# PART XII

# RELIABLE STREAM TRANSPORT SERVICE (TCP)

# Transmission Control Protocol (TCP)

- Major transport service in the TCP/IP suite
- Used for most Internet applications (esp. World Wide Web)

# TCP Characteristics

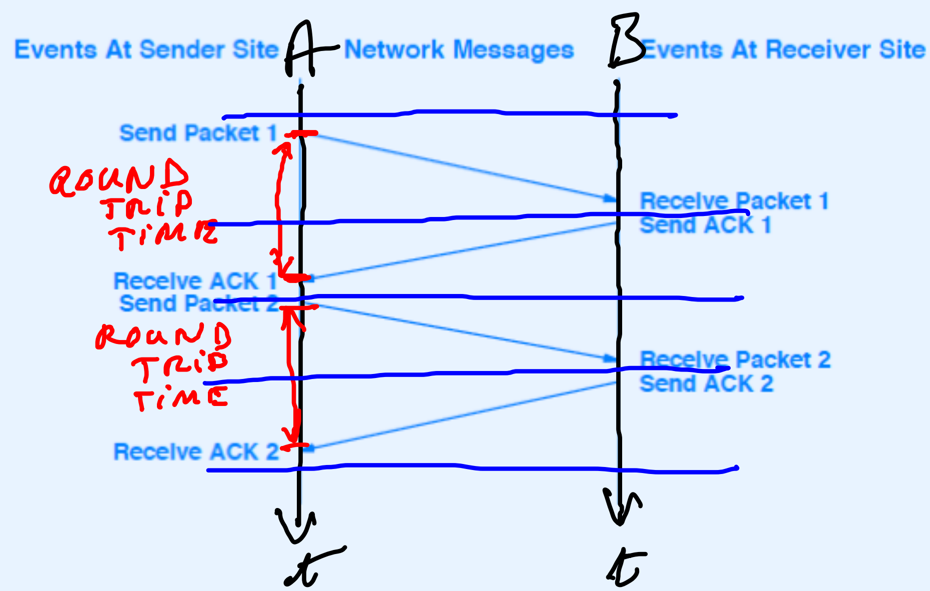
- Stream orientation
- Virtual circuit connection
- Buffered transfer
- Unstructured stream
- Full duplex connection
- Reliability

1 2 3 4 5 6 7 8 9

## Providing Reliability

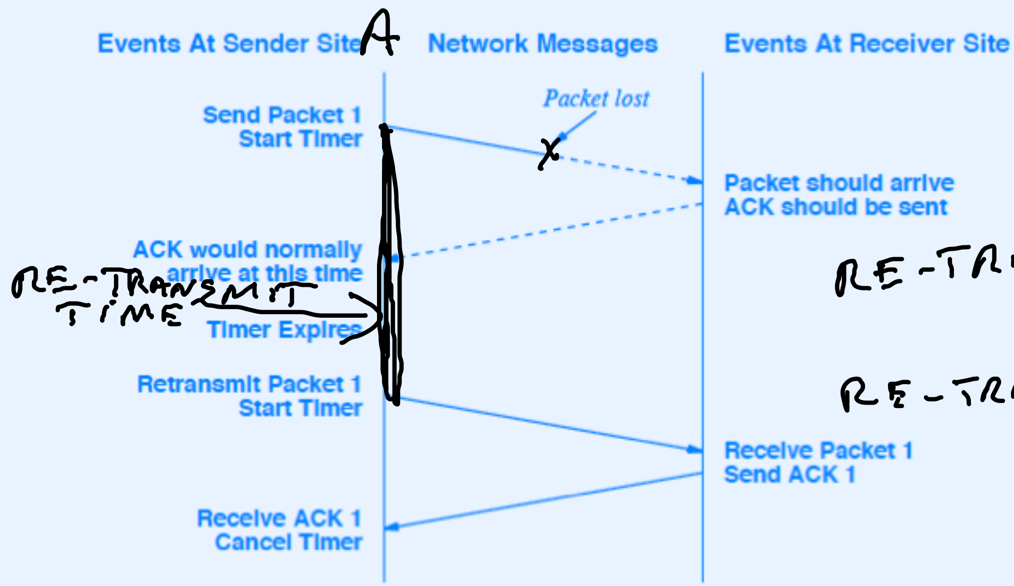
- Traditional technique: Positive Acknowledgement with Retransmission (PAR)
  - Receiver sends *acknowledgement* when data arrives
  - Sender starts timer whenever transmitting
  - Sender retransmits if timer expires before acknowledgement arrives

# Illustration Of Acknowledgements




- Time moves from top to bottom in the diagram

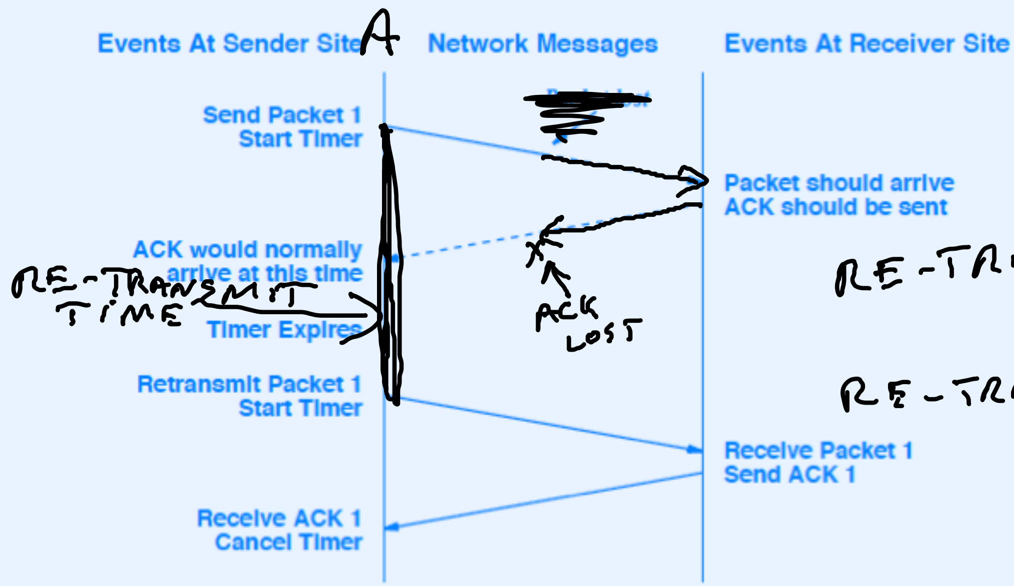
# Illustration Of Recovery After Packet Loss




RE-TRANSMIT TIME > ROUND TRIP TIME

RE-TRANSMIT TIME = 2* ROUND TRIP TIME

# Illustration Of Recovery After Packet Loss




RE-TRANSMIT TIME > ROUND TRIP TIME

RE-TRANSMIT TIME = 2 × ROUND TRIP TIME

# The Problem With Simplistic PAR

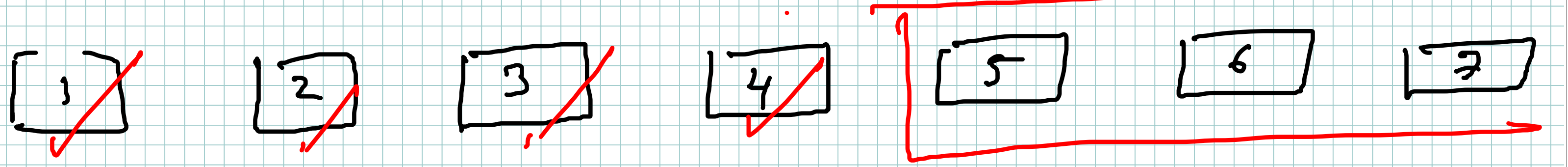
*A simple positive acknowledgement protocol wastes a substantial amount of network bandwidth because it must delay sending a new packet until it receives an acknowledgement for the previous packet.*

- Problem is especially severe if network has long latency

# Solving The Problem

- Allow multiple packets to be outstanding at any time
- Still require acknowledgements and retransmission
- Known as *sliding window*

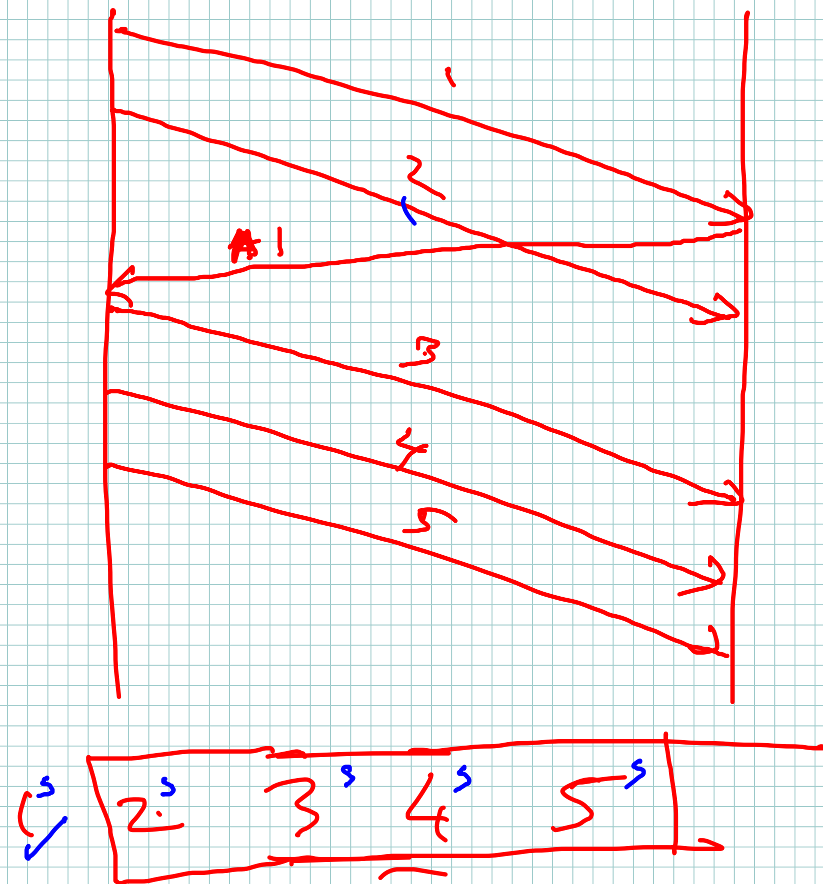
# FIXED SLIDING WINDOW PROTOCOL

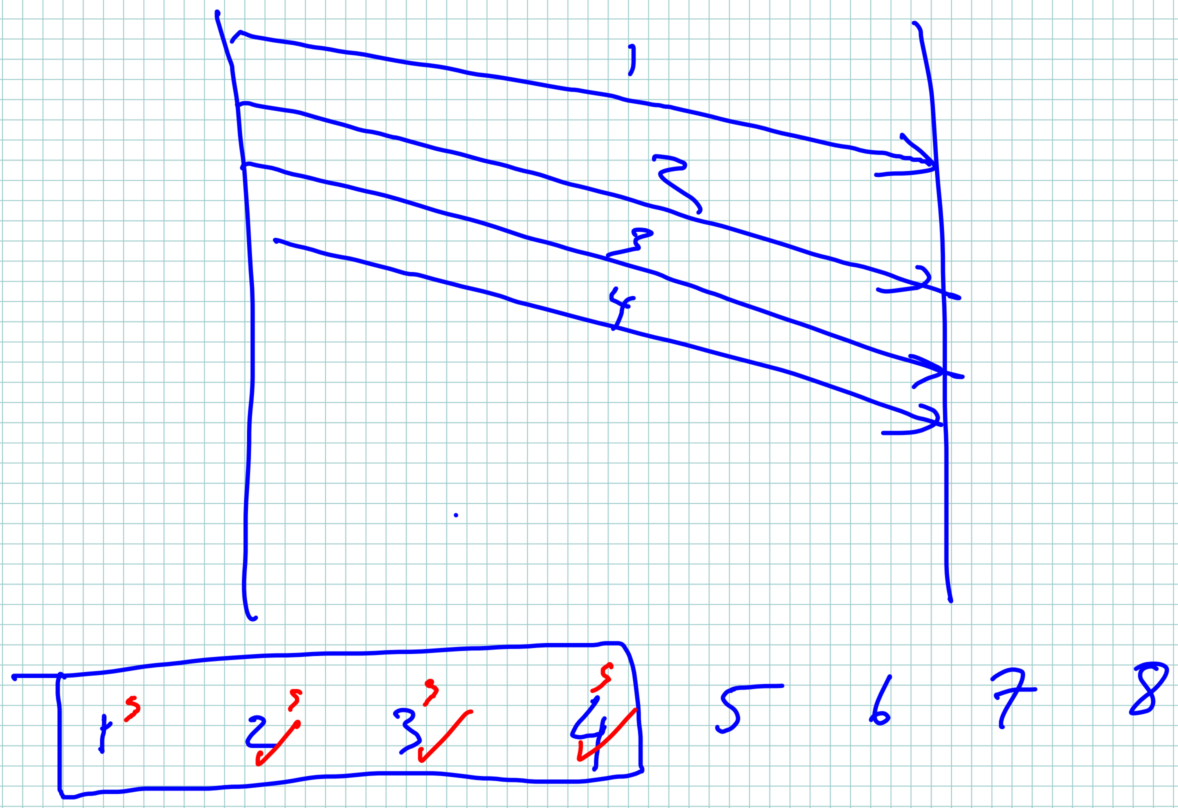



MESSAGES IN WINDOW SENT RIGHT AWAY

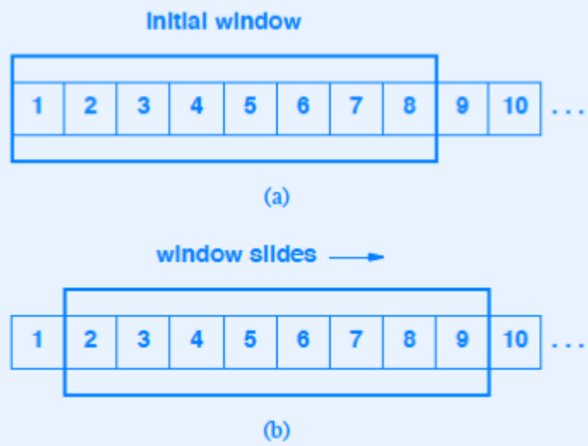
MESSAGES TO LEFT OF WINDOW ARE SENT & ACKed

MESSAGES TO RIGHT OF WINDOW NOT SENT

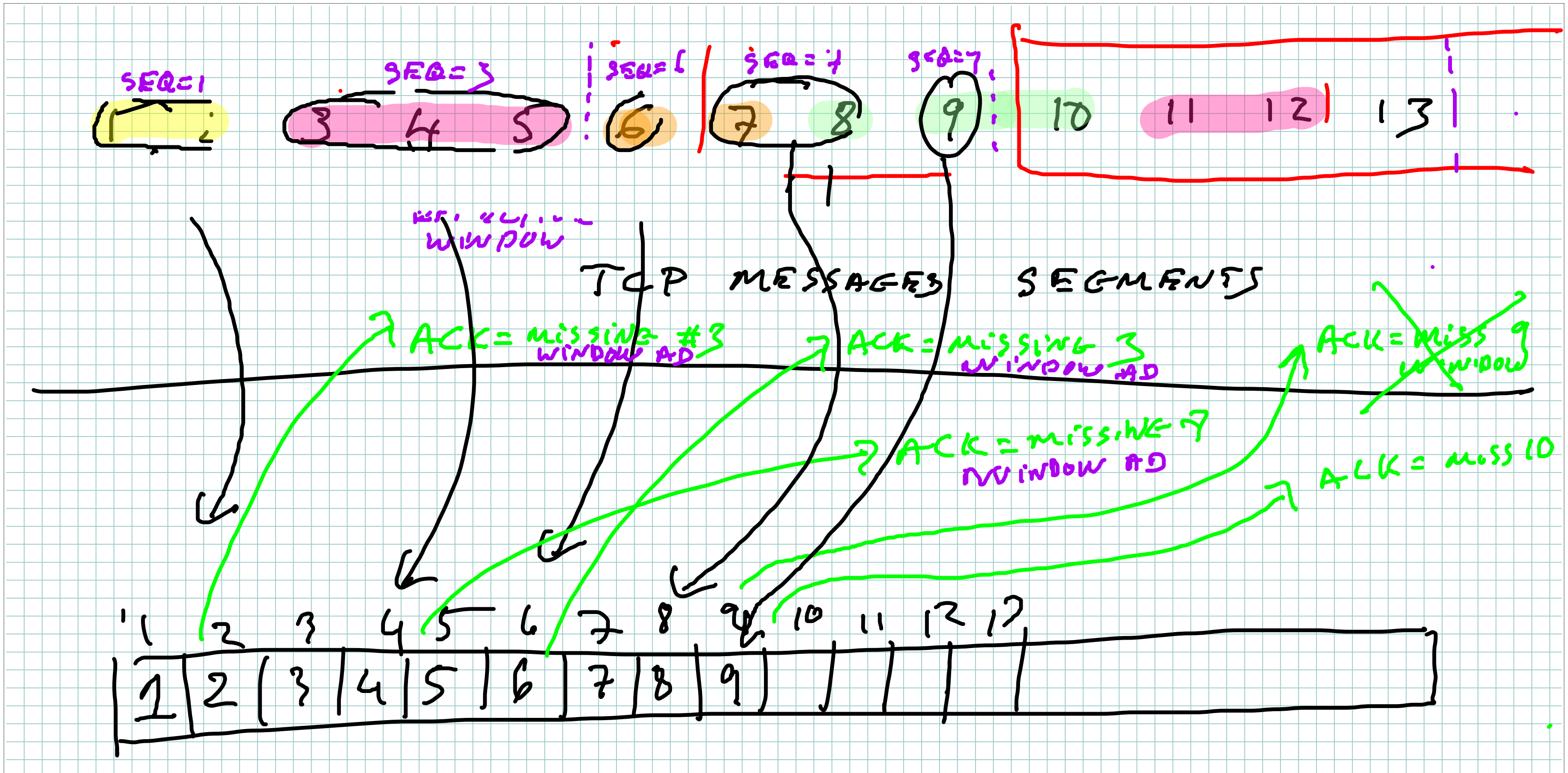

1
2
A1
3
4
5
2 3 4 5


1
2
3
4
1 2 3 4 5 6 7 8

# Illustration Of Sliding Window




- Window size is fixed
- As acknowledgement arrives, window moves forward

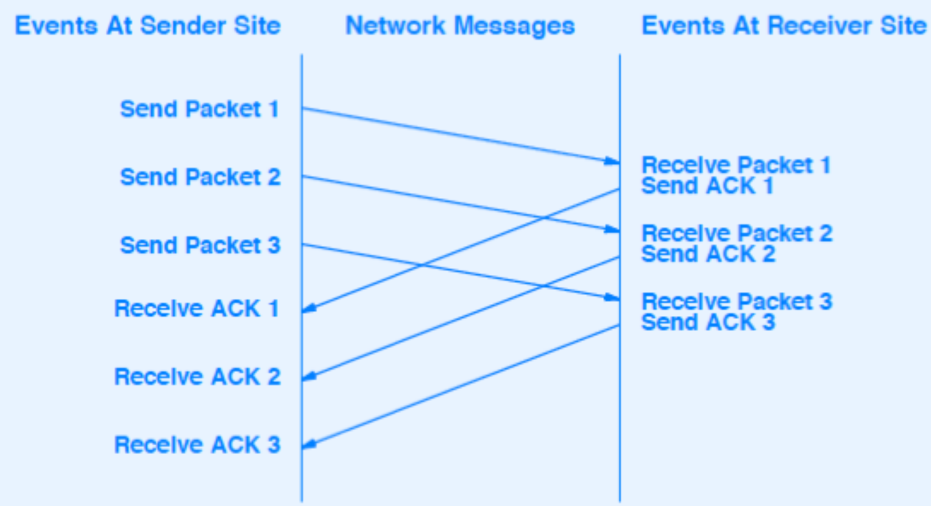


SEQ=1
SEQ=3
TCP MESSAGES SEGMENTS
ACK= MISSING #3
WINDOW AD
ACK = MISSING 3
WINDOW AD
ACK = MISSING 7
WINDOW AD
~~ACK = MISS 9~~
~~WINDOW~~
ACK = MISS 10

# Why Sliding Window Works

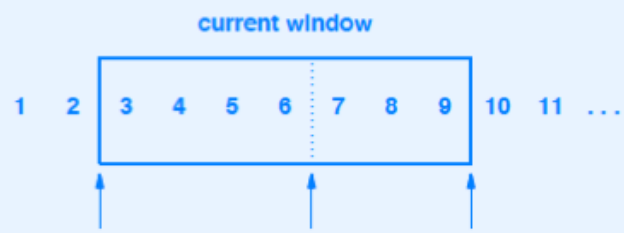
*Because a well-tuned sliding window protocol keeps the network completely saturated with packets, it obtains substantially higher throughput than a simple positive acknowledgement protocol.*

# Illustration Of Sliding Window



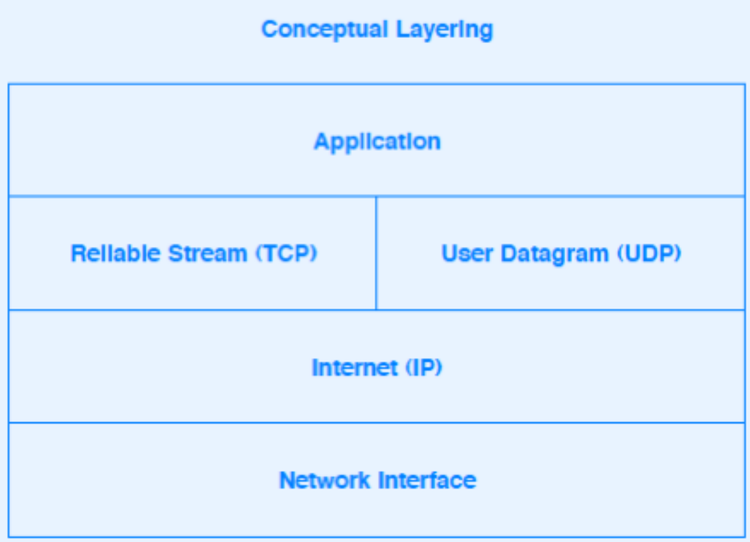
# Sliding Window Used By TCP

- Measured in byte positions
- Illustration




- Bytes through 2 are acknowledged
- Bytes 3 through 6 not yet acknowledged
- Bytes 7 though 9 waiting to be sent
- Bytes above 9 lie outside the window and cannot be sent

# Layering Of The Three Major Protocols



# TCP Ports, Connections, And Endpoints

- Endpoint of communication is application program
- TCP uses protocol port number to identify application
- TCP connection between two endpoints identified by four items
  - Sender's IP address
  - Sender's protocol port number
  - Receiver's IP address
  - Receiver's protocol port number

# TCP Ports, Connections, And Endpoints

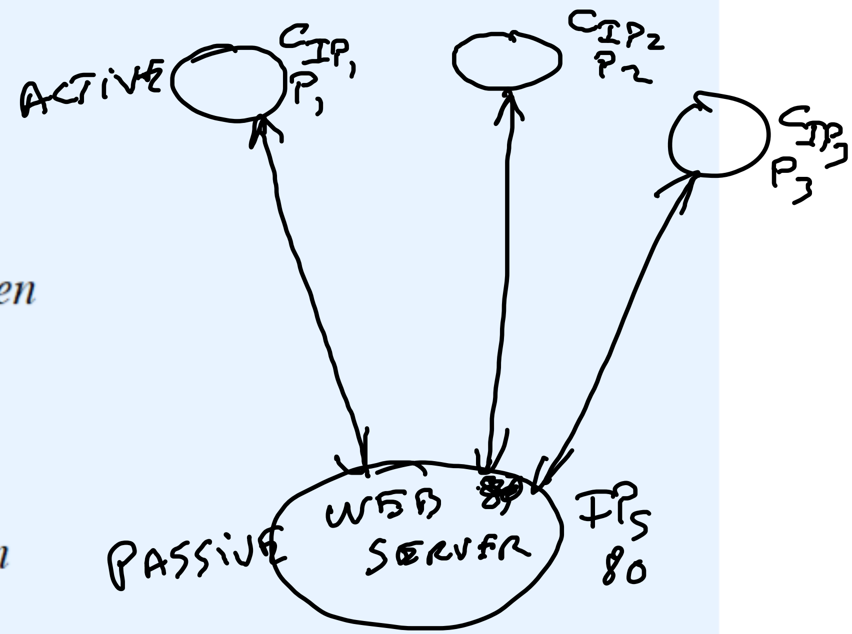
| | |
|---|---|
| Connection from: | |
| host 18.26.0.36 port 1069 | (18.26.0.36, 1069) |
| to | |
| host 128.10.2.3 port 25 | (128.10.2.3, 25) |
| Connection from: | |
| host 128.9.0.32 port 1184 | (128.9.0.32, 1184) |
| to | |
| host 128.10.2.3 port 53 | (128.10.2.3, 53) |
| Connection from: | |
| host 128.2.254.139 port 1184 | (128.2.254.139, 1184) |
| to | |
| host 128.10.2.3 port 53 | (128.10.2.3, 53) |

# An Important Idea About Port Numbers

*Because TCP identifies a connection by a pair of endpoints, a given TCP port number can be shared by multiple connections on the same machine.*

# Passive And Active Opens

- Two sides of a connection
- One side waits for contact
  - A server program
  - Uses TCP's *passive open*
- One side initiates contact
  - A client program
  - Uses TCP's *active open*



# TCP Segment Format

**TCP Header**

<table>
<tr><th>Offsets</th><th>Octet</th><th colspan="8">0</th><th colspan="8">1</th><th colspan="8">2</th><th colspan="8">3</th></tr>
<tr><th>Octet</th><th>Bit</th><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>11</th><th>12</th><th>13</th><th>14</th><th>15</th><th>16</th><th>17</th><th>18</th><th>19</th><th>20</th><th>21</th><th>22</th><th>23</th><th>24</th><th>25</th><th>26</th><th>27</th><th>28</th><th>29</th><th>30</th><th>31</th></tr>
<tr><th>0</th><th>0</th><td colspan="16">Source port</td><td colspan="16">Destination port</td></tr>
<tr><th>4</th><th>32</th><td colspan="32">Sequence number</td></tr>
<tr><th>8</th><th>64</th><td colspan="32">Acknowledgment number (if ACK set)</td></tr>
<tr><th>12</th><th>96</th><td colspan="4">Data offset</td><td colspan="3">Reserved</td><td>NS</td><td>CWR</td><td>ECE</td><td>URG</td><td>ACK</td><td>PSH</td><td>RST</td><td>SYN</td><td>FIN</td><td colspan="16">Window Size</td></tr>
<tr><th>16</th><th>128</th><td colspan="16">Checksum</td><td colspan="16">Urgent pointer (if URG set)</td></tr>
<tr><th>20<br>...</th><th>160<br>...</th><td colspan="32">Options (if Data Offset > 5,padded at end with "0" bytes if necessary)<br>...</td></tr>
</table>

| BEGINNING OF PAYLOAD (DATA) |
|---|
| . . . |

- Offset specifies header size (offset of data) in 32-bit words

# Flow Control And TCP Window

- Receiver controls flow by telling sender size of currently available buffer measured in bytes
- Called *window advertisement*
- Each segment, including data segments, specifies size of window *beyond acknowledged byte*
- Window size may be zero (receiver cannot accept additional data at present)
- Receiver can send additional acknowledgement later when buffer space becomes available

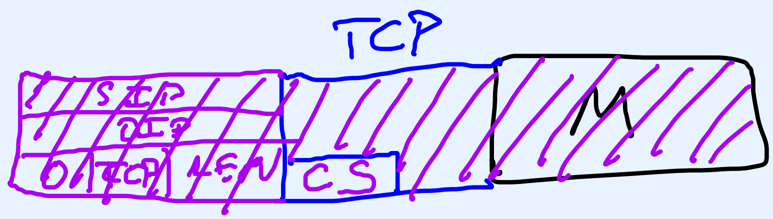
# TCP Checksum Computation

- Covers entire segment (header plus data)
- Required (unlike UDP)
- Pseudo header included in computation as with UDP

# TCP Pseudo Header

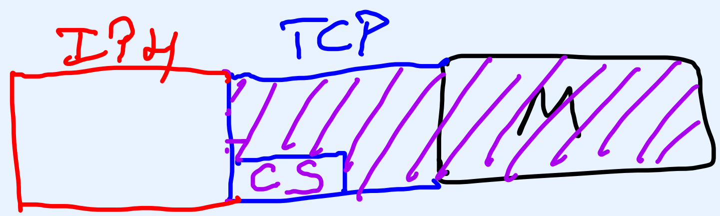
0 8 16 31

| | | |
|---|---|---|
| SOURCE IP ADDRESS | | |
| DESTINATION IP ADDRESS | | |
| ZERO | PROTOCOL | TCP LENGTH |



# TCP Pseudo Header

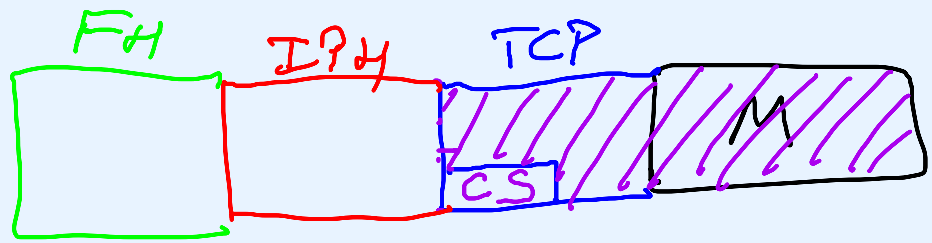
| 0 | 8 | 16 31 |
|---|---|---|
| SOURCE IP ADDRESS | | |
| DESTINATION IP ADDRESS | | |
| ZERO | PROTOCOL | TCP LENGTH |



# TCP Pseudo Header

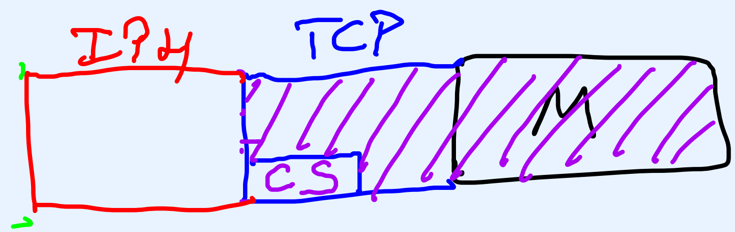
0 8 16 31

| | | |
|---|---|---|
| SOURCE IP ADDRESS | | |
| DESTINATION IP ADDRESS | | |
| ZERO | PROTOCOL | TCP LENGTH |



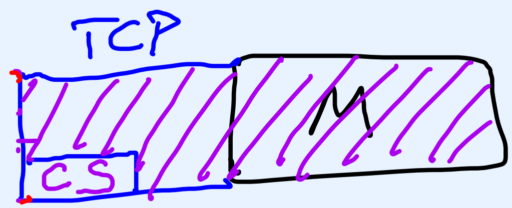
# TCP Pseudo Header

| 0 | 8 | 16 | 31 |
|---|---|---|---|
| SOURCE IP ADDRESS | | | |
| DESTINATION IP ADDRESS | | | |
| ZERO | PROTOCOL | TCP LENGTH | |



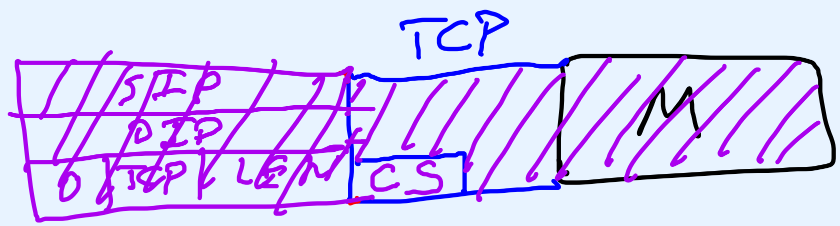
# TCP Pseudo Header

<table>
<tr><th>0</th><th>8</th><th>16</th><th>31</th></tr>
<tr><td colspan="4">SOURCE IP ADDRESS</td></tr>
<tr><td colspan="4">DESTINATION IP ADDRESS</td></tr>
<tr><td>ZERO</td><td>PROTOCOL</td><td colspan="2">TCP LENGTH</td></tr>
</table>



# TCP Pseudo Header

<table>
<tr><td>0</td><td>8</td><td>16</td><td>31</td></tr>
<tr><td colspan="4">SOURCE IP ADDRESS</td></tr>
<tr><td colspan="4">DESTINATION IP ADDRESS</td></tr>
<tr><td>ZERO</td><td>PROTOCOL</td><td colspan="2">TCP LENGTH</td></tr>
</table>



# TCP Pseudo Header

<table>
<tr><th>0</th><th>8</th><th>16</th><th>31</th></tr>
<tr><td colspan="4">SOURCE IP ADDRESS</td></tr>
<tr><td colspan="4">DESTINATION IP ADDRESS</td></tr>
<tr><td>ZERO</td><td>PROTOCOL</td><td colspan="2">TCP LENGTH</td></tr>
</table>

M

## TCP Retransmission

- Designed for Internet environment
  - Delays on one connection vary over time
  - Delays vary widely between connections
- Fixed value for timeout will fail
  - Waiting too long introduces unnecessary delay
  - Not waiting long enough wastes network bandwidth with unnecessary retransmission
- Retransmission strategy must be adaptive

# Adaptive Retransmission

- TCP keeps estimate of round-trip time (RTT) on each connection
- Round-trip estimate derived from observed delay between sending segment and receiving acknowledgement
- Timeout for retransmission based on current round-trip estimate

$$R_{TT_{CUR}} = \alpha R_{TT_{CUR}} + \beta R_{TT_m}$$

$$\alpha + \beta = 1$$

$\alpha = 1$

$\beta = 0$

$R_{TT_{CUR}}$ IS CONSTANT

$\alpha = 0$

$\beta = 1$

$R_{TT_{CUR}} = R_{TT_m}$

$\alpha = 0.5$

$\beta = 0.5$

| $R_{TT_{CUR}}$ | $R_{TT_m}$ |
|---|---|
| 1 | 1 ✓ |
| 1.5 | 2 ✓ |
| 1.25 | 1 ✓ |
| 2.125 | 3 ✓ |
| 3.0625 | 4 ✓ |

$R_{TT_{CUR}} = 2.125$

$R_{TT_{CUR}} = 0.5 \times 2.125 + 0.5 \times 4$

$= 1.0625 + 2$

$$RTT_{CUR} = \alpha \, RTT_{CUR} + \beta \, RTT_{m}$$

$$\alpha + \beta = 1$$

$\alpha = 0.8$
$\beta = 0.2$

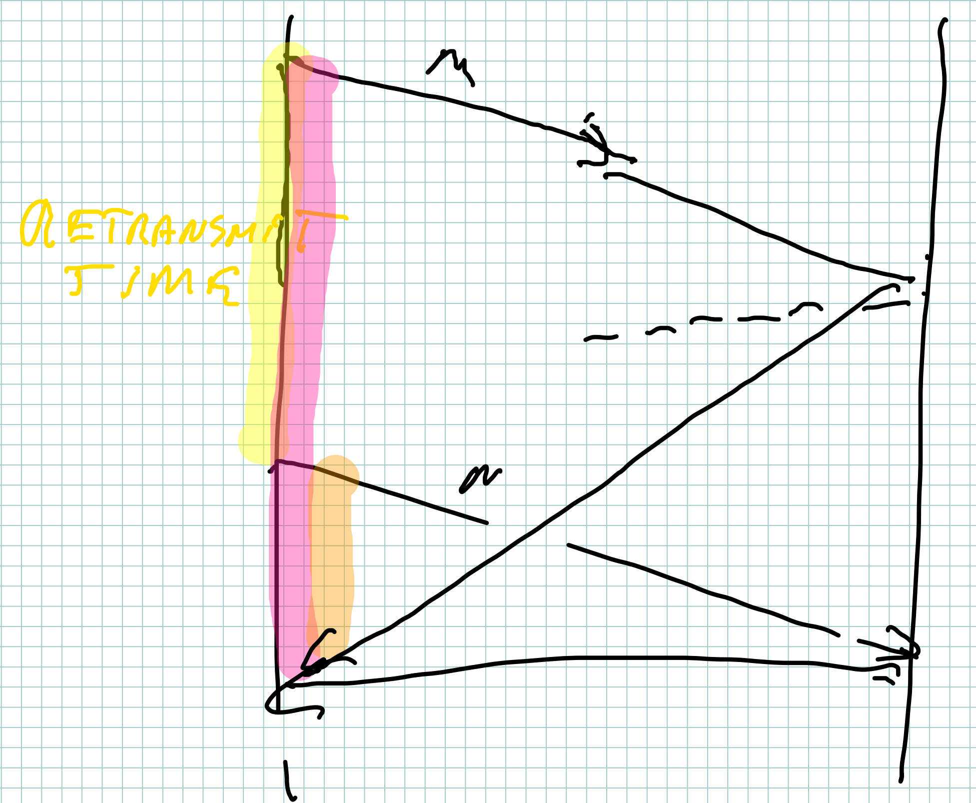
MORE WEIGHT TO HISTORY

$\alpha = 0.2$
$\beta = 0.8$

MORE WEIGHT TO ~~LAST~~ MEASUREMENT

# Difficulties With Adaptive Retransmission

- The problem is knowing when to retransmit
- Segments or ACKs can be lost or delayed, making round-trip estimation difficult or inaccurate
- Round-trip times vary over several orders of magnitude between different connections
- Traffic is bursty, so round-trip times fluctuate wildly on a single connection


n
RETRANSMIT
TIME
n

# Difficulties With Adaptive Retransmission (continued)

- Load imposed by a single connection can congest routers or networks
- Retransmission can *cause* congestion
- Because an internet contains diverse network hardware technologies, there may be little or no control for intra-network congestion

## Solution: Smoothing

- Adaptive retransmission schemes keep a statistically smoothed round-trip estimate
- Smoothing keeps running average from fluctuating wildly, and keeps TCP from overreacting to change
- Difficulty: choice of smoothing scheme

# Original Smoothing Scheme

- Let RTT be current (old) average round-trip time
- Let NRT be a new sample
- Compute

$$RTT = \alpha * RTT + \beta * NRT$$

where

$$\alpha + \beta = 1$$

- Example: $\alpha = .8$, $\beta = .2$
- Large $\alpha$ makes estimate less susceptible to a single long delay (more stable)
- Large $\beta$ makes estimate track changes in round-trip time quickly

## Problems With Original Scheme

- Associating ACKs with transmissions
  - TCP acknowledges receipt of data, not receipt of transmission
  - Assuming ACK corresponds to most recent transmission can cause instability in round-trip estimate (Cypress syndrome)
  - Assuming ACK corresponds to first transmission can cause unnecessarily long timeout
  - Both assumptions lead to lower throughput

# Partridge / Karn Scheme†

- Solves the problem of associating ACKs with correct transmission
- Specifies ignoring round-trip time samples that correspond to retransmissions
- Separates timeout from round-trip estimate for retransmitted packets

†Also called *Karn's Algorithm*

## Partridge / Karn Scheme
### (continued)

- Starts (as usual) with retransmission timer as a function of round-trip estimate
- Doubles retransmission timer value for each retransmission without changing round-trip estimate
- Resets retransmission timer to be function of round-trip estimate when ACK arrives for nonretransmitted segment

# Flow Control And Congestion

- Receiver advertises window that specifies how many additional bytes it can accept
- Window size of zero means sender must not send normal data (ACKs and urgent data allowed)
- Receiver can never decrease window beyond previously advertised point in sequence space
- Sender chooses effective window smaller than receiver's advertised window if congestion detected

# Jacobson / Karels
# Congestion Control

- Assumes long delays (packet loss) due to congestion
- Uses successive retransmissions as measure of congestion
- Reduces effective window as retransmissions increase
- Effective window is minimum of receiver's advertisement and computed quantity known as the *congestion window*

# Multiplicative Decrease

- In steady state (no congestion), the congestion window is equal to the receiver's window
- When segment lost (retransmission timer expires), reduce congestion window by half
- Never reduce congestion window to less than one maximum sized segment

# Jacobson / Karels Slow Start

- Used when starting traffic or when recovering from congestion
- Self-clocking startup to increase transmission rate rapidly as long as no packets are lost
- When starting traffic, initialize the congestion window to the size of a single maximum sized segment
- Increase congestion window by size of one segment each time an ACK arrives without retransmission

# Jacobson / Karels Congestion Avoidance

- When congestion first occurs, record one-half of last successful congestion window (flightsize) in a *threshold* variable
- During recovery, use slow start until congestion window reaches threshold
- Above threshold, slow down and increase congestion window by one segment per window (even if more than one segment was successfully transmitted in that interval)

## Jacobson / Karels Congestion Avoidance (continued)

- Increment window size on each ACK instead of waiting for complete window

  increase = segment / window

  Let *N* be segments per window, or

  N = congestion_window / max segment size

  so

  increase = segment / N

  = (MSS bytes / N)

  = MSS / (congestion_window/MSS)

  or

  increase = (MSS*MSS)/congestion_window

# Changes In Delay

- Original smoothing scheme tracks the mean but not changes
- To track changes, compute

$$\begin{aligned} DIFF &= SAMPLE - RTT \\ RTT &= RTT + \delta * DIFF \\ DEV &= DEV + \delta\,(|DIFF| - DEV) \end{aligned}$$

- DEV estimates mean deviation
- $\delta$ is fraction between 0 and 1 that weights new sample
- Retransmission timer is weighted average of RTT and DEV:

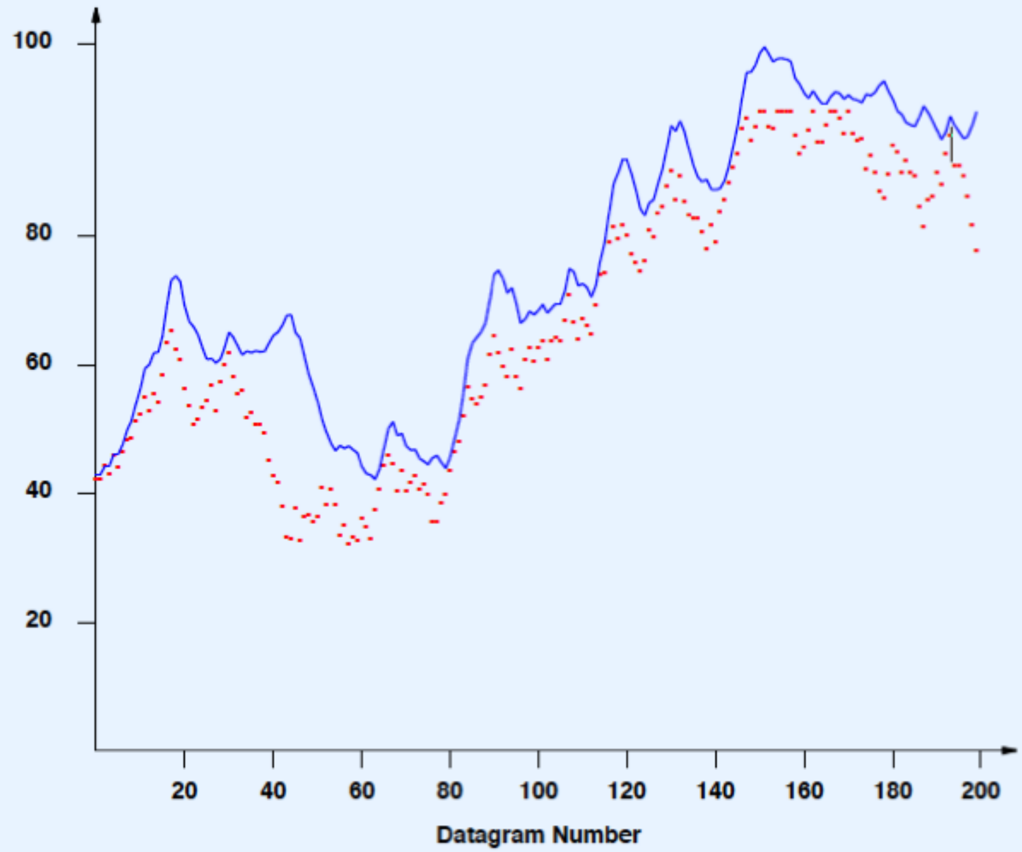
$$RTO = \mu * RTT + \phi * DEV$$

- Typically, $\mu = 1$ and $\phi = 4$

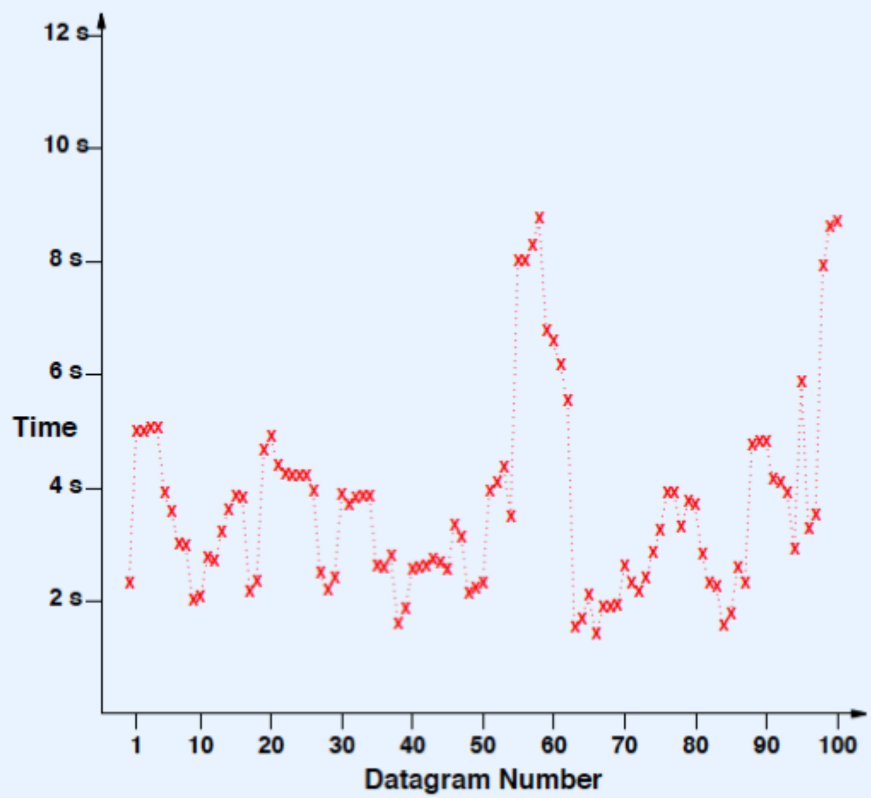
## Computing Estimated Deviation

- Extremely efficient (optimized) implementation possible
  - Scale computation by $2^n$
  - Use integer arithmetic
  - Choose $\delta$ to be $1/2^n$
  - Implement multiplication or division by powers of 2 with shifts
  - Research shows $n = 3$ works well

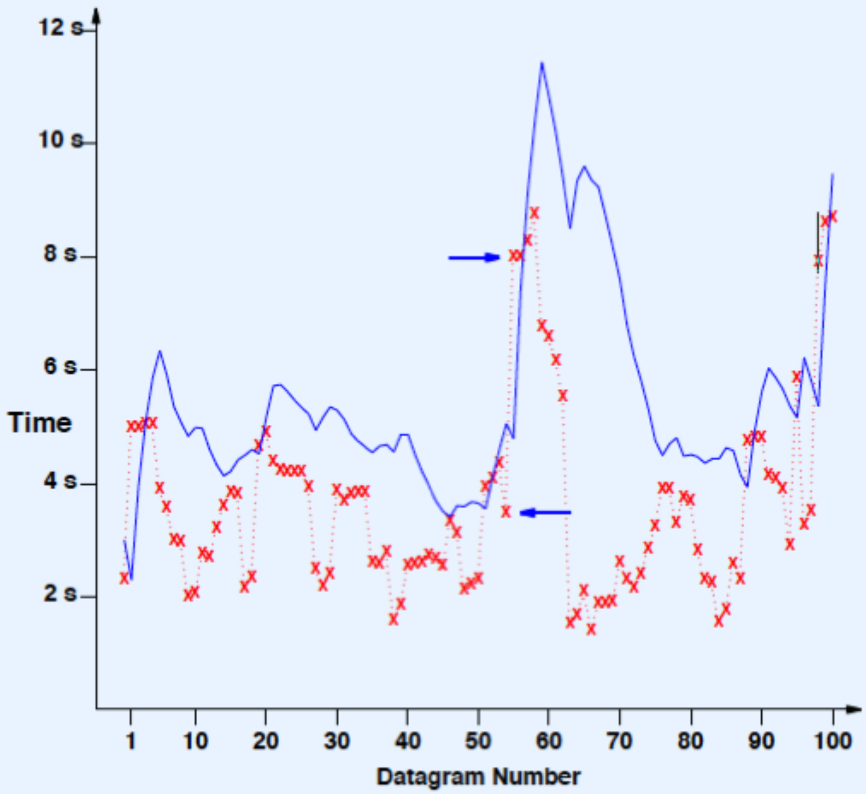
# TCP Round-Trip Estimation



# Measurement Of Internet Delays For 100 Successive Packets At 1 Second Intervals



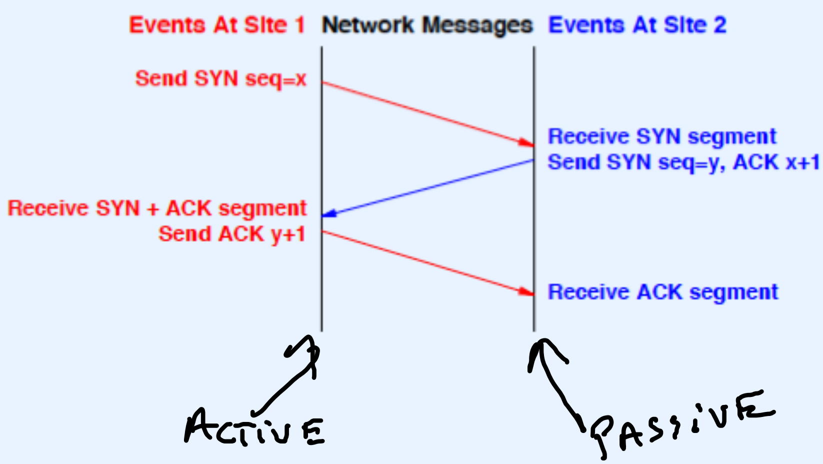
# TCP Round-Trip Estimation For Sampled Internet Delays

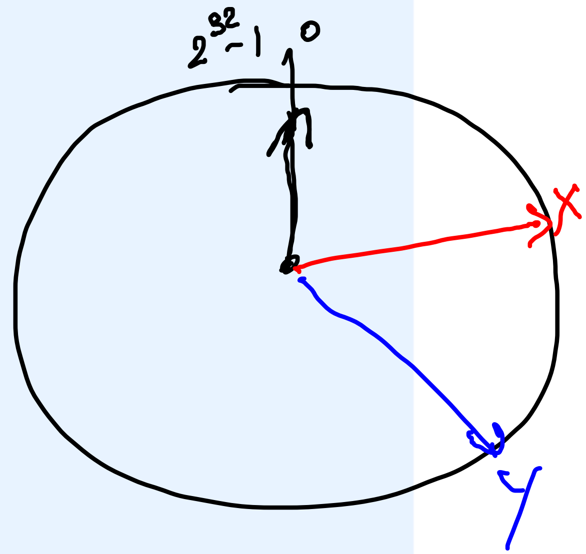


# TCP Details

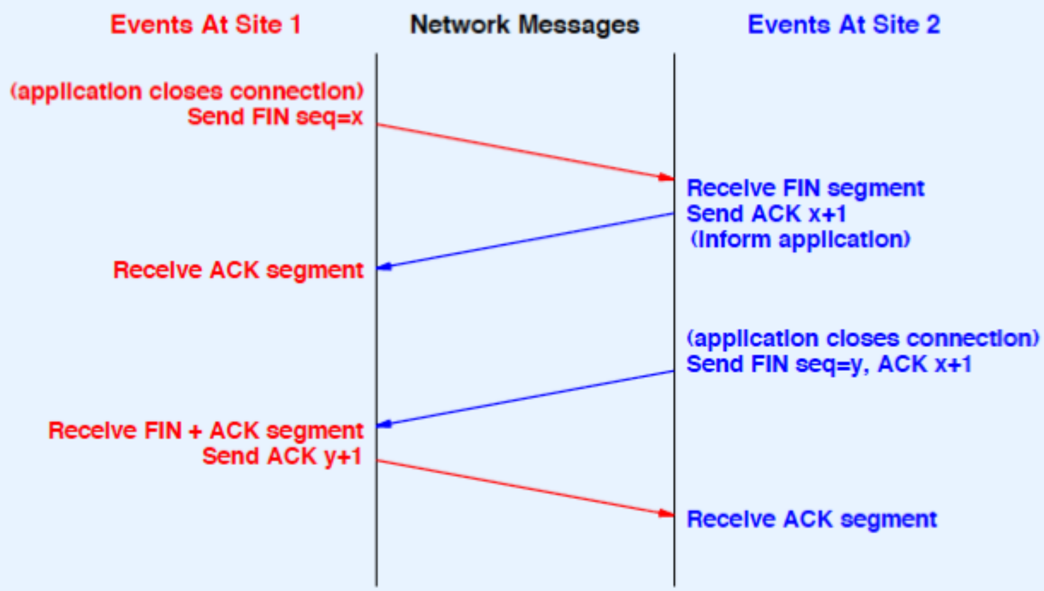
- Data flow may be shut down in one direction
- Connections started reliably, and terminated gracefully
- Connection established (and terminated) with a 3-way handshake

# 3-Way Handshake
# For Connection Startup

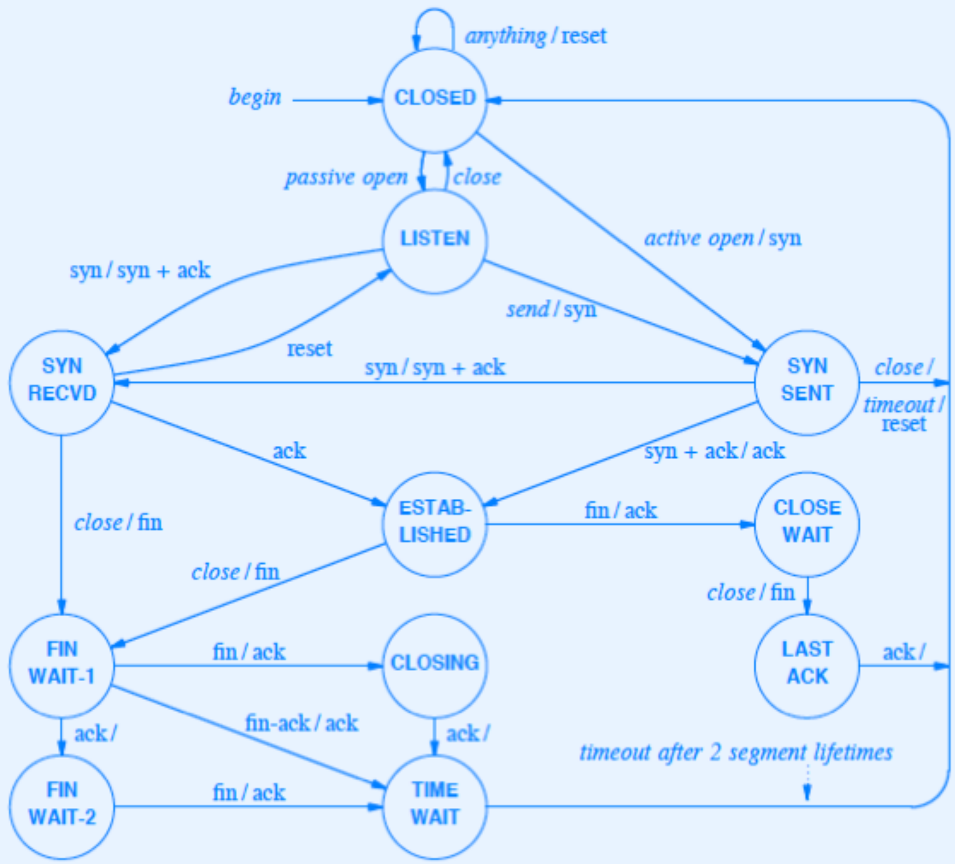


# 3-Way Handshake For Connection Shutdown



# TCP Finite State Machine



# TCP Urgent Data

- Segment with urgent bit set contains pointer to last octet of urgent data
- Urgent data occupies part of normal sequence space
- Urgent data can be retransmitted
- Receiving TCP should deliver urgent data to application ‘‘immediately’’ upon receipt

# TCP Urgent Data
## (continued)

- Two interpretations of standard
  - Out-of-band data interpretation
  - Data mark interpretation

# Data-Mark Interpretation Of Urgent Data

- Has become widely accepted
- Single data stream
- Urgent pointer marks end of urgent data
- TCP informs application that urgent data arrived
- Application receives all data in sequence
- TCP informs application when end of urgent data reached

# Data-Mark Interpretation Of Urgent Data (continued)

- Application
  - Reads all data from one stream
  - Must recognize start of urgent data
  - Must buffer normal data if needed later
- Urgent data marks *read* boundary

# Urgent Data Delivery

- Receiving application placed in *urgent mode*
- Receiving application leaves urgent mode after reading urgent data
- Receiving application acquires *all* available urgent data when in urgent mode

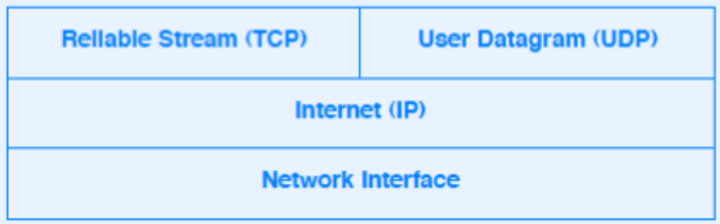
# Fast Retransmit

- Coarse-grained clock used to implement RTO
  - Typically 300 to 500ms per tick
- Timer expires up to 1s after segment dropped
- Fast retransmission
  - Sender uses three duplicate ACKs as trigger
  - Sender retransmits ‘‘early’’
  - Sender reduces congestion window to half

# Other TCP Details

- Silly Window Syndrome (SWS) avoidance
- Nagle algorithm
- Delayed ACKs
- For details, read the text

# Comparison Of UDP And TCP




- TCP and UDP lie between applications and IP
- Otherwise, completely different

# Comparison Of UDP and TCP

| UDP | TCP |
|---|---|
| between apps. and IP | between apps. and IP |
| packets called datagrams | packets called segments |
| | |
| unreliable | reliable |
| checksum optional | checksum required |
| connectionless | connection-oriented |
| record boundaries | stream interface |
| intended for LAN | useful over WAN or LAN |
| no flow control | flow control |
| 1-to-1, 1-many, many-1 | 1-to-1 |
| allows unicast, multicast or broadcast | unicast only |

# TCP Vs. UDP Traffic

Around 95% of all bytes and around 85-95% of all packets on the Internet are transmitted using TCP.

– Eggert, et. al. CCR

# Summary Of TCP

- Major transport service in the Internet
- Connection oriented
- Provides end-to-end reliability
- Uses adaptive retransmission
- Includes facilities for flow control and congestion avoidance
- Uses 3-way handshake for connection startup and shutdown