Lecture 13 : Transport Layer

Shankar Balachandran
Assistant Professor
Dept. of CSE, IIT Madras

Short Term Course on "Teaching Computer Networks Effectively". Sponsored by AICTE.

Network Layer Properties

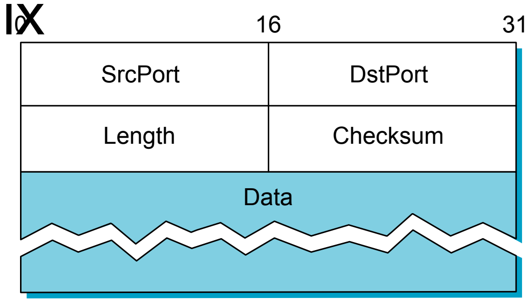
- Underlying best-effort network
  - drop messages
  - re-orders messages
  - delivers duplicate copies of a given message
  - limits messages to some finite size
  - delivers messages after an arbitrarily long delay

End to End Services

- Common end-to-end services
  - guarantee message delivery
  - deliver messages in the same order they are sent
  - deliver at most one copy of each message
  - support arbitrarily large messages
  - support synchronization
  - allow the receiver to flow control the sender
  - support multiple application processes on each host

Simple Demultiplexer (UDP)

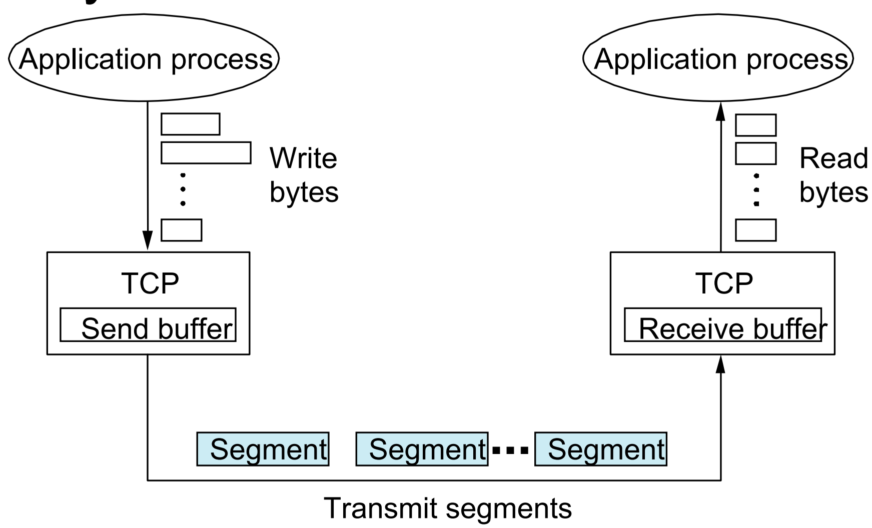
- Unreliable and unordered datagram service
- Adds multiplexing
- No flow control
- Endpoints identified by ports
  - servers have *well-known* ports
  - see **`/etc/services`** on Unix
- Header format



- Optional checksum
  - psuedo header + UDP header + data

TCP Overview

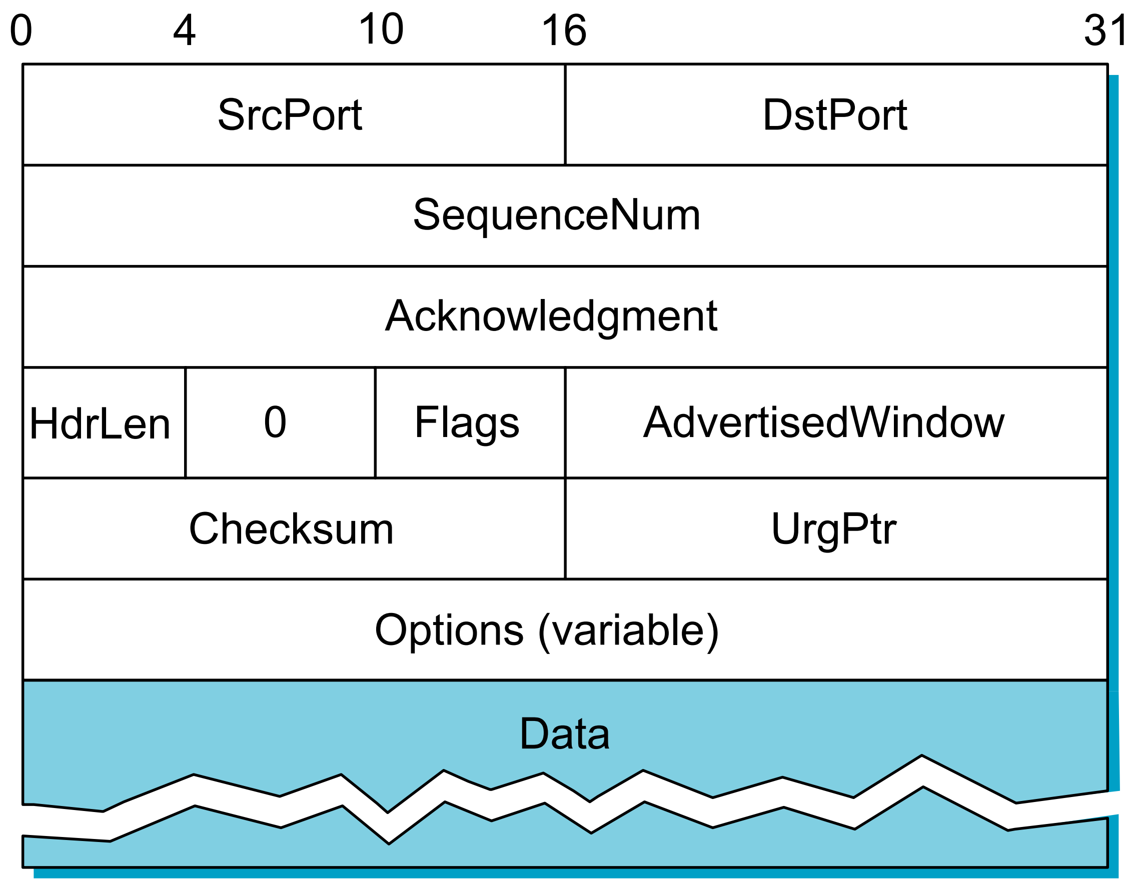
- Connection-oriented
- Byte-stream
  - app writes bytes
  - TCP sends *segments*
  - app reads bytes
- Full duplex
- Flow control: keep sender from overrunning receiver
- Congestion control: keep sender from overrunning network



Data Link Versus Transport

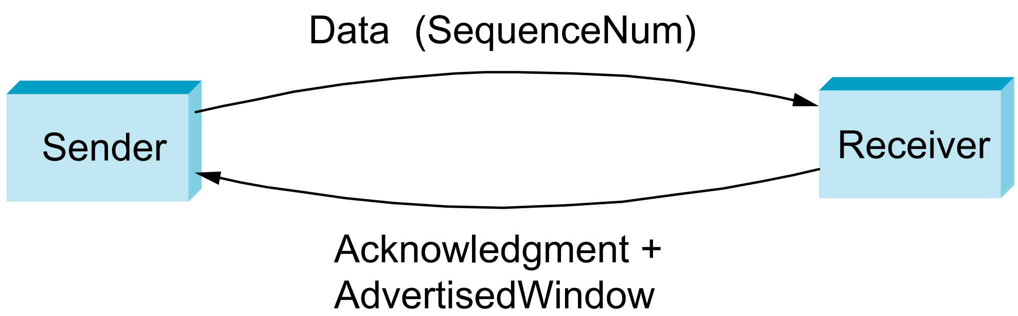
- Potentially connects many different hosts
  - need explicit connection establishment and termination
- Potentially different RTT
  - need adaptive timeout mechanism
- Potentially long delay in network
  - need to be prepared for arrival of very old packets
- Potentially different capacity at destination
  - need to accommodate different node capacity
- Potentially different network capacity
  - need to be prepared for network congestion

Segment Format



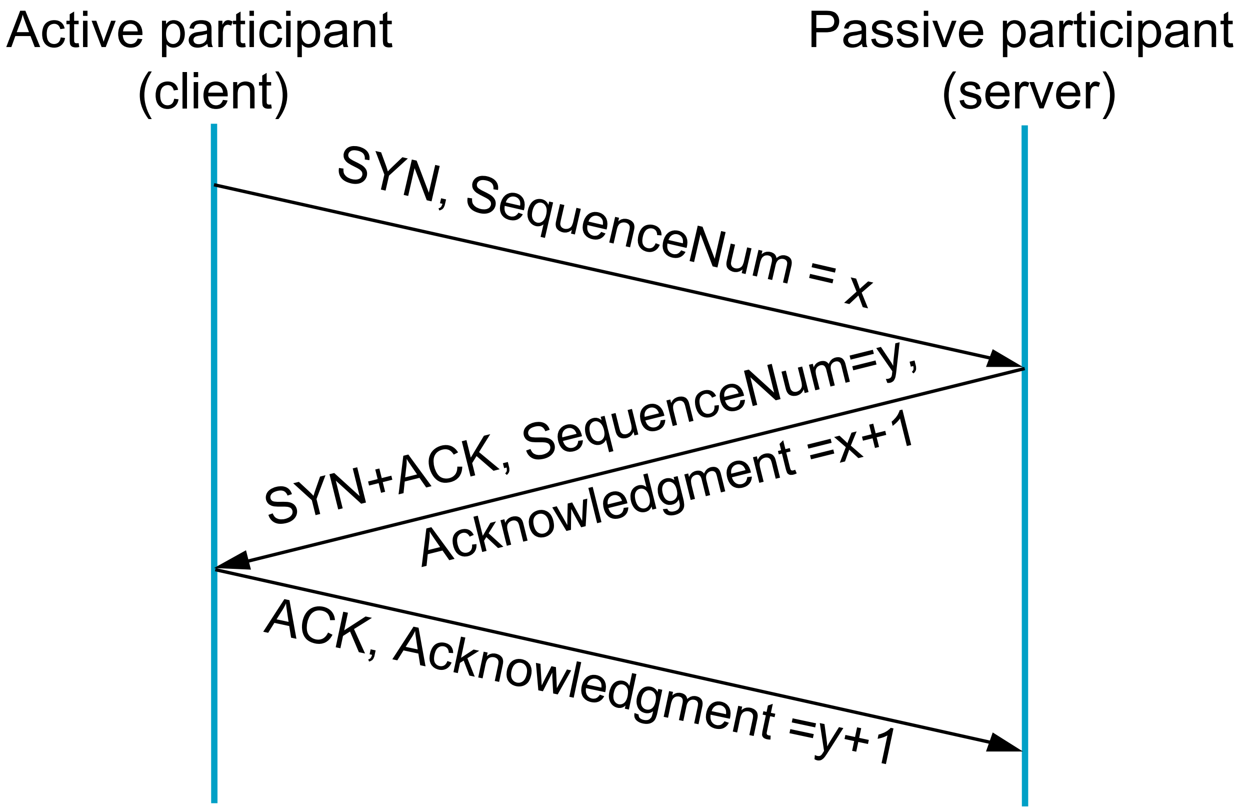
Segment Format (cont)

- Each connection identified with 4-tuple:
  - **`(SrcPort, SrcIPAddr, DstPort, DstIPAddr)`**
- Sliding window + flow control
  - **`acknowledgment, SequenceNum, AdvertisedWindow`**

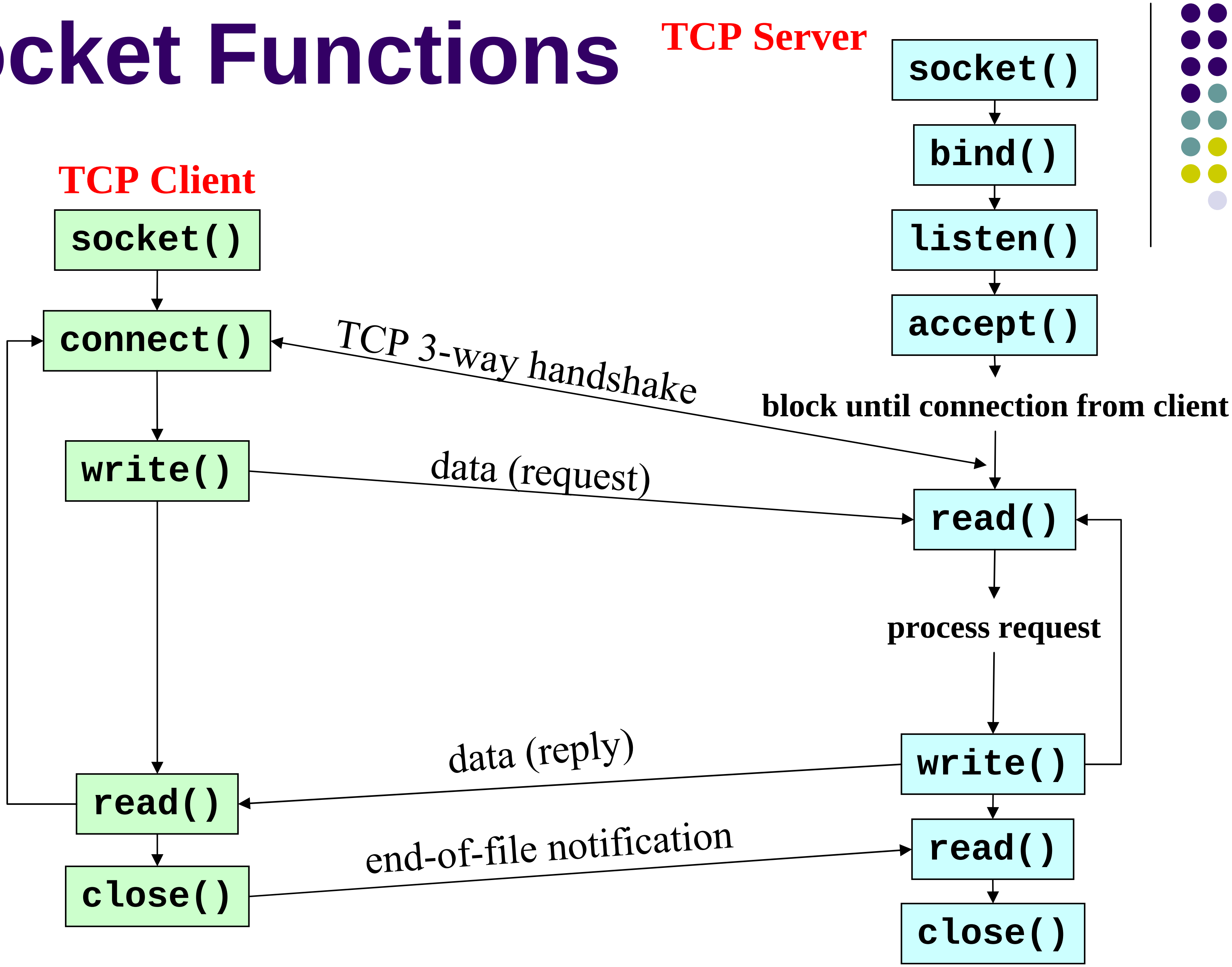


- Flags
  - **`SYN, FIN, RESET, PUSH, URG, ACK`**
- Checksum
  - pseudo header + TCP header + data

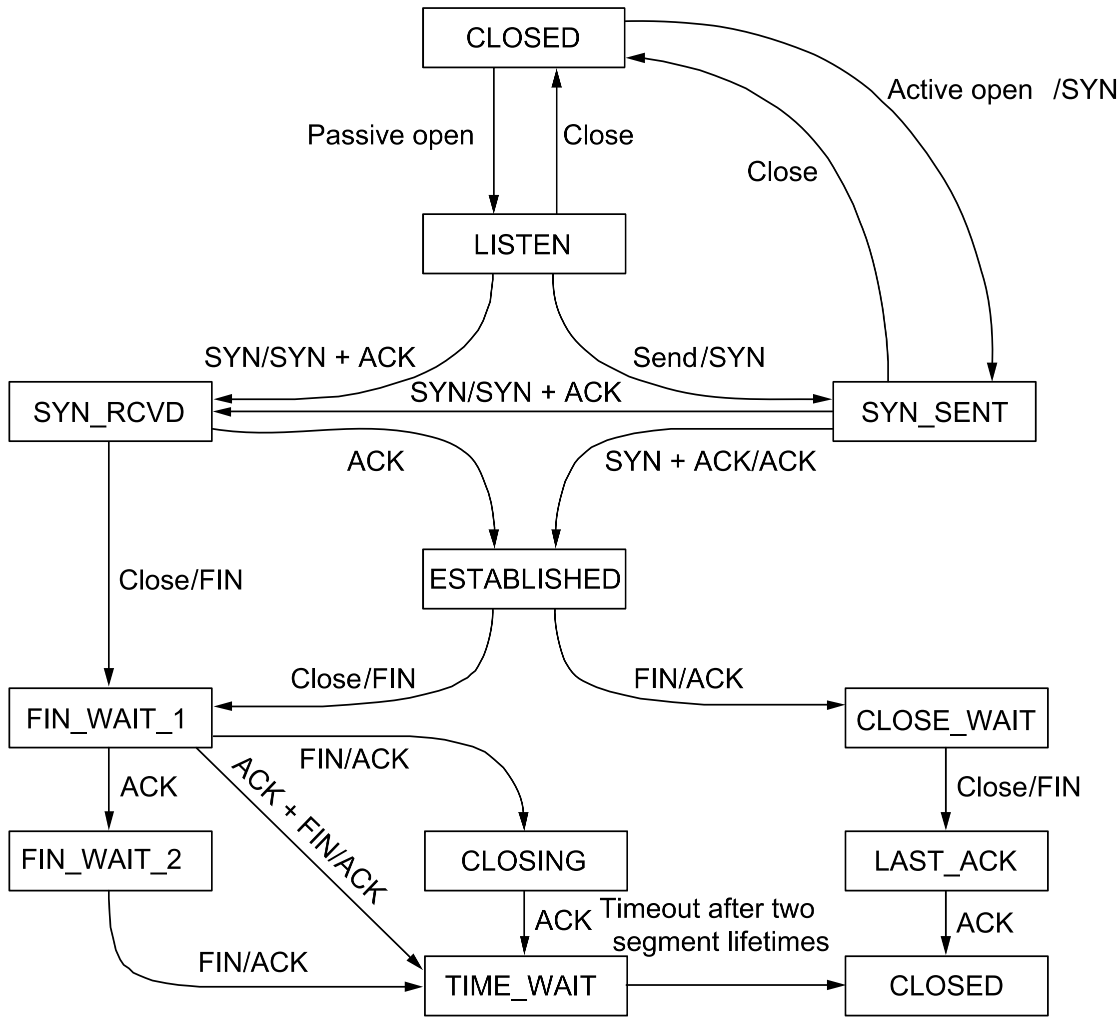
Connection Establishment



Socket Functions

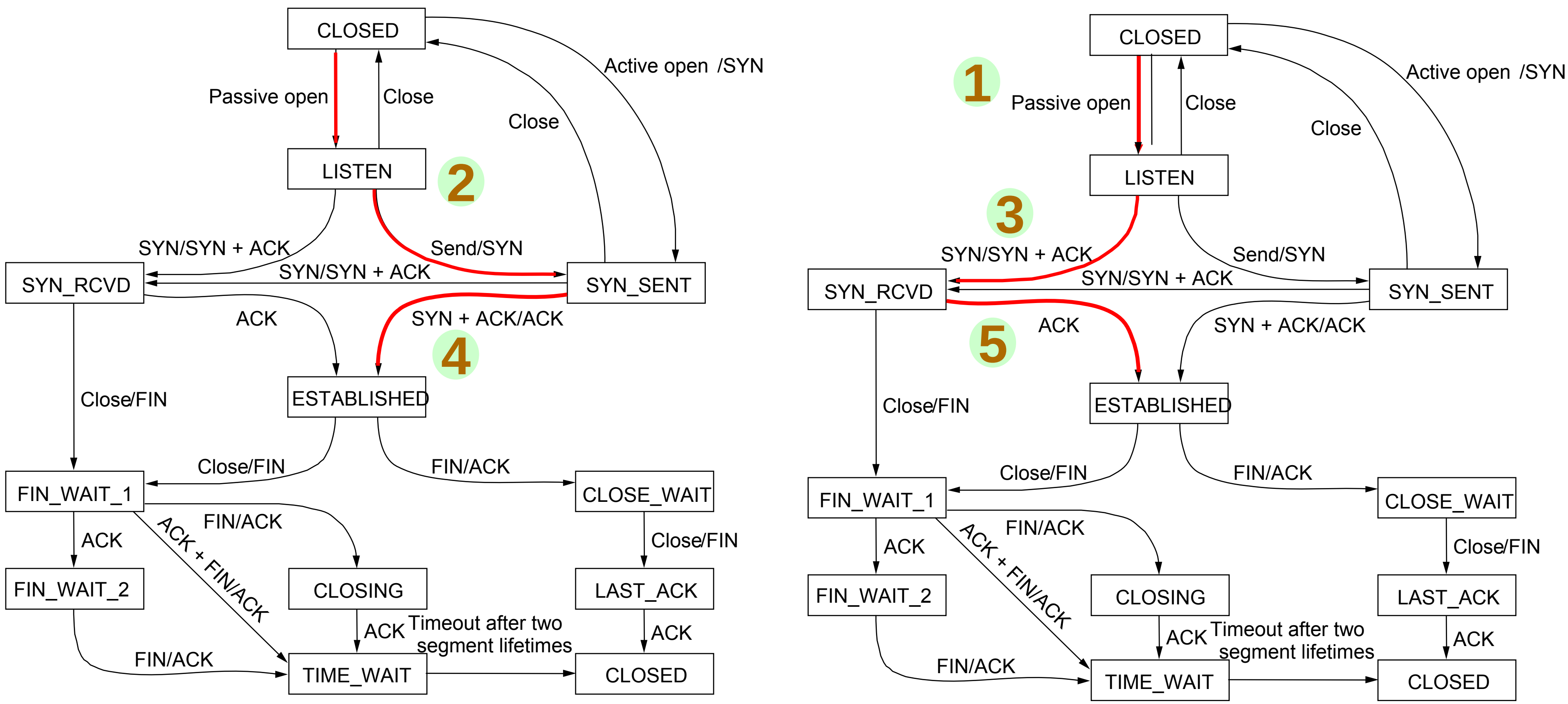


State Transition Diagram



Connection Establishment

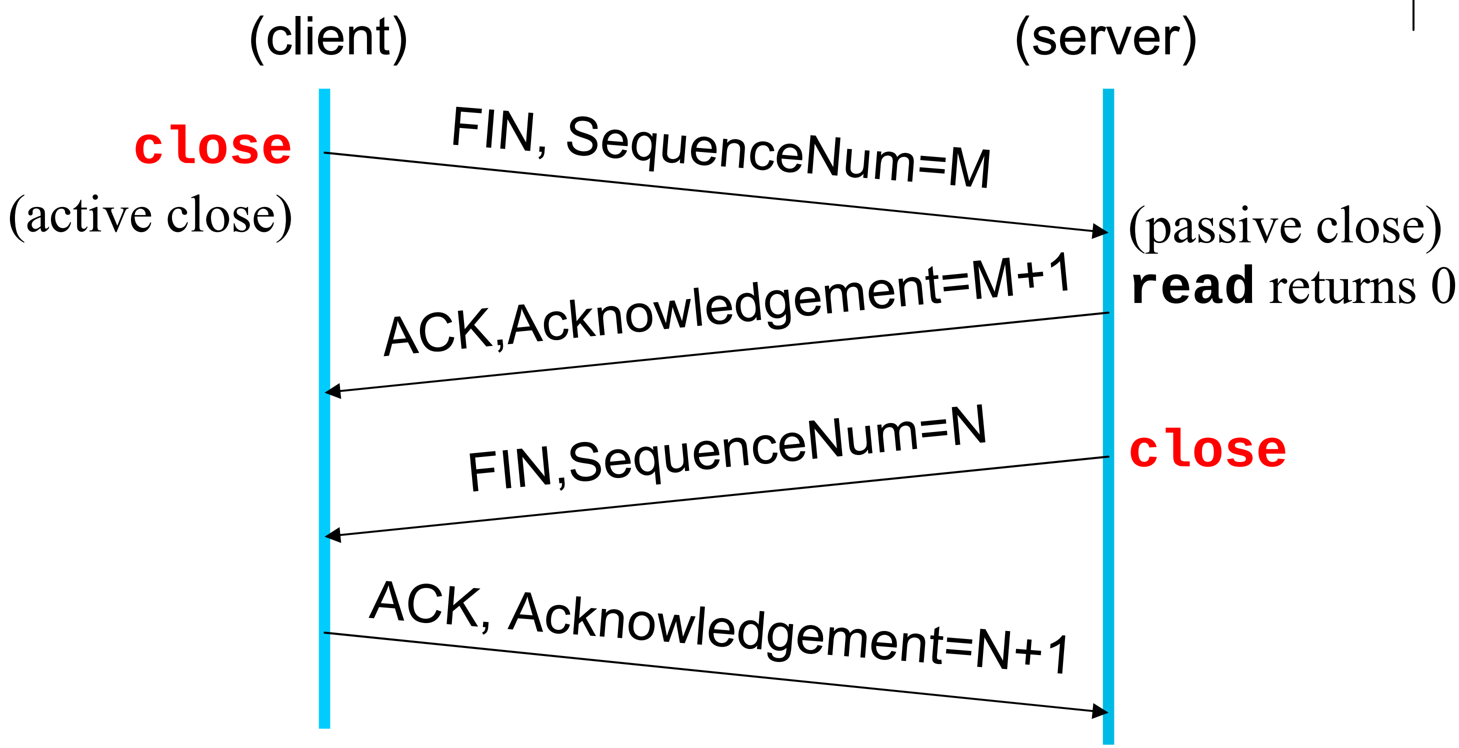
Client | Server



Steps 2,3,4 achieves the 3-Way Handshake

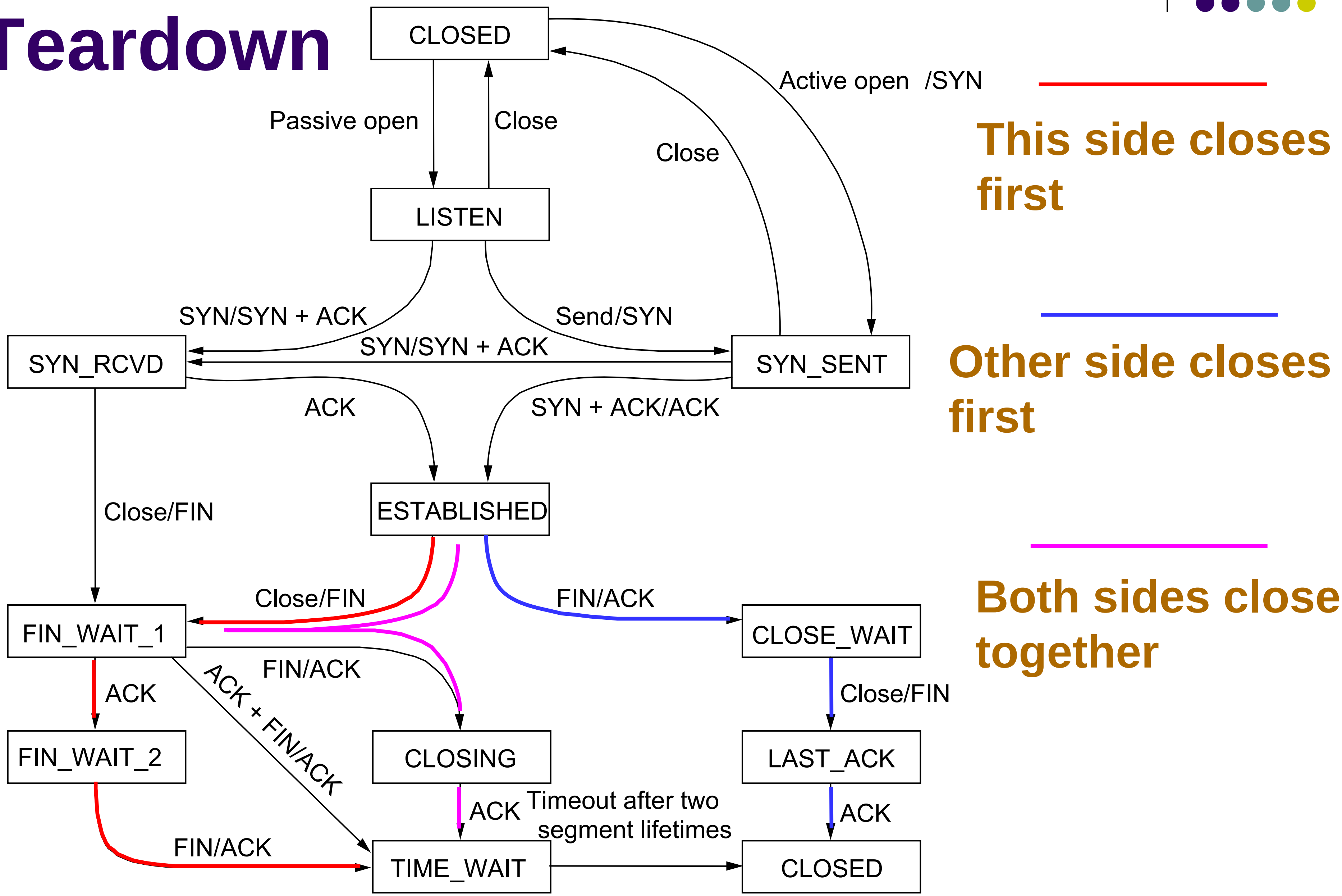
Question : What if ACK is lost? (@ step 4)

TCP Connection Termination



It takes 4 TCP segments to terminate a connection.

Connection Teardown

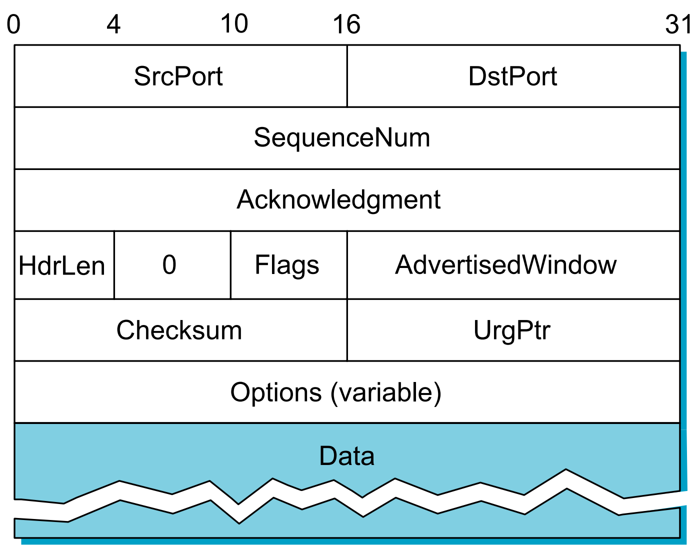


13.2 Sliding Window

TCP's sliding window algorithm:

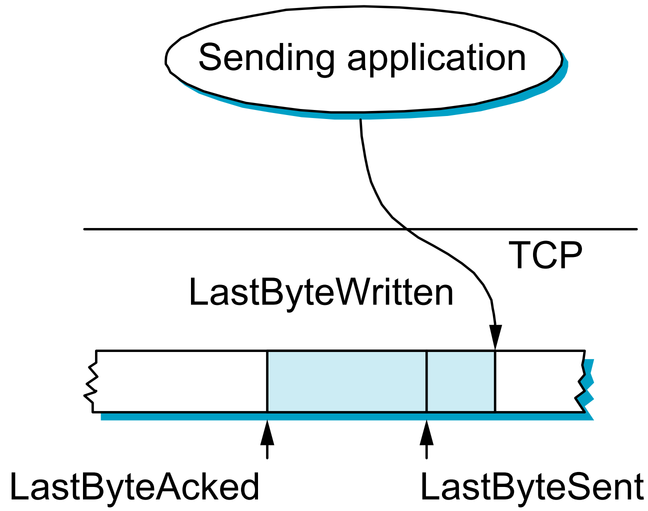
1) **Guarantees reliable delivery.**
2) **Guarantees in-order delivery.**
3) **Enforces flow control.**

Look again at the TCP segment header:



The receiver tells the sender the number of unacknowledged bytes of data it will allow (based on buffer size).

Flow Control

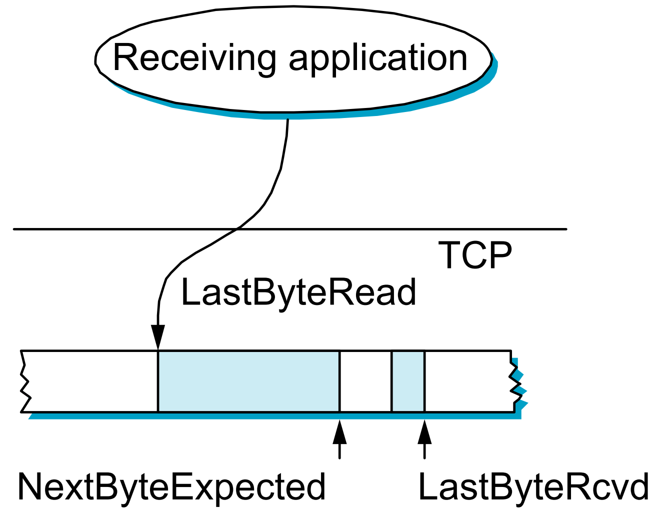


(a)

LastByteAcked $\leq$ LastByteSent

LastByteSent $\leq$ LastByteWritten

Buffer bytes between
LastByteAcked & LastByteWritten



(b)

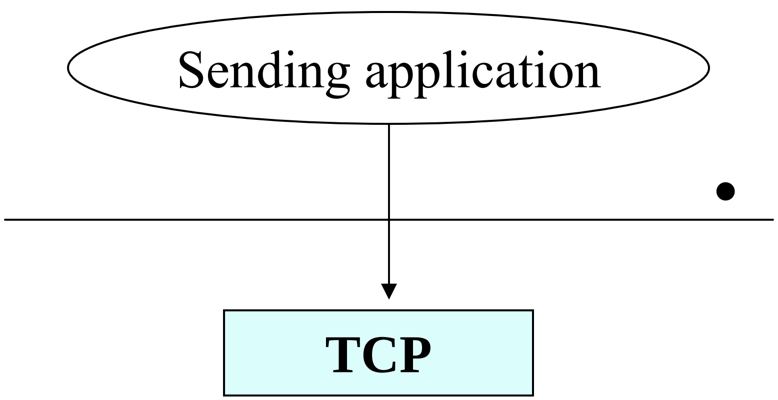
LastByteRead < NextByteExpected

NextByteExpected $\leq$ LastByteRcvd+1

Flow Control

- Send buffer size: **`MaxSendBuffer`**
- Receive buffer size: **`MaxRcvBuffer`**
- Receiving side
  - **`LastByteRcvd`** - **`LastByteRead`** < = **`MaxRcvBuffer`**
  - **`AdvertisedWindow`** = **`MaxRcvBuffer`** - (**`NextByteExpected`** - **`NextByteRead`**)
- Sending side
  - **`LastByteSent`** - **`LastByteAcked`** < = **`AdvertisedWindow`**
  - **`EffectiveWindow`** = **`AdvertisedWindow`** - (**`LastByteSent`** - **`LastByteAcked`**)
  - **`LastByteWritten`** - **`LastByteAcked`** < = **`MaxSendBuffer`**
  - block sender if (**`LastByteWritten`** - **`LastByteAcked`**) + *y* > **`MaxSenderBuffer`**
- Always send ACK in response to arriving data segment
- Persist when **`AdvertisedWindow`** **= 0**

Triggering Transmission



When does TCP decide to send a segment?

- As soon as MSS (maximum segment size) bytes have been collected from sender.
- Whenever the sending application explicitly requests it (***push***).
- Whenever a “timer” fires and then however many bytes have been buffered so far are sent.

Maximum Segment Size (MSS)

Rule of thumb: Usually set to the size of the largest segment TCP can send without causing the local IP to fragment.

IP header	**TCP header**	**MSS= sizeof(MTU) - sizeof(IP header) – sizeof(TCP header)**

MTU of the directly connected network

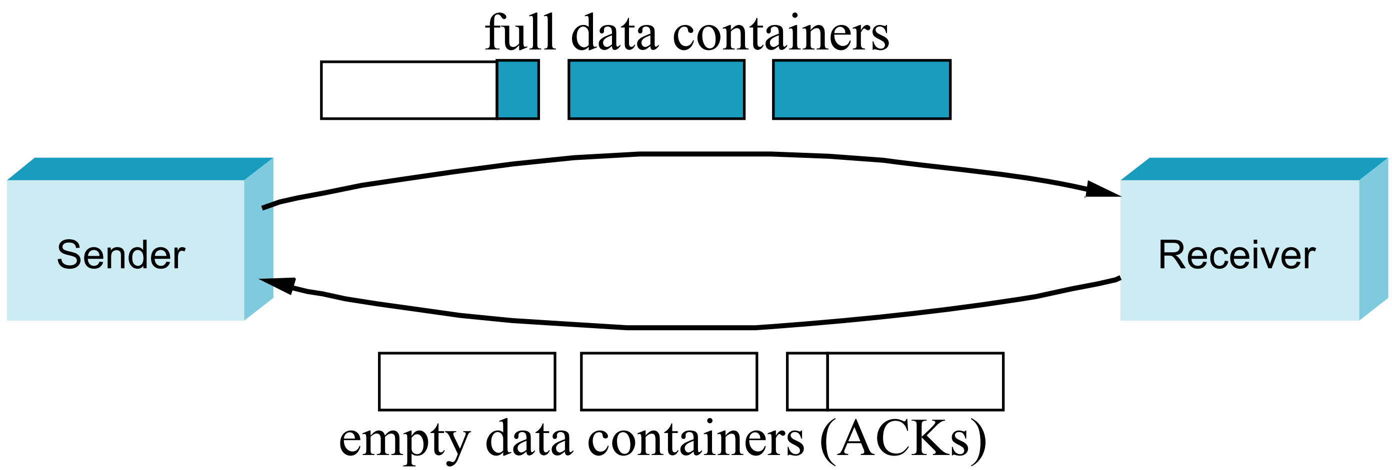
“Aggressive Send”

Now, consider what happens in the presence of flow control:

- AdvertisedWindow=0, so the sender is accumulating bytes to send.
- ACK arrives and AdvertisedWindow=MSS/2.
- **Question:** Should the sender go ahead and send MSS/2 or wait for AdvertisedWindow to increase all the way to MSS?

Silly Window Syndrome

- Consequence of aggressively taking advantage of any available window. Think of the TCP stream as a conveyor belt:



If the sender fills an empty container as soon as it arrives, then any small container introduced into the system remains indefinitely: it is immediately filled and emptied in each end.

Nagle's Algorithm

- How long does sender delay sending data?
  - too long: hurts interactive applications
  - too short: poor network utilization
  - strategies: timer-based vs self-clocking

Nagle's Algorithm

```
when (application has data to send) {
        if (available data>MSS) && (AdvertisedWindow>MSS)
           send full segment // OK to send full segment
        else
           if (unACKed data in transit)
             buffer the new data until an ACK arrives // wait for in-flight ACKs
           else
             send all the new data immediately // OK to send one small segment
    }
```

- Interactive applications like telnet
  - App writes one byte at a time
  - Segment = one byte or as many bytes that can be typed in one RTT
- For other apps :
  - TCP_NODELAY in socket should take care of it

Adaptive Retransmission (Original Algorithm)

- Measure `SampleRTT` for each segment / ACK pair
- Compute weighted average of RTT
  - **`EstRTT`** = α x **`EstRTT`** + β x **`SampleRTT`**
  - where $\alpha + \beta = 1$
    - α between 0.8 and 0.9
    - β between 0.1 and 0.2
- Set timeout based on `EstRTT`
  - **`TimeOut = 2`** x **`EstRTT`**

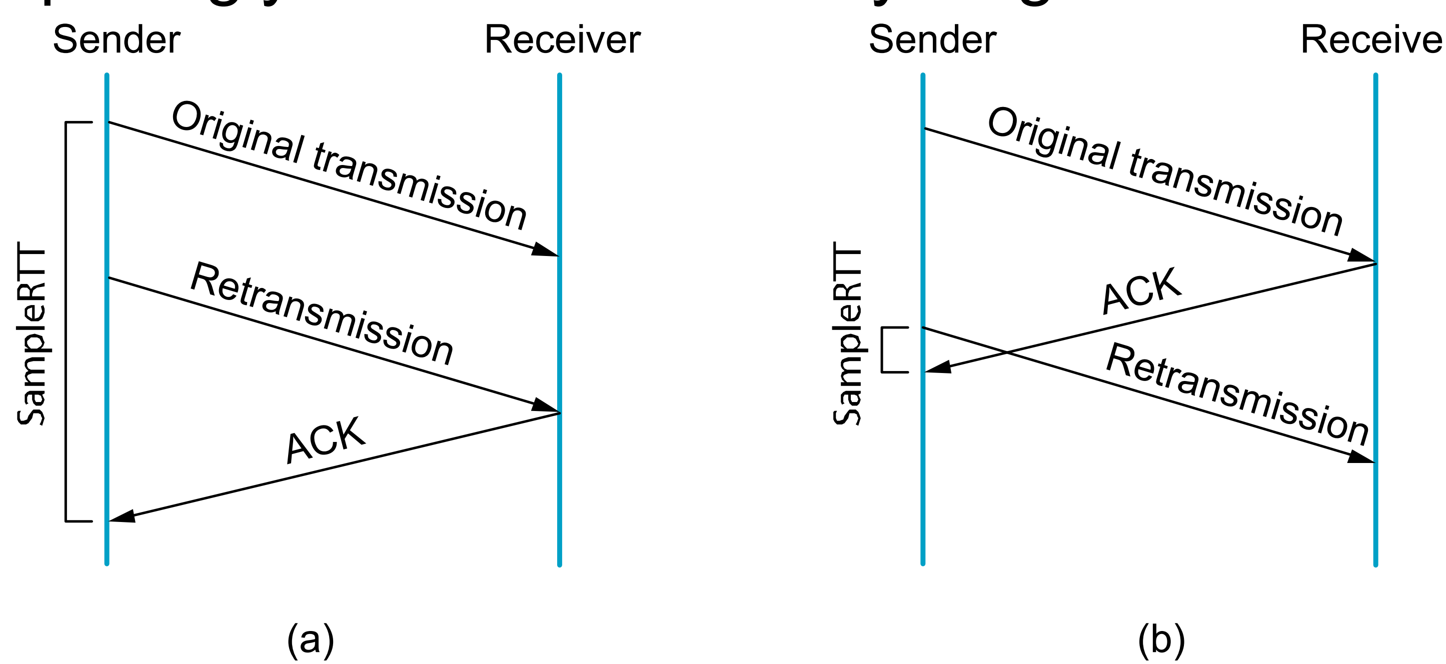
Question : What happens if α is too small?

Karn/Partridge Algorithm

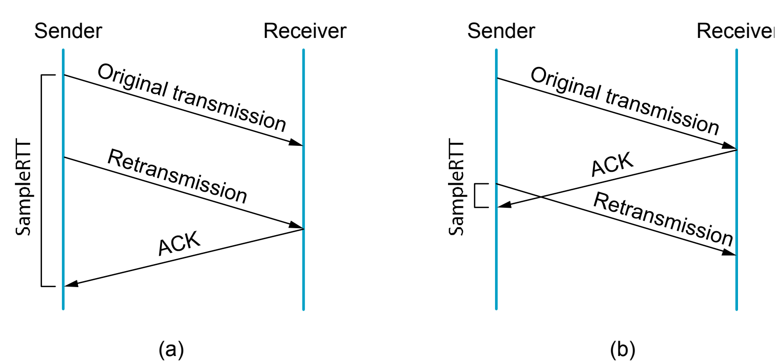
- ACK doesn't acknowledge a particular transmission.
- ACK acknowledges the ***receipt*** of data.

Question: If you don't know which transmission is being ACKed, how do you compute SampleRTT?

Surprisingly noticed after a very long time.



Karn/Partridge Algorithm



Solution:

- Compute RTT only for the first transmission of a packet.
- Each time TCP retransmits a packet, set the timeout value to 2xTimeOut.

Rationale: If packets are being lost, this is likely to be the consequence of congestion, so the sender should be less aggressive.

Jacobson/ Karels Algorithm

- New Calculations for average RTT
- **`Diff`** = **`SampleRTT`** - **`EstRTT`**
- **`EstRTT = EstRTT + (`**δ x **`Diff)`**
- **`Dev = Dev + `**δ**`( |Diff| - Dev)`**
  - where δ is a factor between 0 and 1
- Consider variance when setting timeout value
- **`TimeOut`** = μ x **`EstRTT`** + ϕ x **`Dev`**
  - where μ = 1 and ϕ = 4
- Notes
  - algorithm only as good as granularity of clock (500ms on Unix)
  - accurate timeout mechanism important to congestion control (later)

Protection Against Wrap Around

- 32-bit **SequenceNum**

Bandwidth	Time Until Wrap Around
T1 (1.5 Mbps)	6.4 hours
Ethernet (10 Mbps)	57 minutes
T3 (45 Mbps)	13 minutes
FDDI (100 Mbps)	6 minutes
STS-3 (155 Mbps)	4 minutes
STS-12 (622 Mbps)	55 seconds
STS-24 (1.2 Gbps)	28 seconds

Keeping the Pipe Full

- 16-bit **`AdvertisedWindow`**

Bandwidth	Delay x Bandwidth Product
T1 (1.5 Mbps)	18KB
Ethernet (10 Mbps)	122KB
T3 (45 Mbps)	549KB
FDDI (100 Mbps)	1.2MB
STS-3 (155 Mbps)	1.8MB
STS-12 (622 Mbps)	7.4MB
STS-24 (1.2 Gbps)	14.8MB

assuming 100ms RTT

Lot worse than Seqnum; Cannot handle T3

TCP Extensions

- Implemented as header options
- Store timestamp in outgoing segments
  - Receiver sends the timestamp back in acknowledgment
  - Global synchronization is not necessary
    - Why?
- Extend sequence space with 32-bit timestamp
  - 64 bit value : Time stamp + Sequence Number
  - Seqnum will not wrap around now
    - Time always shifts
- Shift (scale) advertised window

Summary

- End to end argument
- UDP
- TCP
- Flow control in TCP