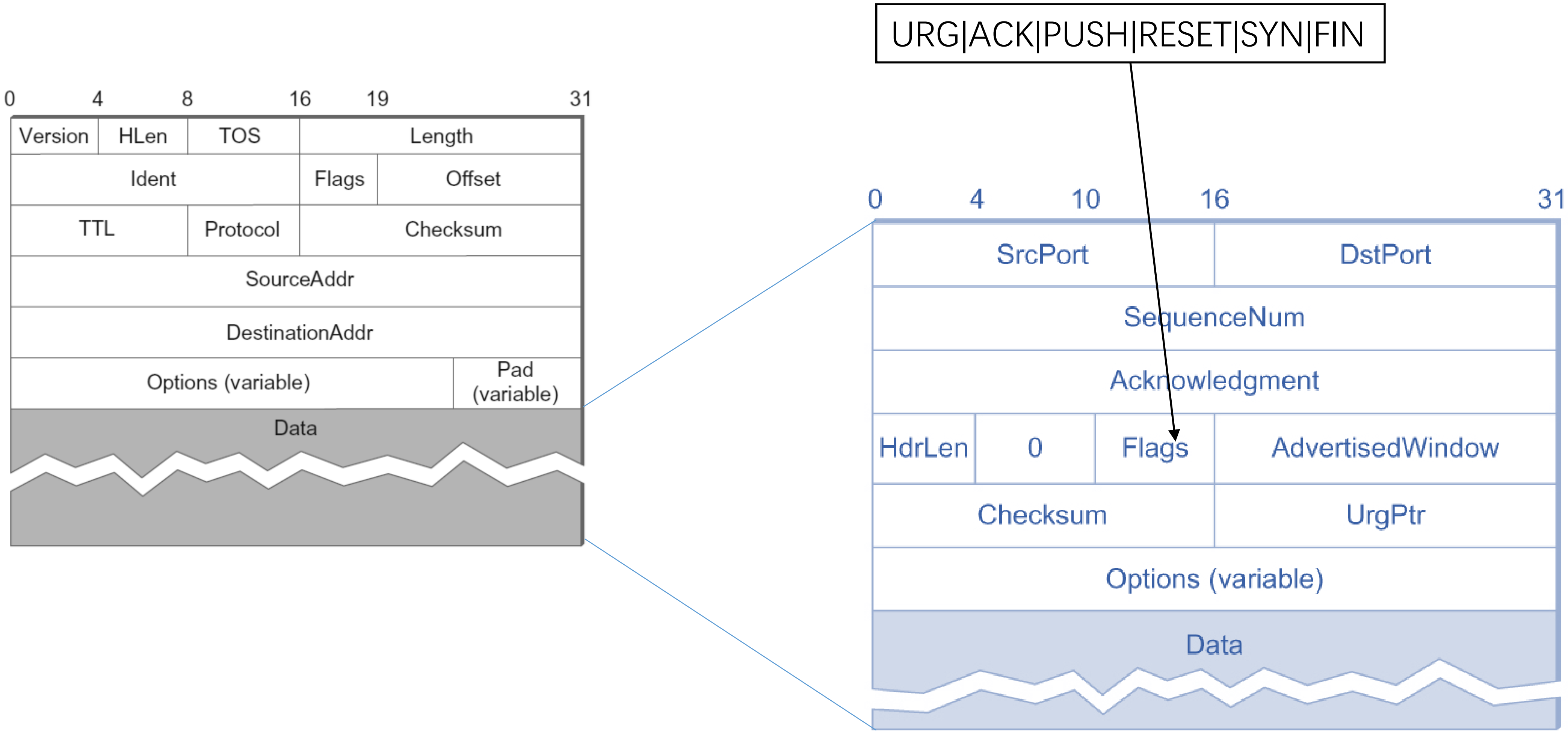
TCP Record Boundaries

- 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

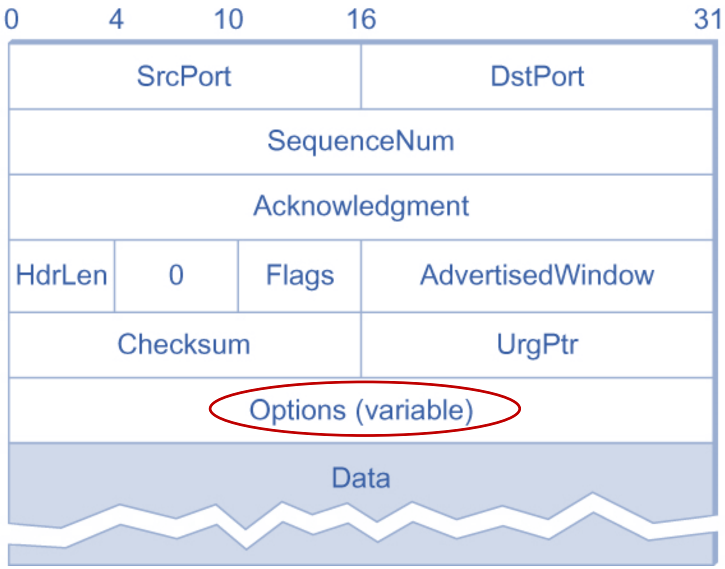
TCP: Header



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

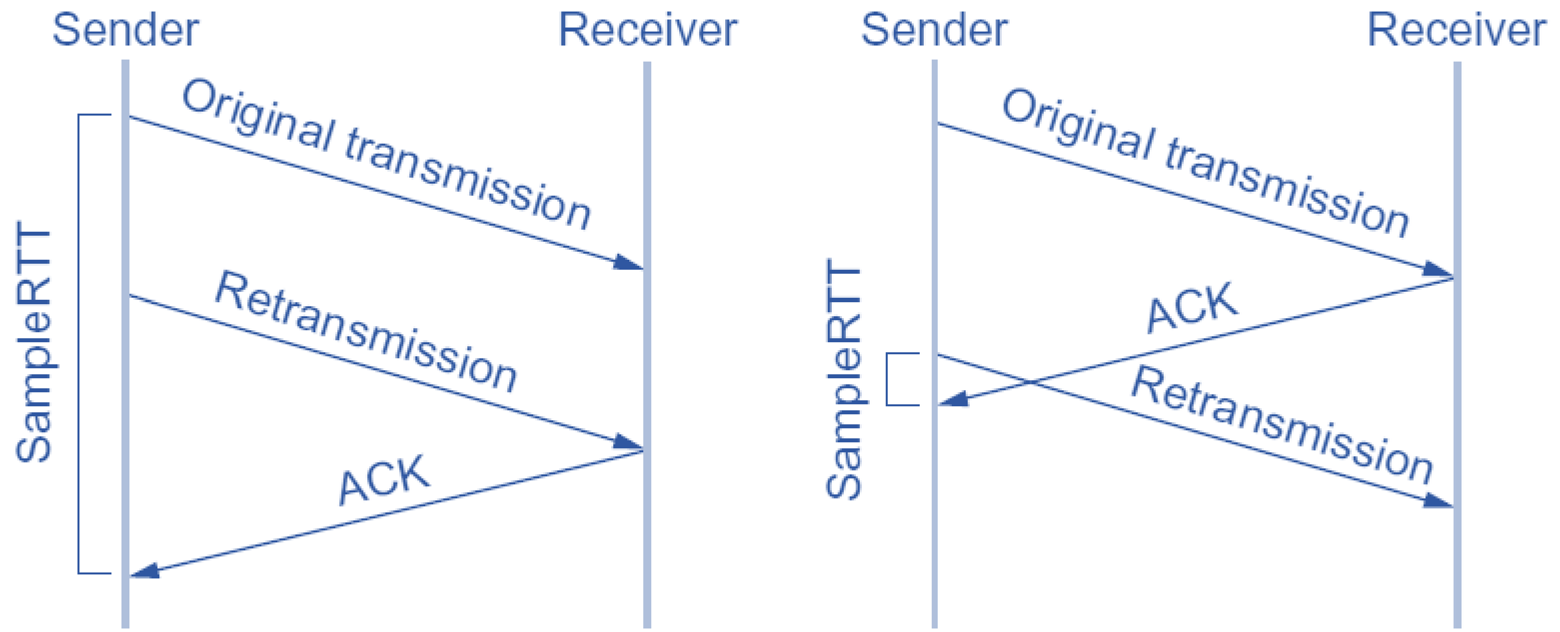
TCP Extensions

- TCP keeps evolving to incorporate high performance network



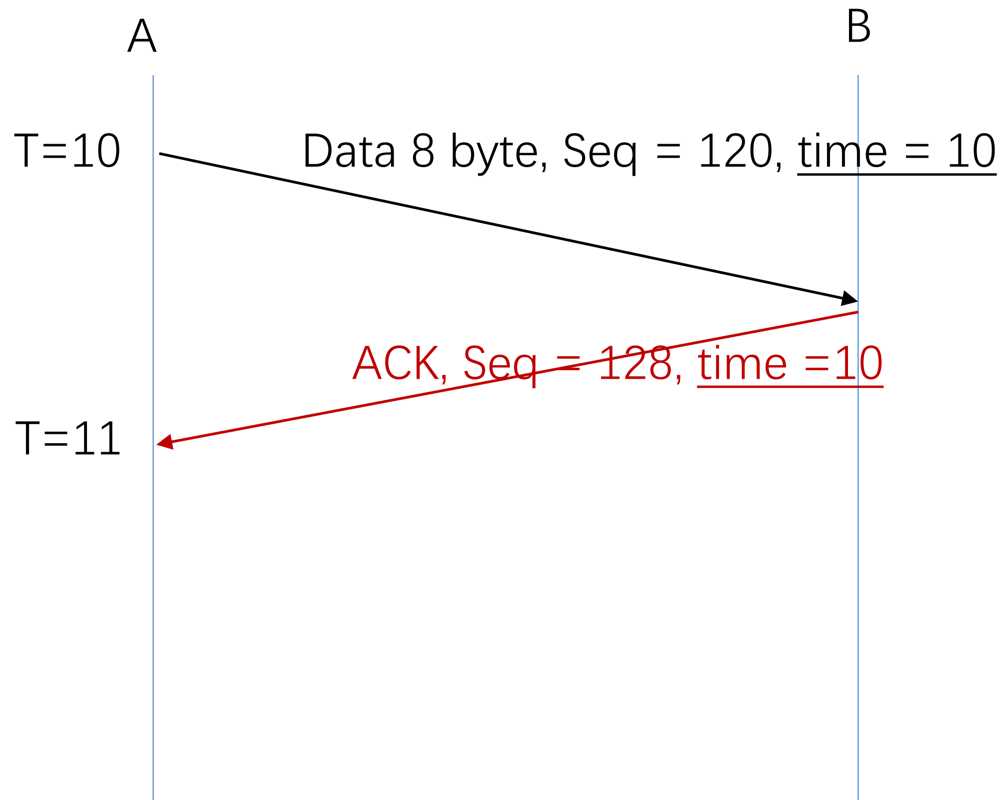
Sliding Window in TCP: Adaptive Timeout

- Original Algorithm
  - Mea
  - Cor
    -
    -
  - Set
    -



TCP Extensions

- Better Estimation of RTT
  - Introduce 32-bit timestamp



TCP Extensions

- 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

- Keeping the Pip Full: 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

- Selective ACK

Demo

- TCP Handshake
  - filter: tcp.stream eq X
- TCP Sliding Window
- TCP Extensions

TCP: Design Summary

- Stream-oriented v.s. Request/reply

- Byte-stream v.s. Message
  - Arbitrary long message v.s. Limited message
- In Order v.s. Out Order

- Explicit Setup v.s. Implicit Setup
- Window-based v.s. Rate-based

Reference

- Textbook 5.2
- TCP Extenstions https://www.ietf.org/rfc/rfc1323.txt