TCP, Lecture 4

Internternet Protocols

CSC / ECE 573

Fall, 2005

N. C. State University

Today's Lecture

I. Congestion Control: Fast Retransmit and Recovery

II. Silly Windows

III. Urgent Data

IV. (Some) TCP Options

V. Router Queue Management

FAST RETRANSMIT AND RECOVERY

Fast Retransmit and Recovery (RFC 2581)

- When an out-of-order segment arrives at the receiver…
  - receiver will generate an **ACK** with the same sequence number as the previous ACK; called a *duplicate **ACK***
  - indicates that some data is getting through to receiver
- Receipt of 3 duplicate **ACKs** in a row for segment *j* is a strong indication that segment *j+1* was lost
  - immediately retransmit without waiting for timer to expire: *fast retransmit*
  - then enter congestion avoidance phase directly (i.e., bypass slow start): *fast recovery*

Example

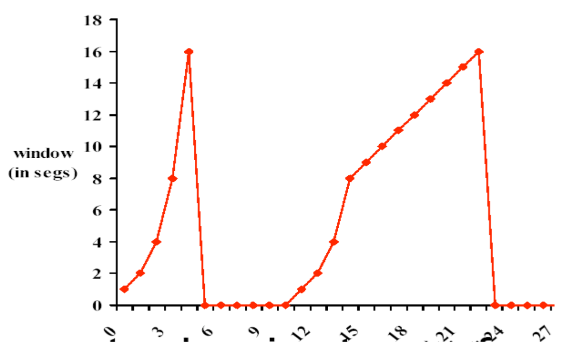
A sends S1	B acks S1
A sends S2	(lost in transmission)
A sends S3	B acks S1
A sends S4	B acks S1
A sends S5	B acks S1

3 duplicate ACKs for segment S1, probably means segment S2 was lost

Fast Retransmission and Recovery

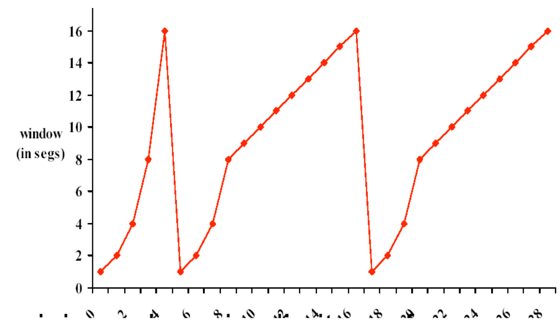
- Sender actions (after receiving 3rd duplicate **ACK**)
  - retransmit: retransmit segment *j*+1 without waiting for timeout
  - recover:
    - **ssthresh** ← MAX(2, ½ * **cwnd**)
    - **cwnd** ← **ssthresh** + 3
- Each time another duplicate **ACK** arrives…
  - **cwnd** ← **cwnd** + 1 */* haven't started congestion avoidance */*
  - transmit another packet (if allowed)
- When **ACK** for a retransmitted segment arrives…
  - **cwnd** ← **ssthresh** */* now in congestion avoidance */*

Behavior: Slow Start+Congestion Avoidance



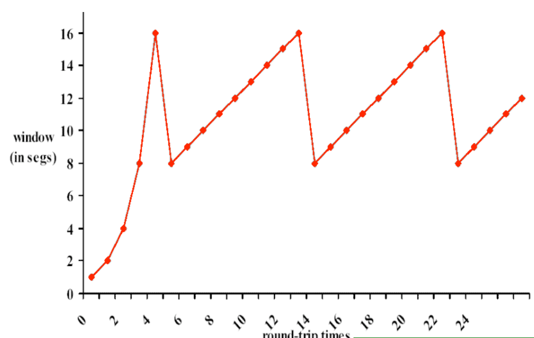
Source: Jay Lepreau, U. Utah

Behavior: SS + CA + Fast Retransmit



Source: Jay Lepreau, U. Utah

Behavior: SS + CA + FRXMIT + Fast Recovery



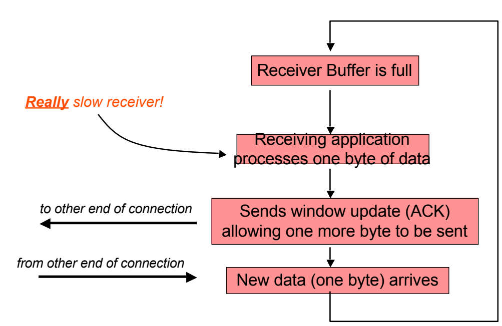
Source: Jay Lepreau, U. Utah

"SILLY" WINDOWS

"Silly Window" Syndrome (RFC 813)

- A serious problem in sliding window operation
- Causes
  1. sending application program creates data slowly
  2. receiving application program consumes data slowly
- In either case, data may be sent in small segments
  - inefficient use of bandwidth
  - increased processing by TCP

"Silly Windows" Caused by Receiver



"Silly Window" Solution

- *Clark's solution*: do not send window advertisements for 1 byte
- Instead, advertise a window size of zero and wait until
  1. there is space for a maximum sized segment of data, *or*
  2. the receive buffer is half-empty
- Then advertise this new Window Size

Combining Solutions

- Nagle's algorithm: sender accumulates data until "enough" data to send
- Clark's solution: receiver consumes data until "enough" space available to advertise
- These solutions are complementary and can be used together

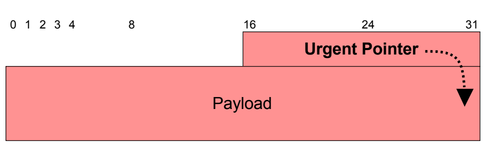
URGENT AND PUSH FLAGS

Urgent Data

- TCP does not provide a separate "control" channel for applications
- Examples of control
  - ftp: "stop sending data"
  - telnet: interrupt running process, or suspend the telnet session
- Choices
  - use a second (companion) TCP connection for control, *or…*
  - insert control into data channel and mark as urgent

Urgent Data (cont'd)

- Marking urgency: set the **URG Flag** and set **Urgent Pointer** to indicate location of the control information
- TCP notifies application "urgent data" has been received
  - processing is application specific



The PSH Flag

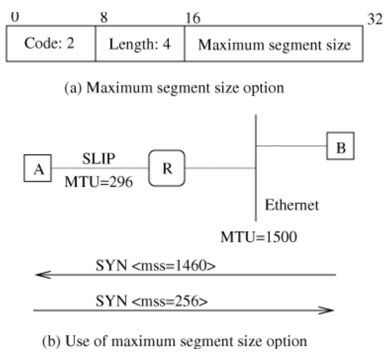
- Purposes
  - Sender: forces TCP to send a segment without waiting for further data to be generated
  - Receiver: forces TCP to notify the application that data is waiting to be processed
- Example: after each command typed, during an interactive application
- Prevents TCP buffering (for efficiency) from adding undesirable delivery latency

(SOME) TCP OPTIONS

#1. MSS Option (cont'd)

- MSS = Maximum Segment Size
  - defines the maximum segment size the receiver is willing to accept
  - MSS must be ≤ receiver interface MTU - 40 bytes
- Declared during connection establishment phase (i.e., in **`SYN`** segments)
  - cannot be specified or changed during data transfer

#1. MSS Option (cont'd)

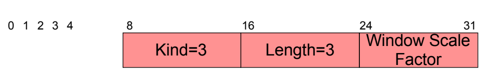


#2. Window Scale Option (RFC 1323)

- Reminder: optimal **`Window Size`** = RTT * receiver bandwidth
- For RTT = 100ms, and bandwidth > 640 KB/s, optimal value is larger than maximum **`Window Size`** (64KB)
- Solution: negotiate (during connection establishment only) a scale factor that increases possible window sizes
  - may have different scale factors in the two directions

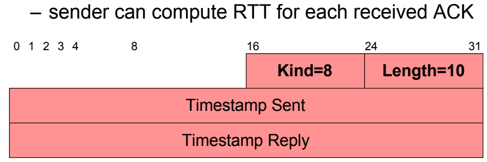
#2. Window Scale Option (cont'd)

- Scale factor is the exponent of a power-of-two increase in the window size
  - "effective" window size = **`Window Size`** × $2^{\texttt{window-scale-factor}}$
- Maximum value = 14
  - max effective window size = $(2^{16}-1)*2^{14}$ ($\approx 2^{30}$)
  - At RTT=100ms, max receiver bandwidth = 10 GB/s



#3. Timestamp Option

- Without timestamps, RTT (in many implementations) is calculated once every window
  - OK for small (e.g., < 8 segment) windows
  - but larger windows require better RTT calculations
- Sender puts timestamp option in segment; option is "reflected" by receiver in the acknowledgment
  - sender can compute RTT for each received ACK



#4. Selective Acknowledgments Option (RFC 2018)

- Selective Acknowledgments: indicate specifically what non-contiguous blocks of data have been received
- Option format:

0 1 2 3 4	8	16	24	31
		Type	Length	
Left Edge of 1st Received Block				
Right Edge of 1st Received Block				
Left Edge of 2nd Received Block				
Right Edge of 2nd Received Block				
etc.				

#4. Selective Acknowledgments (cont'd)

- Receiver notifies sender that non-contiguous blocks of data have been received
  - at most 4 blocks can be specified
  - should be included in all ACK=1 segments that do not acknowledge the highest contiguous sequence number received
  - report the most recent non-contiguous blocks
- Sender will not retransmit selectively-acknowledged segments

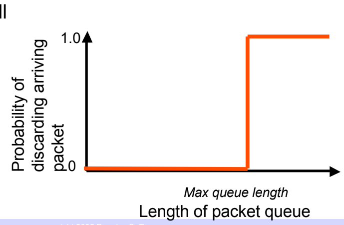
#4. Selective Acknowledgments (cont'd)

- Can result in better throughput when losses are common
- Requires negotiating "SACK-permission" during connection establishment

ROUTER QUEUE MANAGEMENT

Packet Dropping Policies at Routers

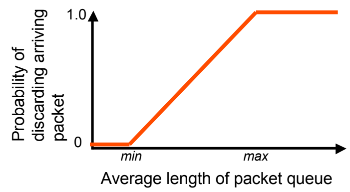
- Incoming packets at a router, after the forwarding decision, are queued for output
- During congestion, something has to give!
  - "Drop Tail" policy: drop an incoming packet if the queue is already full



RED (RFC 2309)

- Another way: drop the arriving packet randomly, with probability derived from the queue length: *RED (random early discard) policy*
  - queue length used is an exp. weighted moving average
- Parameters
  - *min* and *max* thresholds per output queue

RED (cont'd)



RED Evaluation

- RED considerably more complex than drop-tail
- Claim: RED results in shorter average queue lengths (thus, lower latency)
- Drop-tail synchronizes losses across flows
  - i.e., they all congest at same time, all back off at same time, then all congest at same time, …
- Claim: drop-tail unfair to *bursty* traffic flows
- Claim: RED gives significantly better throughput

Summary

- Fast Retransmit and Recovery provide improved "steady state" behavior
- "Silly" Windows leads to inefficient data transfer
- Ideas about congestion control improvements are never-ending ☺
- TCP options provide useful extensions; MSS is universally used
- Active queue management has been widely promoted as providing better throughput and fairness

Next Lecture

- The Sockets Network Programming API