Transport Control Protocol

Outline

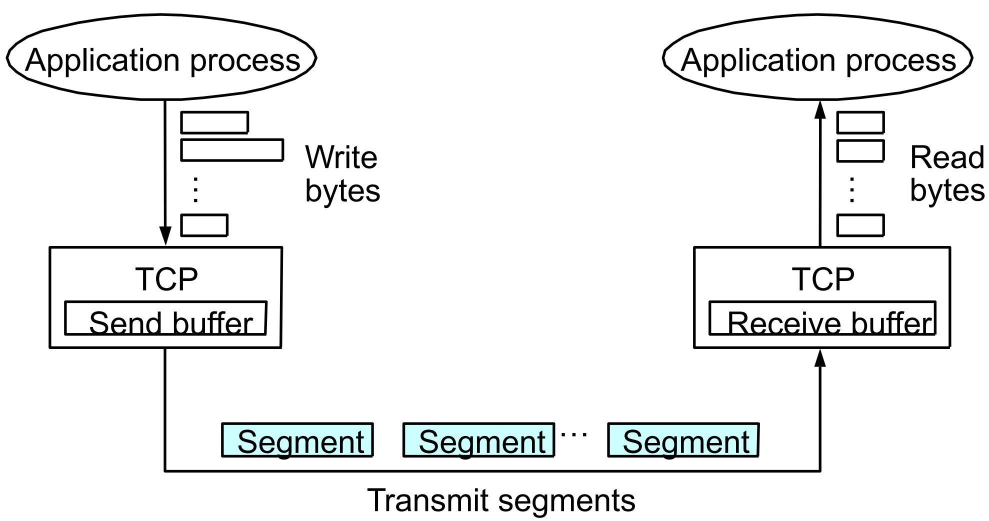
- TCP objectives revisited
- TCP basics
- New algorithms for RTO calculation

TCP Overview

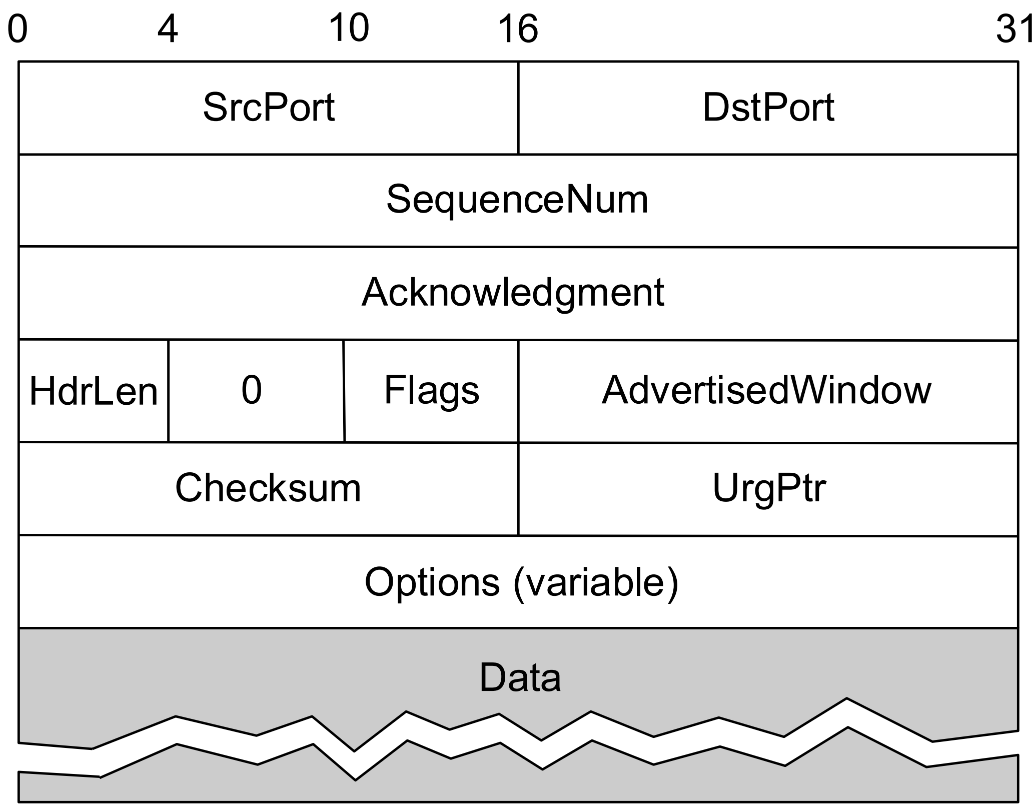
- TCP is the most widely used Internet protocol
  - Web, Peer-to-peer, FTP, telnet, …
- A two way, reliable, byte stream oriented end-to-end protocol
  - Includes flow and congestion control
- Closely tied to the Internet Protocol (IP)
- A focus of intense study for many years
  - Our goal is to understand the RENO version of TCP
    - RENO is most widely used TCP today
    - RFC 2001 (now expired)
    - RENO mainly specifies mechanisms for dealing with congestion

TCP Features

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

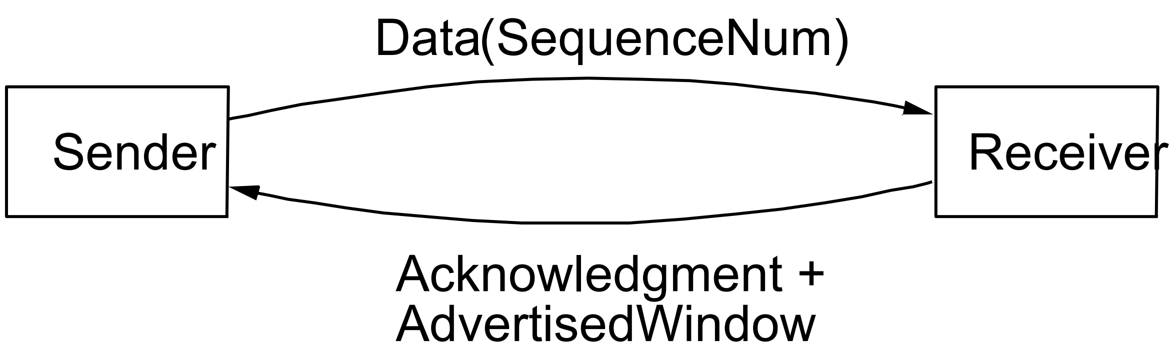


Segment Format



Segment Format (cont)

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



- Flags
  - **`SYN, FIN, RESET, PUSH, URG, ACK`**
- Checksum is the same as UDP
  - pseudo header + TCP header + data

Sequence Numbers

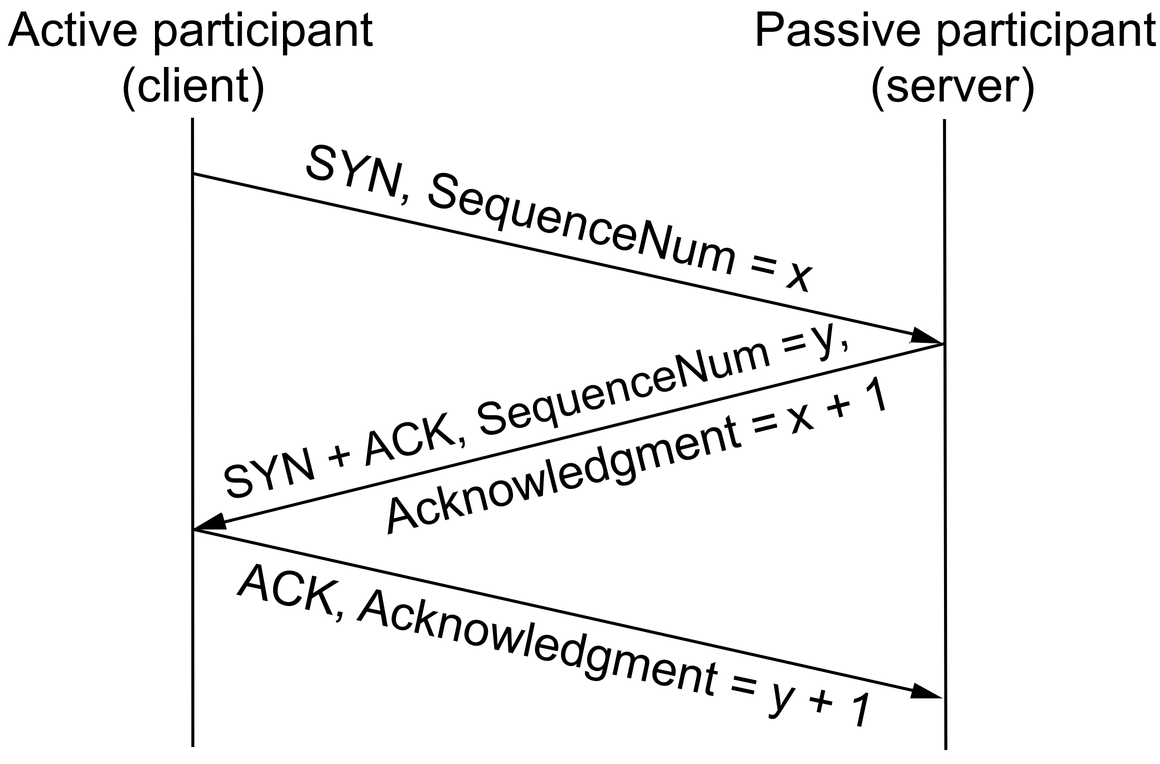
- 32 bit sequence numbers
  - Wrap around supported
- TCP breaks byte stream from application into packets (limited by Max. Segment Size)
- Each byte in the data stream is considered
- Each packet has a sequence number
  - Initial number selected at connection time
  - Subsequent numbers indicate first data byte number in packet
- ACK's indicate ***next byte expected***

Sequence Number Wrap Around

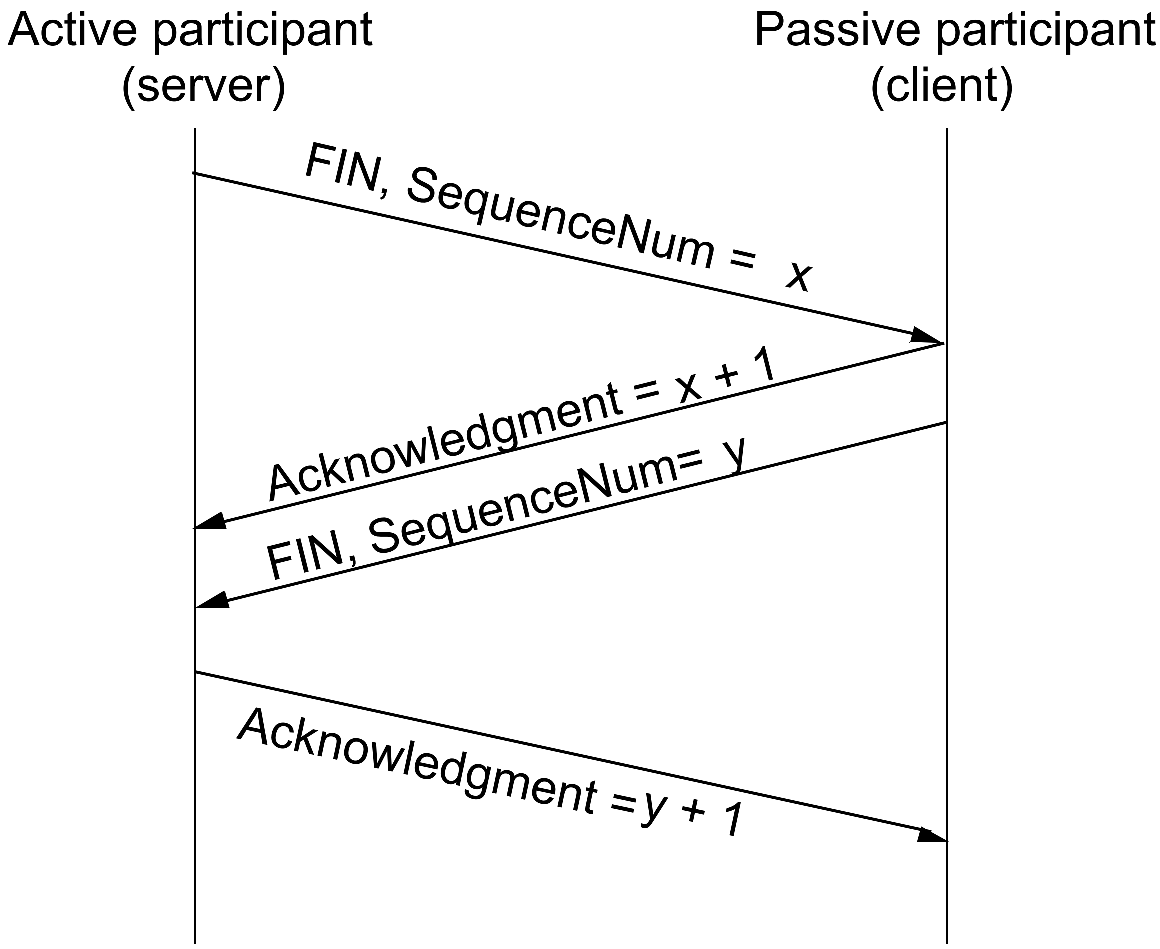
Bandwidth	Time Until Wrap Around
T1 (1.5 Mbps)	6.4 hours
Ethernet (10 Mbps)	57 minutes
T3 (45 Mbps)	13 minutes
FDDI (100 Mbps)	6 minutes
STS-3 (155 Mbps)	4 minutes
STS-12 (622 Mbps)	55 seconds
STS-24 (1.2 Gbps)	28 seconds

- Protect against this by adding a 32-bit timestamp to TCP header

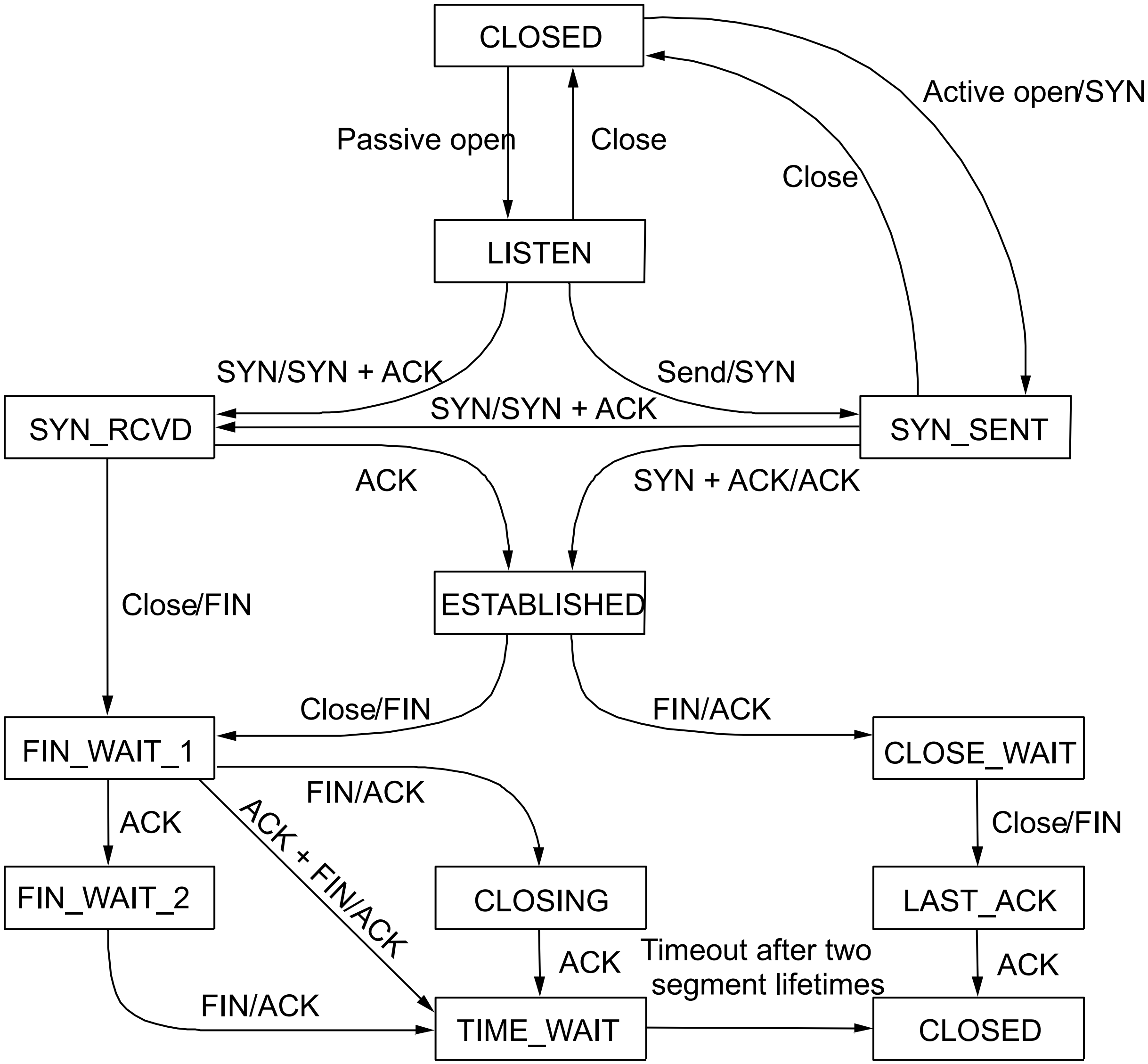
Connection Establishment



Connection Termination



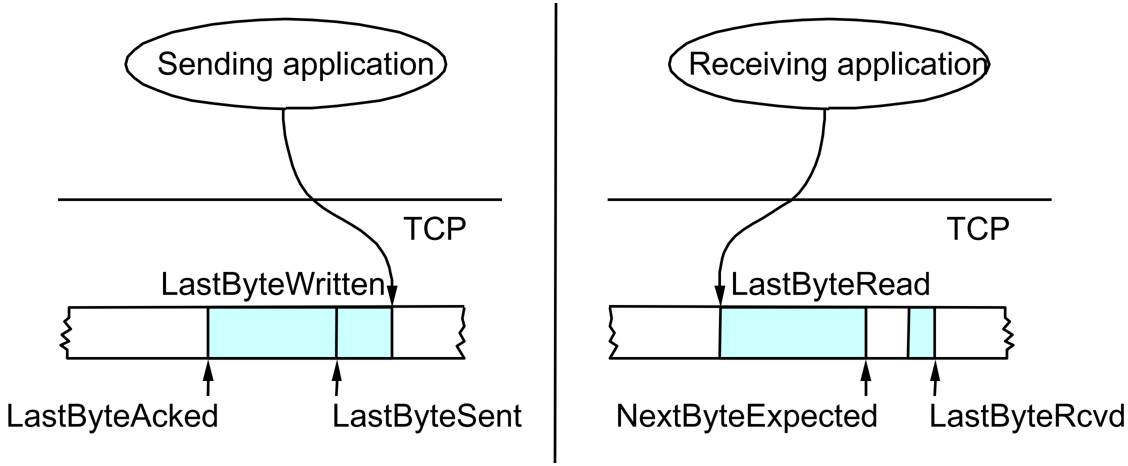
State Transition Diagram



Reliability in TCP

- Checksum used to detect bit level errors
- Sequence numbers used to detect sequencing errors
  - Duplicates are ignored
  - Reordered packets are reordered (or dropped)
  - Lost packets are retransmitted
- Timeouts used to detect lost packets
  - Requires RTO calculation
  - Requires sender to maintain data until it is ACKed

Sliding Window Revisited



- Sending side
  - **`LastByteAcked`** <= **`LastByteSent`**
  - **`LastByteSent`** <= **`LastByteWritten`**
  - buffer bytes between **`LastByteAcked`** and **`LastByteWritten`**

- Receiving side
  - **`LastByteRead`** < **`NextByteExpected`**
  - **`NextByteExpected`** <= **`LastByteRcvd +1`**
  - buffer bytes between **`NextByteRead`** and **`LastByteRcvd`**

Flow Control in TCP

- Send buffer size: **MaxSendBuffer**
- Receive buffer size: **MaxRcvBuffer**
- Receiving side
  - **LastByteRcvd** - **LastByteRead** < = **MaxRcvBuffer**
  - **AdvertisedWindow** = **MaxRcvBuffer** - (**LastByteRcvd** - **LastByteRead**)
- 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 sending one byte seg. when **AdvertisedWindow = 0**

Keeping the Pipe Full

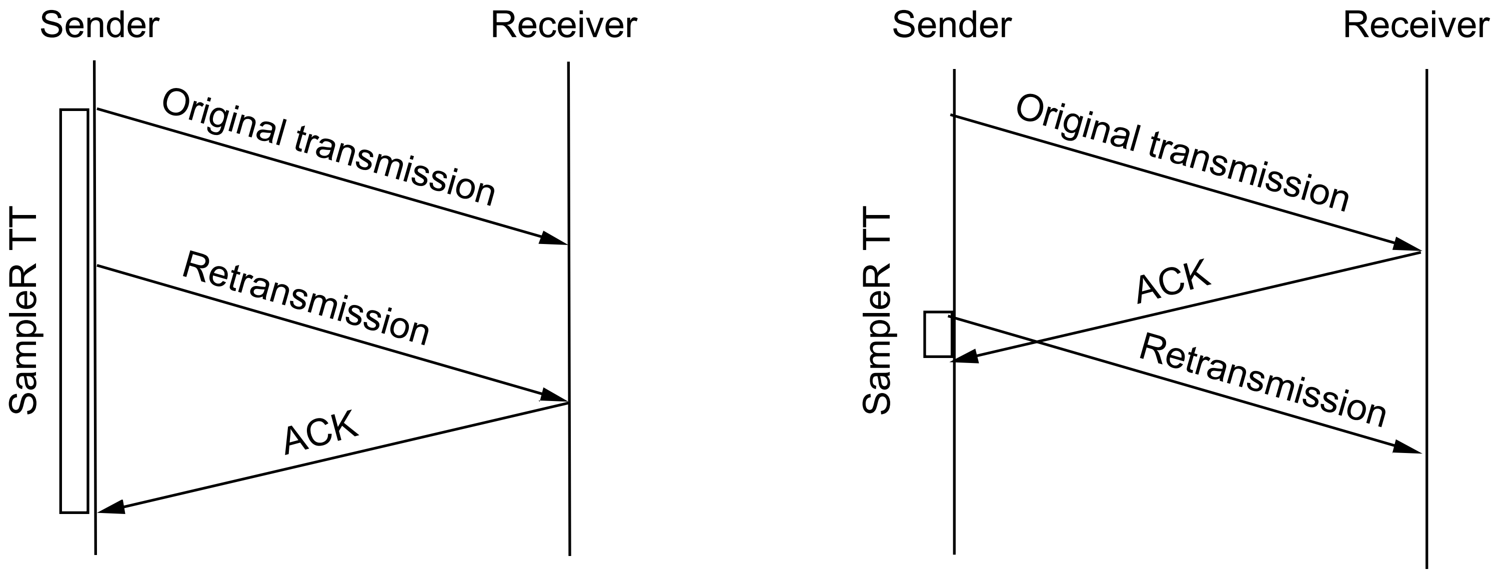
- 16-bit **`AdvertisedWindow`** controls amount of pipelining
- Assume RTT of 100ms
- Add scaling factor extension to header to enable larger windows

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

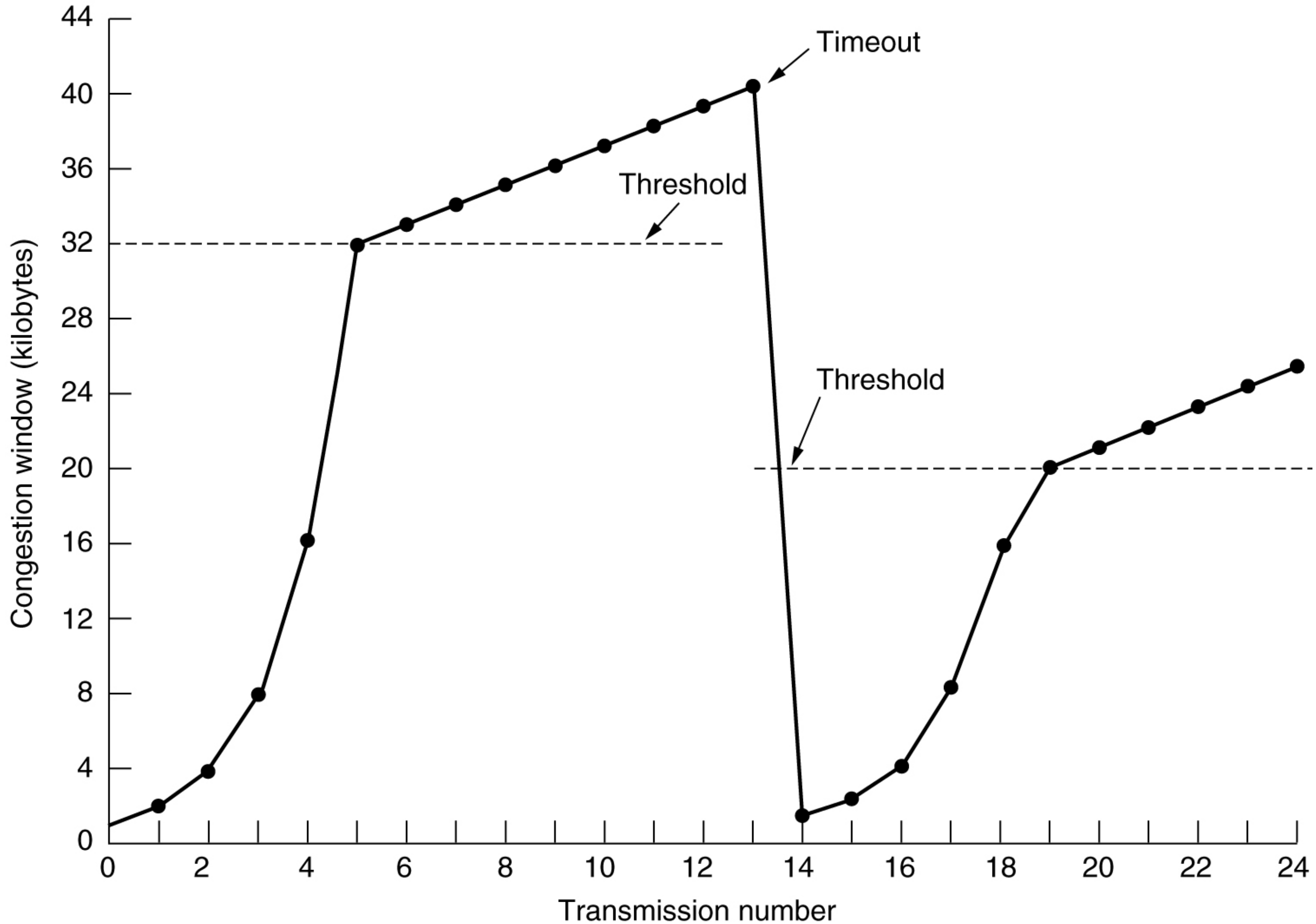
Making TCP More Efficient

- Delayed acknowledgements
  - Delay for about 200ms
  - Try to piggyback ACKs with data
- Acknowledge every other packet
  - Many instances in transmission sequence which require an ACK
- Don’t forget Nagle’s algorithm
  - Can be switched off

Karn/Partridge Algorithm for RTO



- Two degenerate cases with timeouts and RTT measurements
  - Solution: Do not sample RTT when retransmitting
- After each retransmission, set next RTO to be double the value of the last
  - Exponential backoff is well known control theory method
  - Loss is most likely caused by congestion so be careful


Timeout
Threshold
Threshold
Congestion window (kilobytes)
Transmission number
0
4
8
12
16
20
24
28
32
36
40
44
0
2
4
6
8
10
12
14
16
18
20
22
24

Jacobson/ Karels Algorithm

- In late ’80s, Internet was suffering from *congestion collapse*
- New Calculations for average RTT – Jacobson ’88
- Variance is not considered when setting timeout value
  - If variance is small, we could set RTO = EstRTT
  - If variance is large, we may need to set RTO > 2 x EstRTT
- New algorithm calculates both variance and mean for RTT
- **`Diff`** = **`sampleRTT`** - **`EstRTT`**
- **`EstRTT = EstRTT + ( d x Diff)`**
- **`Dev = Dev + d ( |Diff| - Dev)`**
  - Initially settings for **`EstRTT`** and **`Dev`** will be given to you
  - where d is a factor between 0 and 1
  - typical value is 0.125

Jacobson/ Karels contd.

- **`TimeOut`** = μ x **`EstRTT`** + ϕ x **`Dev`**
  - where $\mu = 1$ and $\phi = 4$
- When variance is small, TimeOut is close to EstRTT
- When variance is large Dev dominates the calculation
- Another benefit of this mechanism is that it is very efficient to implement in code (does not require floating point)
- Notes
  - algorithm only as good as granularity of clock (500ms on Unix)
  - accurate timeout mechanism important to congestion control (later)
- These issues have been studied and dealt with in new RFC's for RTO calculation.
- TCP RENO uses Jacobson/Karels