Transmission Control Protocol

- Major transport service in the TCP/IP suite
- Reliable transfer
- Stream paradigm
- Full duplex connections
- Flow control
- Uses IP for datagram transmission

Transmission Control Protocol Details

- Allows sender to generate a stream of bytes in convenient chunks
- Divides stream into small segments for transmission
- Sends each segment in IP datagram
- Receiving TCP returns acknowledgement upon successful receipt of data
- Sender starts timer after segment sent, and retransmits unless positive acknowledgement arrives

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 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
- Heuristics can sometimes fail (e.g., round-trip delay changes quickly)

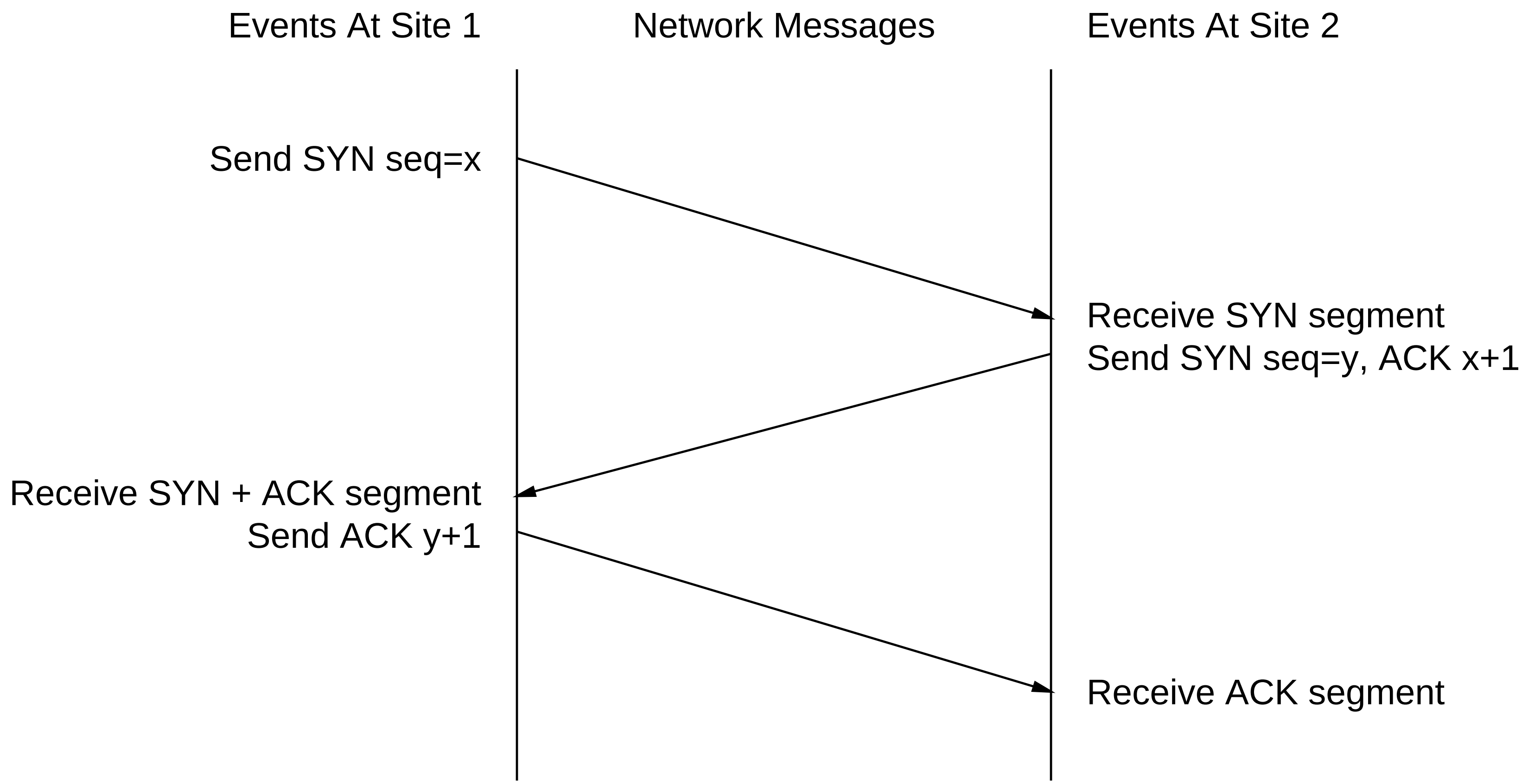
TCP Details

- Segment contains checksum for data being sent
- Receiver acknowledges highest byte received, not each specific segment
- Protocol port numbers used to distinguish among multiple application programs
- Receiver controls flow by telling sender size of currently available buffer
- Called window *advertisement*
- Each segment contains advertisement, including data segments

TCP Details
(continued)

- Receiver can send additional acknowledgments whenever buffer space becomes available
- 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



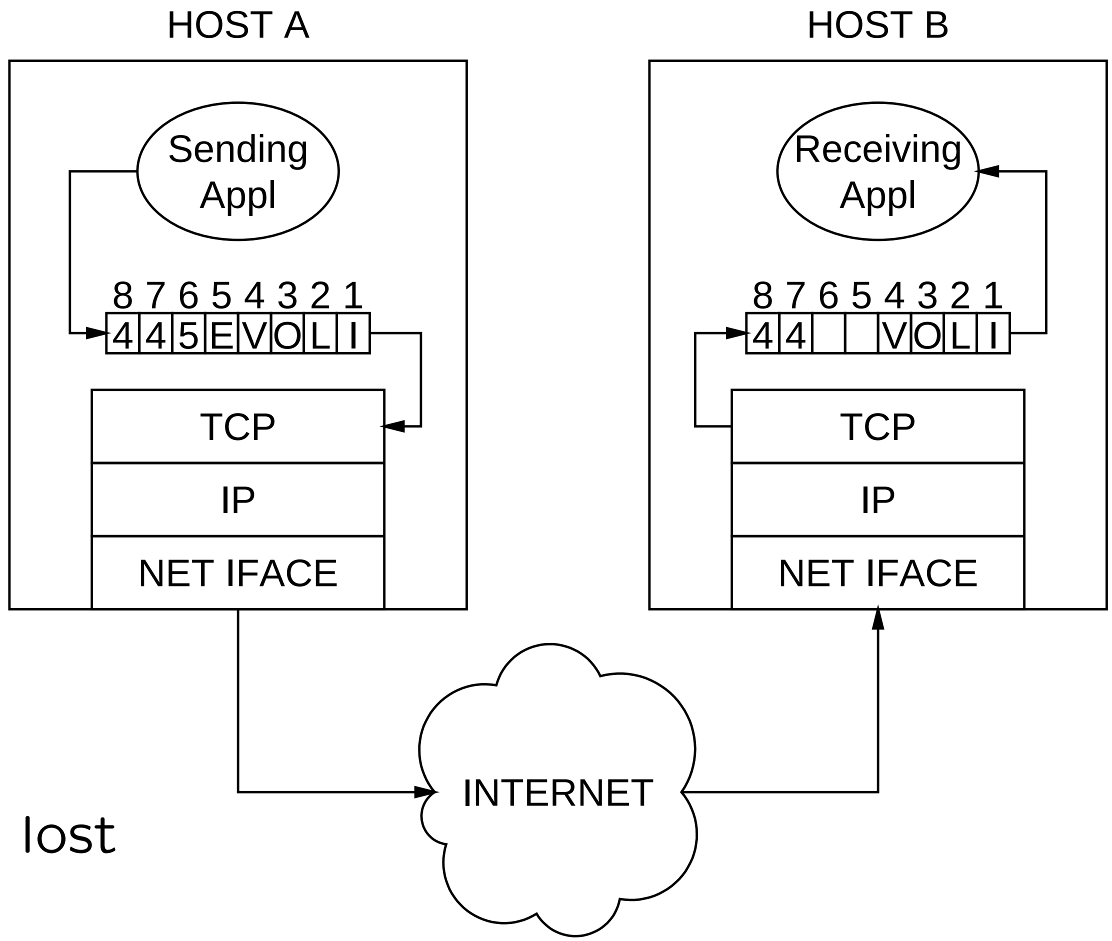
TCP Segment Format

0 16 31

<table>
<tr><td colspan="2">TCP SOURCE PORT</td><td colspan="2">TCP DESTINATION PORT</td></tr>
<tr><td colspan="4">SEQUENCE NUMBER</td></tr>
<tr><td colspan="4">ACK NUMBER</td></tr>
<tr><td>HLEN & RES</td><td>CODE BITS</td><td colspan="2">WINDOW</td></tr>
<tr><td colspan="2">CHECKSUM</td><td colspan="2">URGENT POINTER</td></tr>
<tr><td colspan="3">OPTIONS ...</td><td>padding</td></tr>
<tr><td colspan="4">... DATA ...</td></tr>
</table>

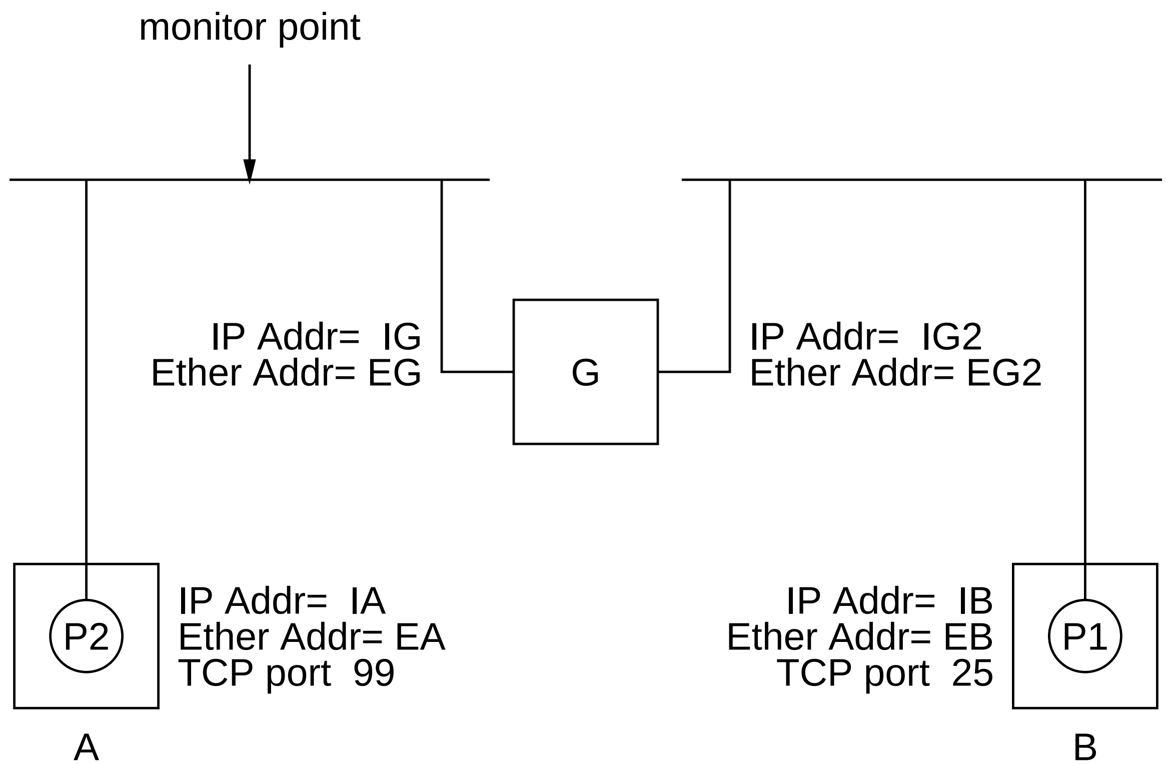
- Offset specifies header size (offset of data) in 32-bit words
- Code bits specify *urgent*, *ack*, *push*, *reset*, *syn*, or *fin*

TCP Acknowledgement Example



- Assume octets 5 & 6 lost
- Sender transmits octets 7 & 8
- Receiver acknowledges octets 1 4

Example Packet Trace For TCP Connection



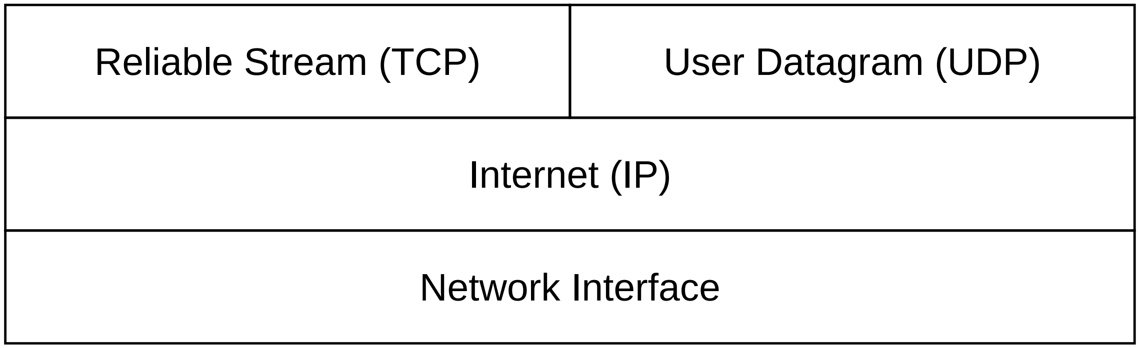
- Machines A, B, G boot
- P1 forms TCP connection to P2, sends one octet of data, and closes connection

Example Packet Trace (continued)

	Hardware Frame			Address Resolution Message				
	Src	Dst	Typ	Op	Snd IP	Snd E	Tar IP	Tar E
1	EA	*	ARP	REQ	IA	EA	IG	?
2	EG	EA	ARP	RSP	IG	EG	IA	EA

	Hardware Frame			IP Datagram			TCP Segment		
	Src	Dst	Typ	Src	Dst	Typ	Src	Dst	Type
3	EA	EG	IP	IA	IB	TCP	99	25	SYN
4	EG	EA	IP	IB	IA	TCP	25	99	SYN+ACK
5	EA	EG	IP	IA	IB	TCP	99	25	ACK
6	EA	EG	IP	IA	IB	TCP	99	25	DAT
7	EG	EA	IP	IB	IA	TCP	25	99	ACK
8	EA	EG	IP	IA	IB	TCP	99	25	FIN+ACK
9	EG	EA	IP	IB	IA	TCP	25	99	ACK
10	EG	EA	IP	IB	IA	TCP	25	99	FIN+ACK
11	EA	EG	IP	IA	IB	TCP	99	25	ACK

Conceptual Layering



Assignment Of Protocol Ports

- Need globally fixed ports for globally-known services
- Need dynamically allocated ports for other services
- Accommodate with two port types
  - Statically assigned ports
  - Dynamically assigned ports
- Note: servers use statically assigned ports; clients use dynamically assigned ports

Statically Assigned Ports

- Called “well-known”
- Used for services like e-mail
- Fixed by Internet Assigned Numbers Authority
- Use “small” values
- In UNIX, values less than 1000 reserved for privileged programs

Dynamically Assigned Ports

- Available for user applications
- Operating system chooses when application begins
- Programmer responsible for devising mechanism to inform other programs
- Use “large” values

Program Interface To Port Assignment

- Port numbers should not be encoded in programs as literal constants
- Most systems provide
  - Database of service names
  - Library routines that use the database to map names into protocol port numbers (e.g., getservbyname)
- Site can add local definitions to the database

Example Service Mapping Database (/etc/services in UNIX)

```
echo              7/tcp
echo              7/udp
ftp              21/tcp
telnet           23/tcp
smtp             25/tcp
time             37/tcp
time             37/udp
nameserver       53/tcp
nameserver       53/udp
foobar         2001/udp
```

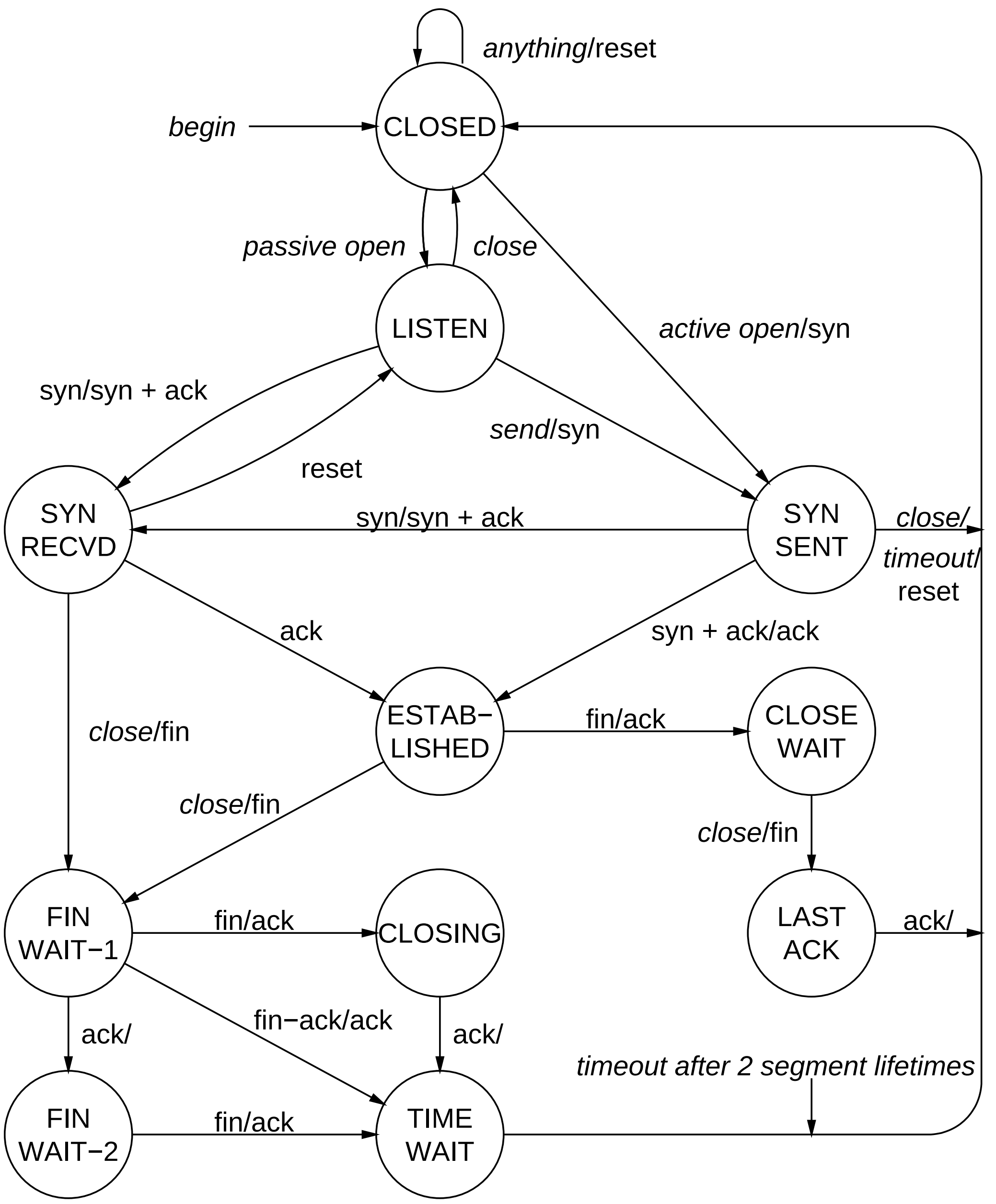
Example Service Mapping Database (continued)

- Program contains literal constants for name of service and name of protocol
- Program calls library routine to obtain port number
- Port mapping can be changed without recompiling program

TCP Formal Specification (Finite State Machine)

- TCP behavior specified with finite state machine
- At any instant, each side of TCP connection is in one state
- Think of the state machine as controlling response to input
- Arrival of a segment can cause a state transition
- A local operation can also cause a state transition (e.g., *close*)

TCP Finite State Machine



Example Transition: Opening A Connection

- Both sides create TCP endpoint (e.g., using socket calls)
- TCP software on both sides record that connection is initially in *CLOSED* state
- Server side issues passive open and waits in *LISTEN* state
- Client issues active open, sends SYN segment, and moves to *SYN SENT* state
- Server side receives SYN, sends SYN plus ACK, and moves to *SYN RECVD* state
- Client receives SYN plus ACK, sends ACK, and moves to *ESTABLISHED* state
- Server receives ACK and moves to *ESTABLISHED* state
- Now both sides agree that connection is open

Maximum Segment Size

- TCP endpoints use the MSS option to exchange the maximum segment that they are willing to receive
  - Improves efficiency
  - Is a function of the networks between the hosts
    - TCP tries to avoid sending segments that will have to be fragmented
      - Fragmentation decreases effeciency
      - Fragmentation decreases throughput
- Normal sizes are network MTU for local connections and 576 for non-local

Adaptive Retransmission

- The problem is knowing when to retransmit
- TCP keeps estimate of round-trip time for each connection
- Round-trip estimate computed from observing difference in times when segment transmitted, and time when ACK arrives
- Timeout for retransmission is function of round trip estimate

Difficulties With Adaptive Retransmission

- 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
- Load imposed by a single connection can congest gateways 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: $\alpha = 0.8, 0.2$
- 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
(Also called *Karn's Algorithm*)

- 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
- 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

Partridge/Karn Scheme
(continued)

- Resets retransmission timer to be function of round trip estimate when ACK arrives for nonretransmitted segment
- Works well for occasional packet loss
- Provides exponential backoff from completely saturated network
- Does not solve the problem of flow control or congestion

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 size in a *threshold* variable (field *ssthresh* in the code)
- 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)

J/K 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 win/MSS)
or
     increase  = (MSS*MSS)/congestion win
```

Changes In Delay

- Original smoothing scheme tracks the mean but not changes
- To track changes, compute DIFF = SAMPLE - RTT
  RTT = RTT + δ * DIFF
  DEV = DEV + δ (|DIFF| - DEV)
- 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 = μ*RTT + ϕ*DEV
- Typically, $\mu = 1$ and $\phi = 4$

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