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)

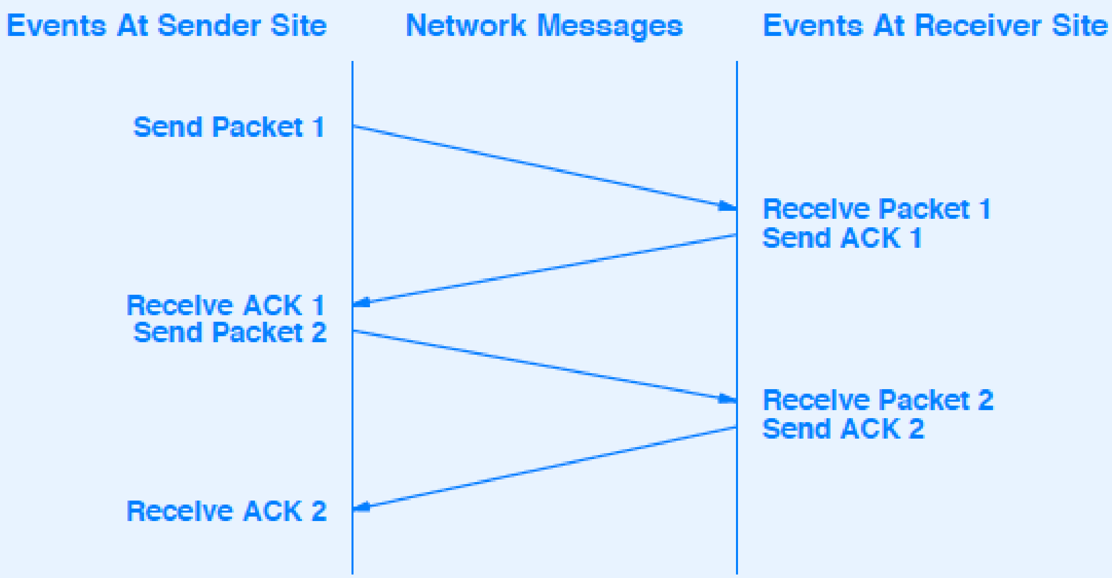
TCP Characteristics

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

Providing Reliability

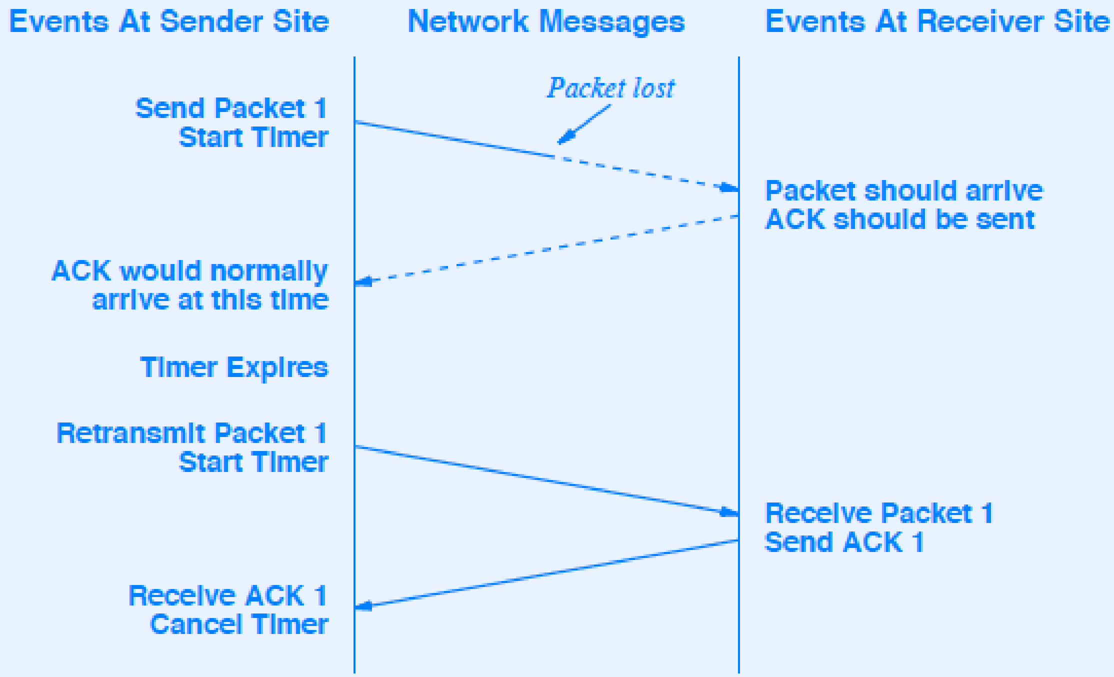
- Traditional technique: Positive Acknowledgement with Retransmission (PAR)
  - Receiver sends *acknowledgement* when data arrives
  - Sender starts timer whenever transmitting
  - Sender retransmits if timer expires before acknowledgement arrives

Illustration Of Acknowledgements



- Time moves from top to bottom in the diagram

Illustration Of Recovery After Packet Loss



The Problem With Simplistic PAR

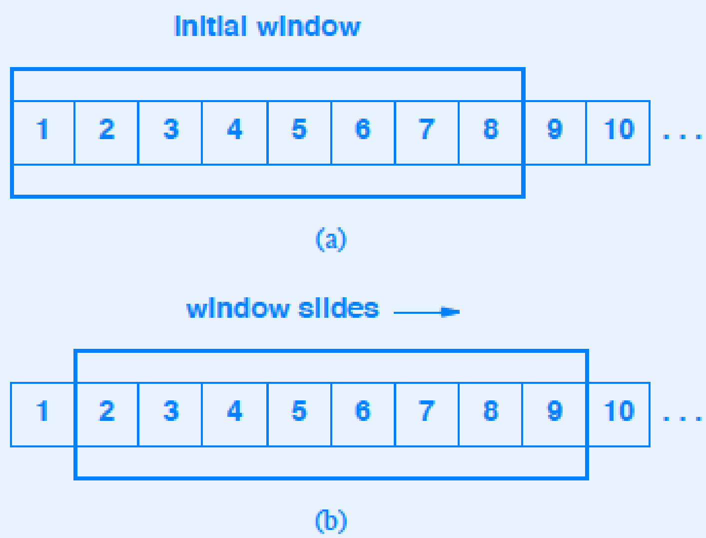
A simple positive acknowledgement protocol wastes a substantial amount of network bandwidth because it must delay sending a new packet until it receives an acknowledgement for the previous packet.

- 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*

Illustration Of Sliding Window

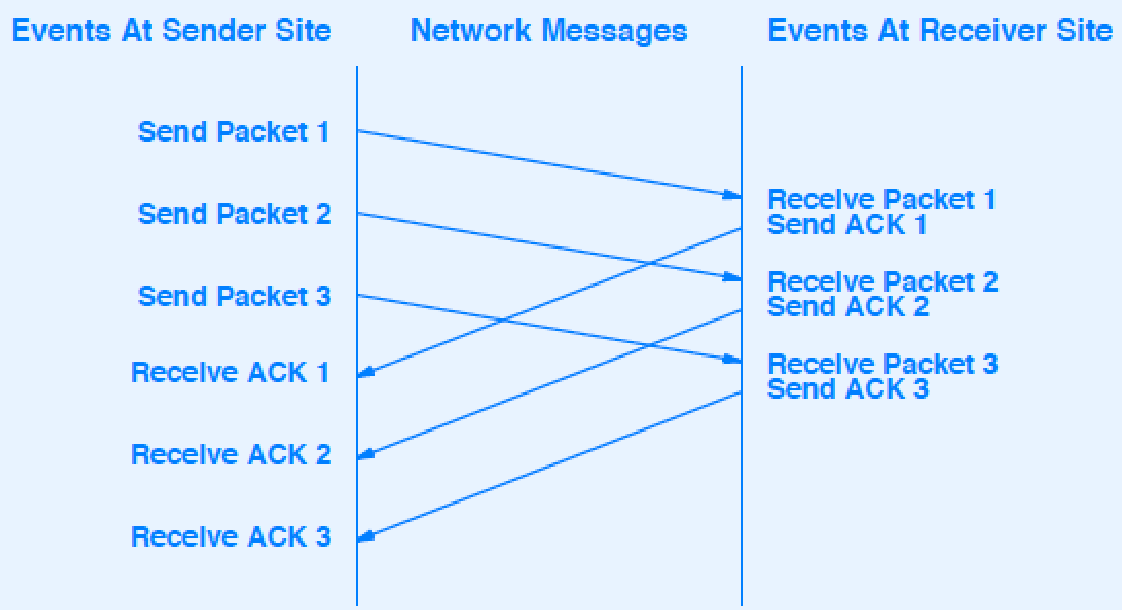


- Window size is fixed
- As acknowledgement arrives, window moves forward

Why Sliding Window Works

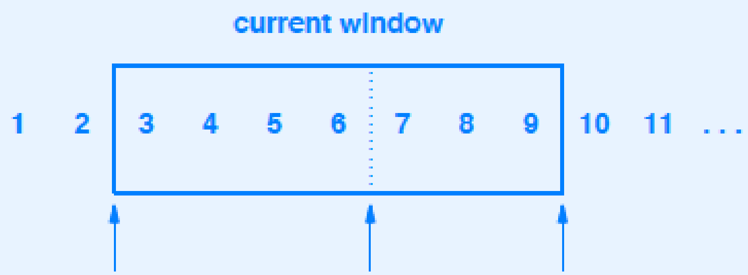
Because a well-tuned sliding window protocol keeps the network completely saturated with packets, it obtains substantially higher throughput than a simple positive acknowledgement protocol.

Illustration Of Sliding Window



Sliding Window Used By TCP

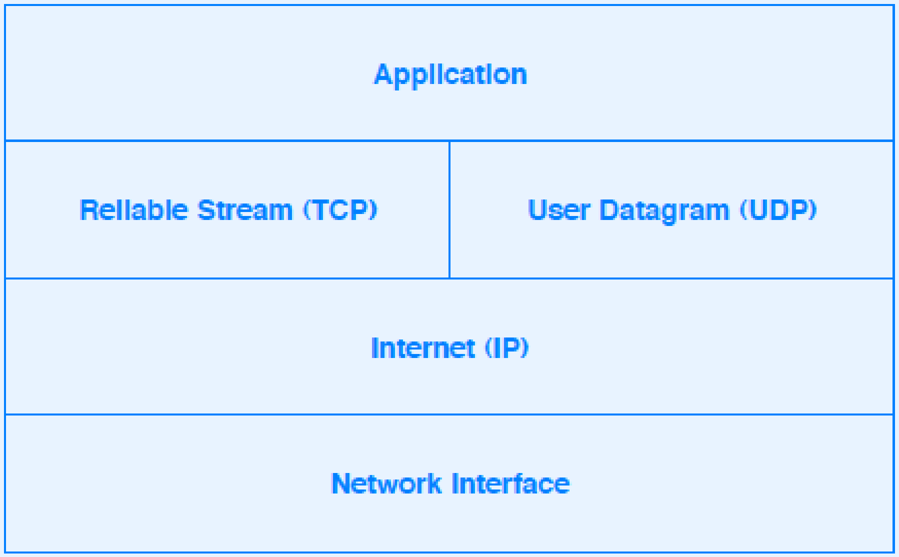
- Measured in byte positions
- Illustration



- 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

Conceptual Layering



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

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

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 α makes estimate less susceptible to a single long delay (more stable)
- Large β 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
- δ is fraction between 0 and 1 that weights new sample
- Retransmission timer is weighted average of RTT and DEV:

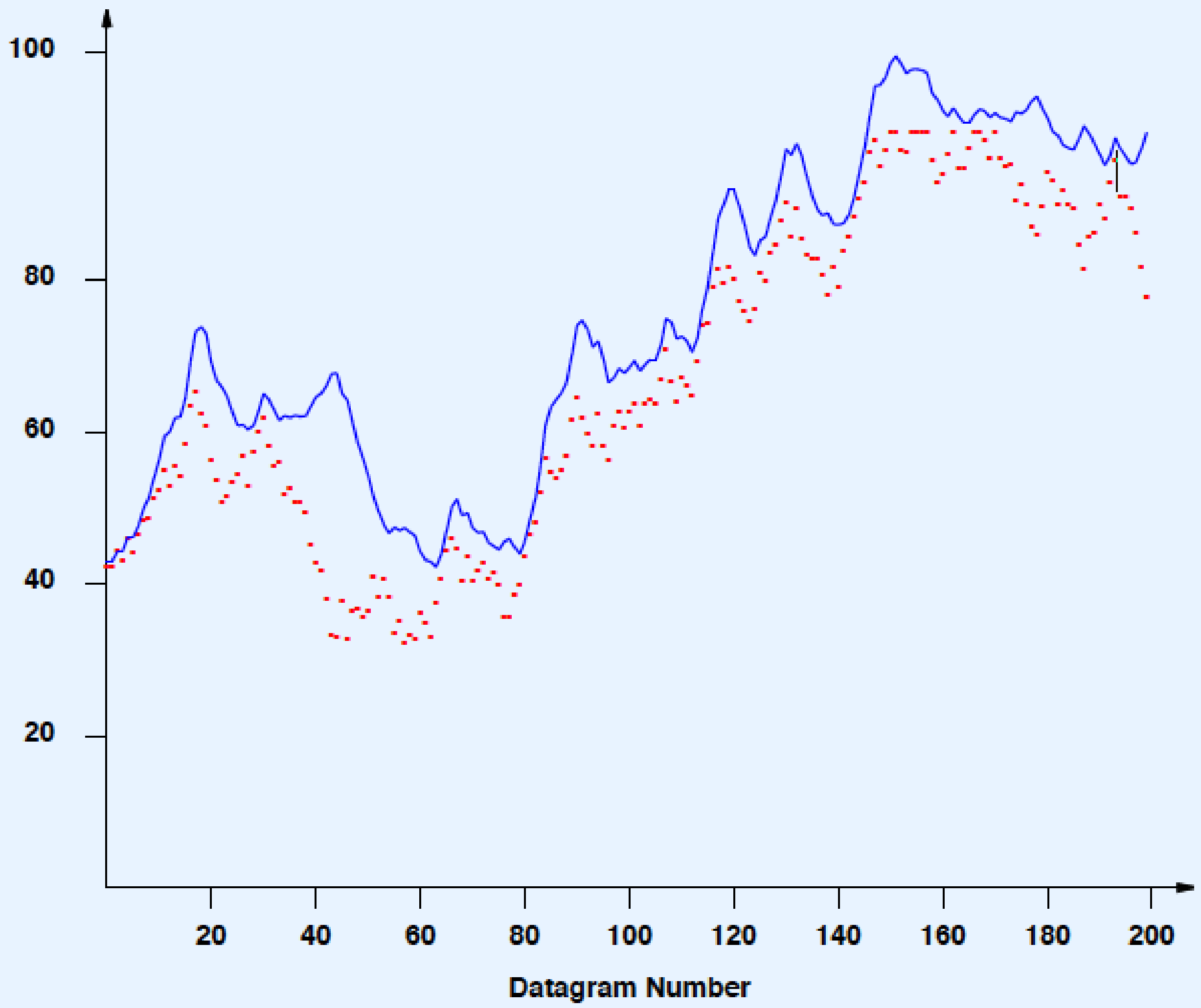
$$RTO = \mu * RTT + \phi * DEV$$

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

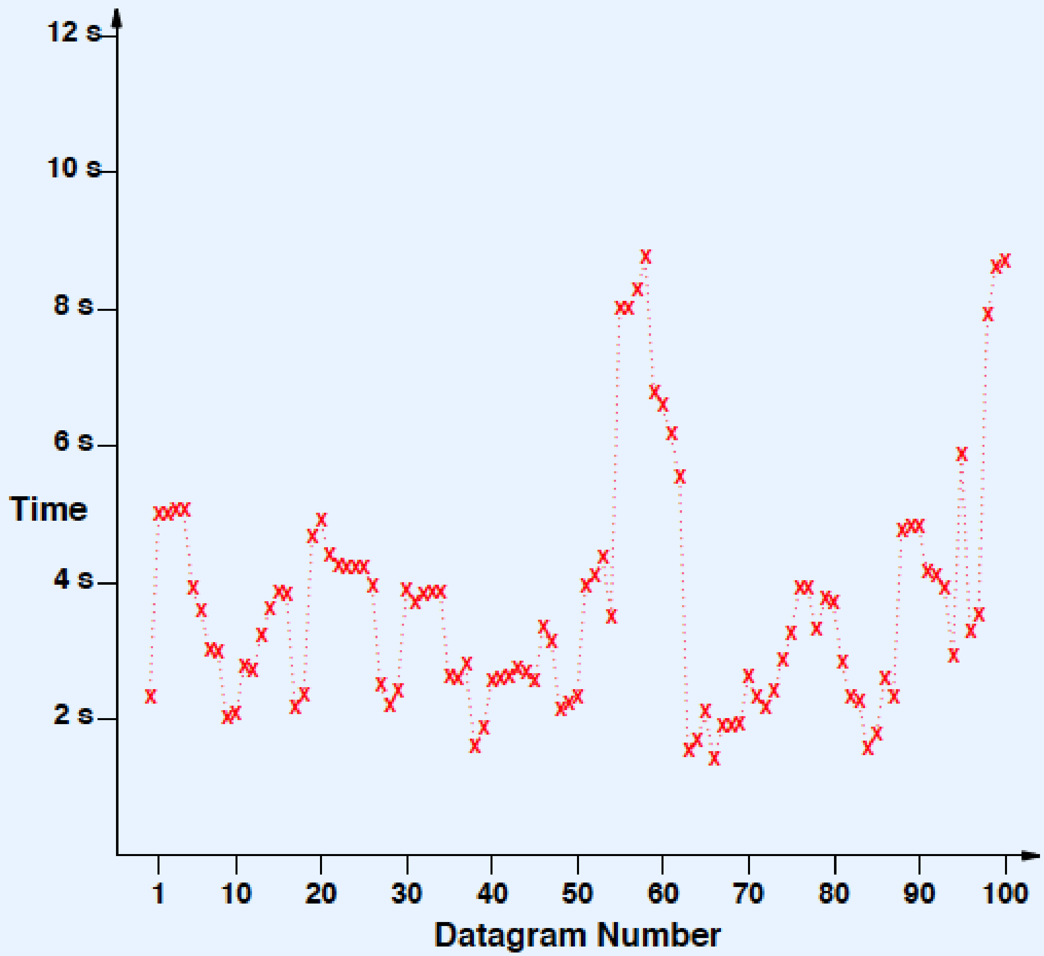
Computing Estimated Deviation

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

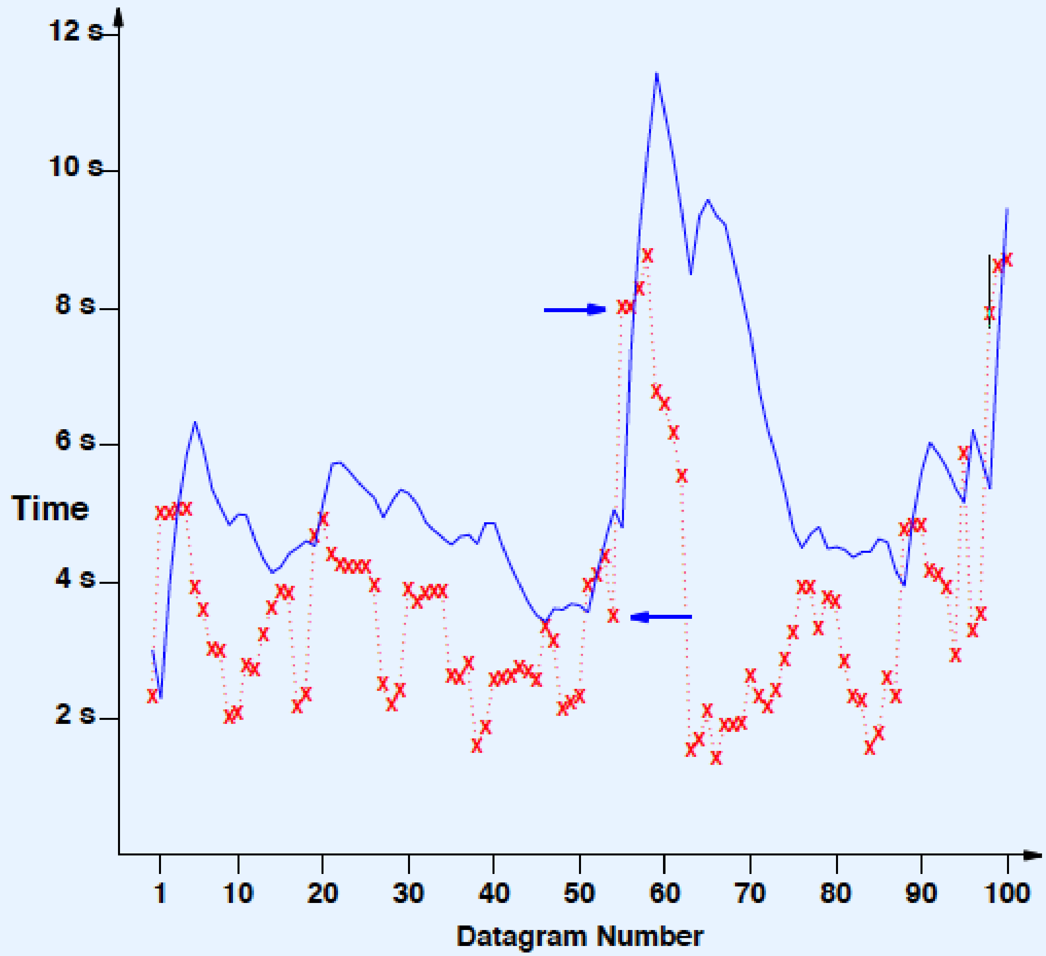
TCP Round-Trip Estimation



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



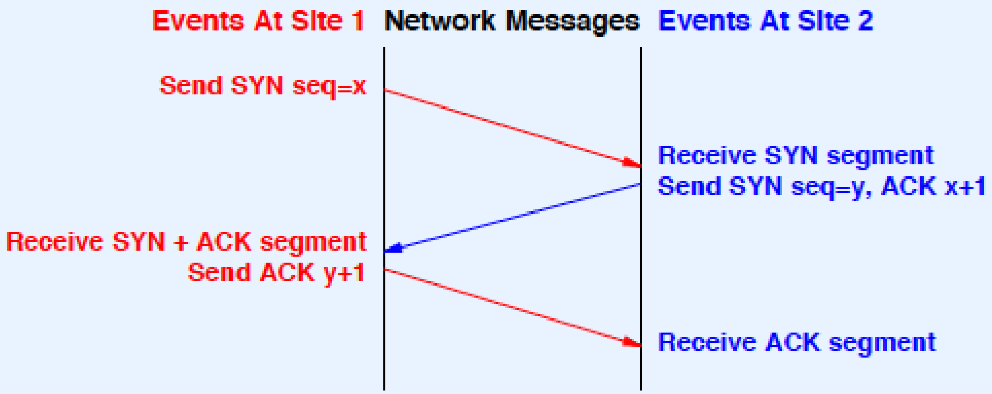
TCP Round-Trip Estimation For Sampled Internet Delays



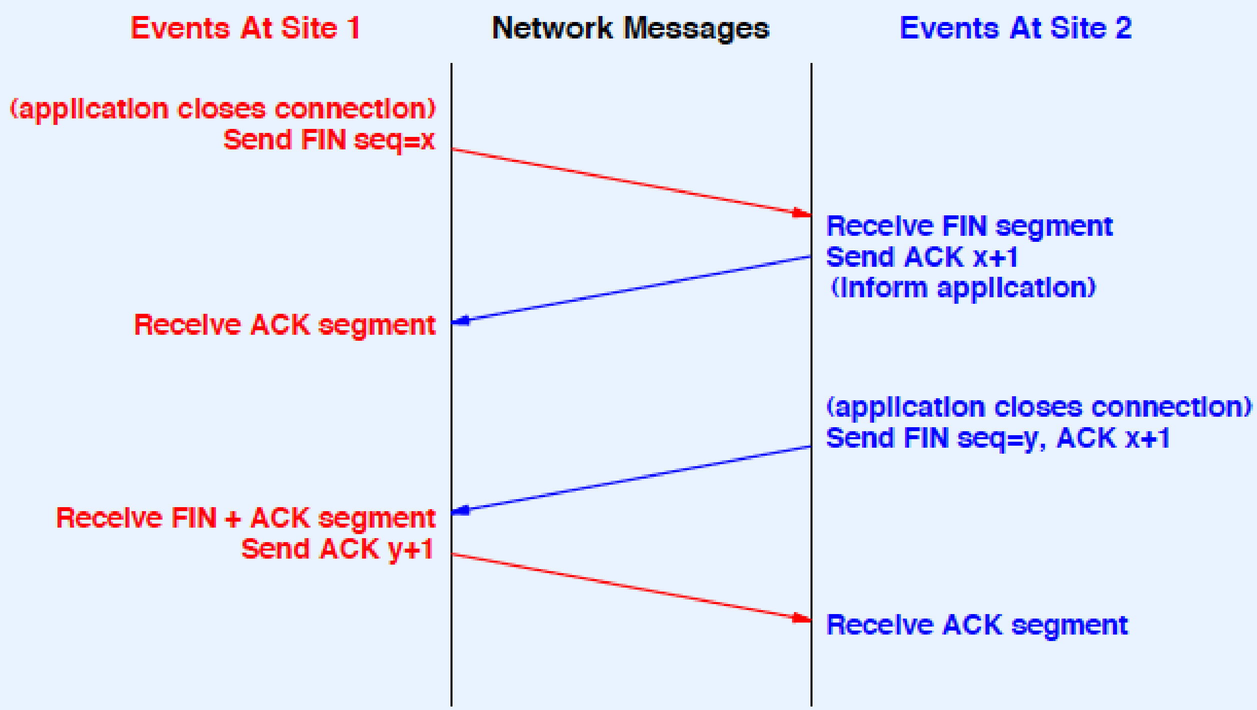
TCP Details

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

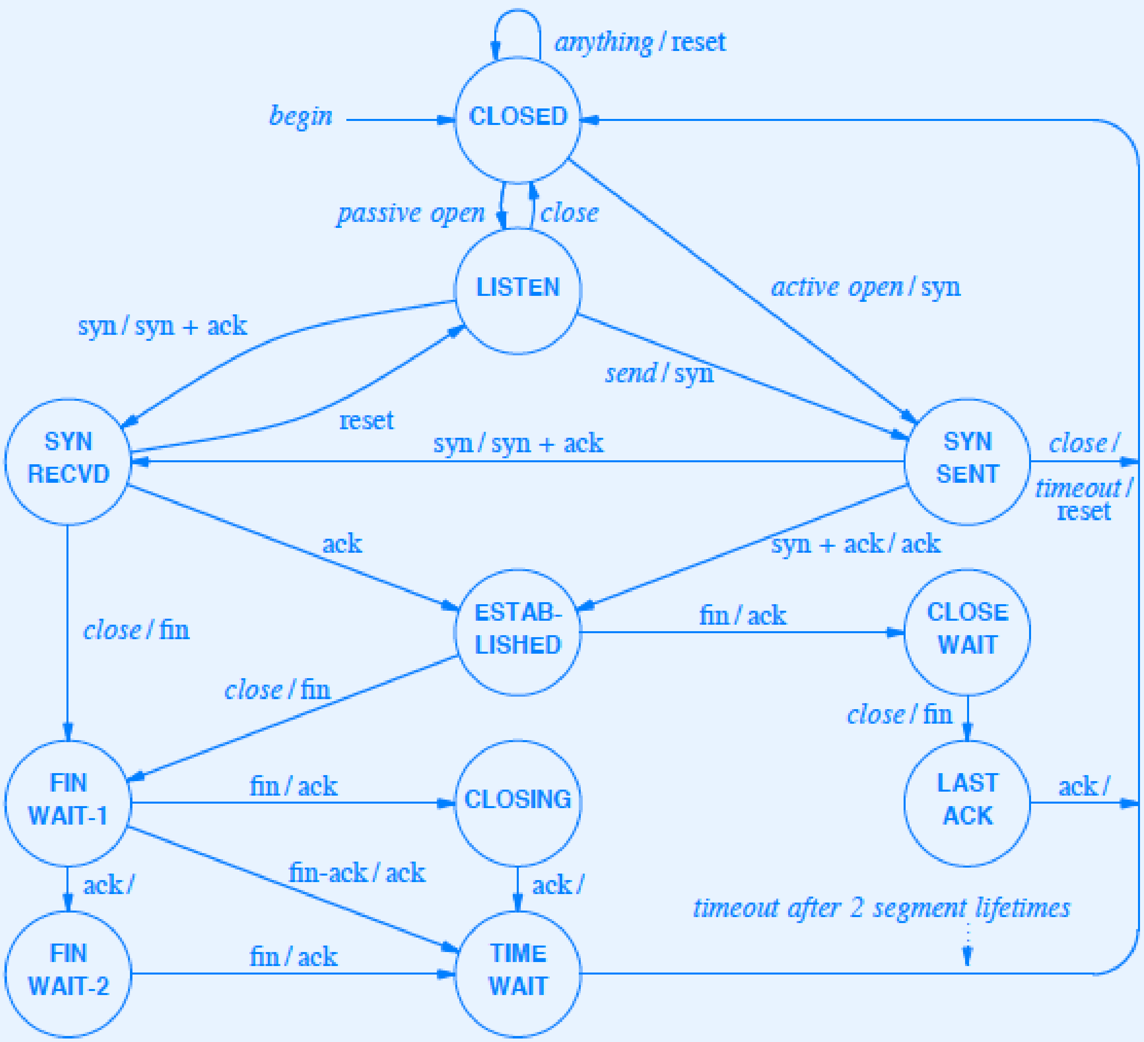
3-Way Handshake For Connection Startup



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

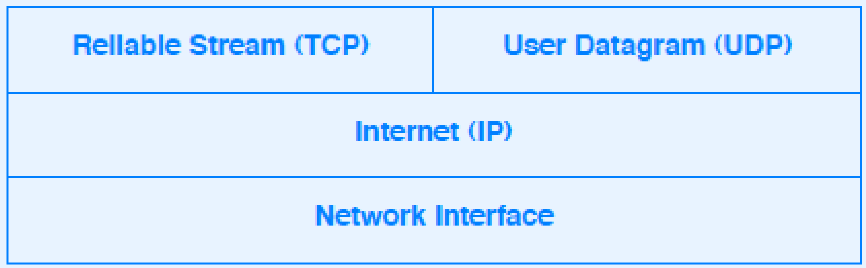
Fast Retransmit

- 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