TCP, Lecture 2

Internet Protocols

CSC / ECE 573

Fall, 2005

N. C. State University

Today's Lecture

I. The TCP "State Machine" Diagram

II. TCP Timers

III. Interactive (i.e., low rate) data transfer

IV. Flow control (window size advertisements)

Project Part I posted

HW3 will be posted today

Project Choices

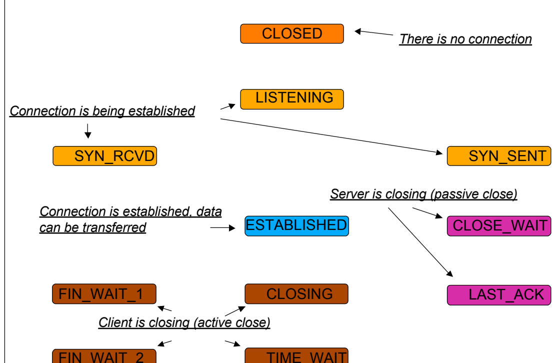
- Implement TCP at the application layer
- Implement the routing protocol RIPv2 at the the application layer.
- Implement a new protocol from either Infocom or SIGCOMM
- Implement interplanetary TCP
- Implement a DNS server
- Implement IPSec Authentication Header
- Implement basic VoIP
- Implement another Internet protocol

THE TCP STATE MACHINE

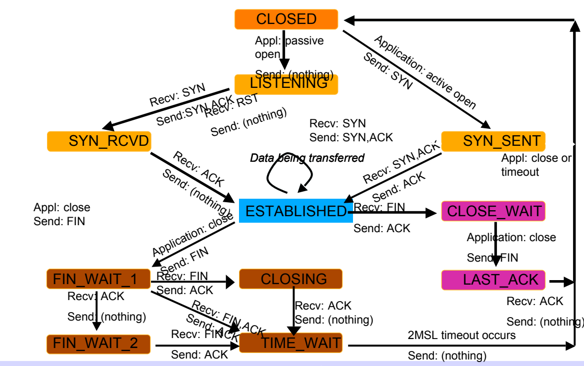
Interpreting State Machine Diagrams

- Each shows the TCP state machine for one endpoint of a connection
  - other endpoint may be in a different state
- The transitions between states (arcs) are labeled with…
  - the input event that causes the transition
  - the output signal or message that will be sent to the other endpoint

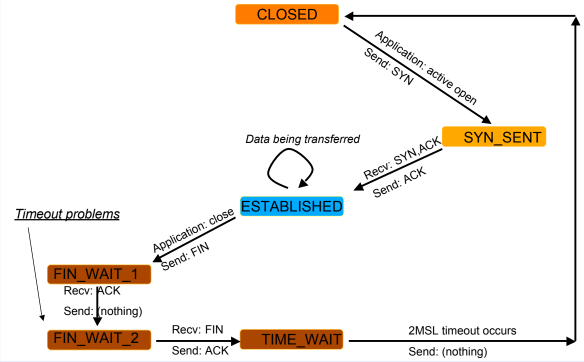
States of a TCP Connection



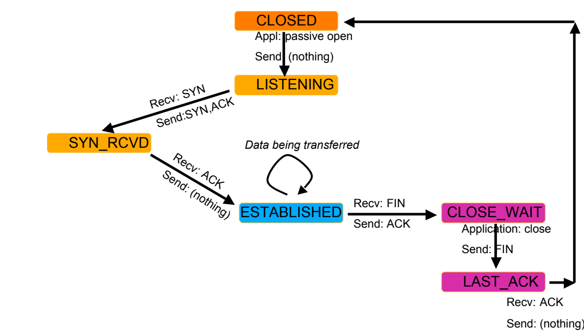
Complete State Diagram



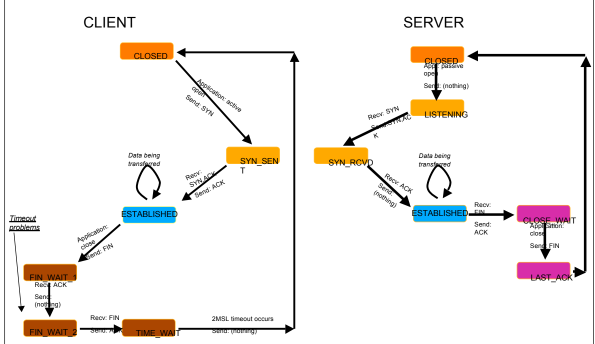
Normal Client Sequence



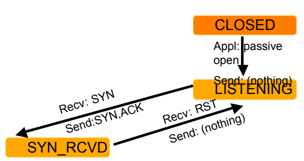
Normal Server Sequence



All Together, Now

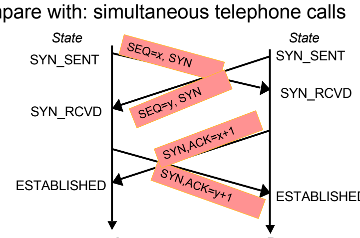


Server States When Client Resets Connection Before Establishment Complete



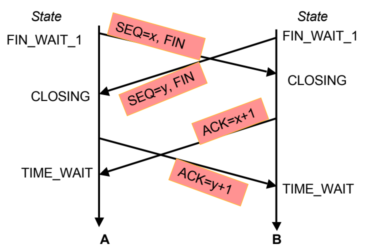
Simultaneous Active Opens

- A connects with B, B connects with A at same time (pass each other in the network)
  - only one connection will be established!
  - compare with: simultaneous telephone calls



Simultaneous Close

- A sends FIN to B, B sends FIN to A at same time (pass each other in the network)
  – still 4 messages, but less time



TCP TIMERS

Time-Wait (2 MSL) Timer

- After agreeing with the other side to close a connection, TCP enters the TIME_WAIT state
  – starts a timer that runs for twice the maximum packet *lifetime*
  – the connection is closed after the timer expires, allowing port number to be reused
- Ensures all packets (and their ACKs) have been delivered or discarded before "reusing" the connection
  – helps prevent the overlapping connections problem

Keepalive Timer

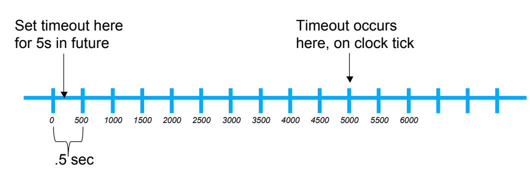
- It is possible for TCP connections to be idle for a l-o-n-g time
  – Ex.: client opens a connection to a server, and then makes a request once every 30 minutes
  – Ex.: client opens a connection to a server, but never makes a request
- Connections never closed unless explicit termination by the application or other endpoint?
- Problem: consumes resources (memory) on the server

Keepalive Timer (cont'd)

- *Keepalive* timer maintained by some implementations
  – not part of TCP standard, somewhat controversial
  – example: timer interval = 2 hours
  – the timer is restarted every time a segment is received
- When timer expires, check if other side is still connected
  – send "probe" packet 10 times at 75 second intervals
  – if there is no response, the connection is terminated
  – (details of probe packet omitted)

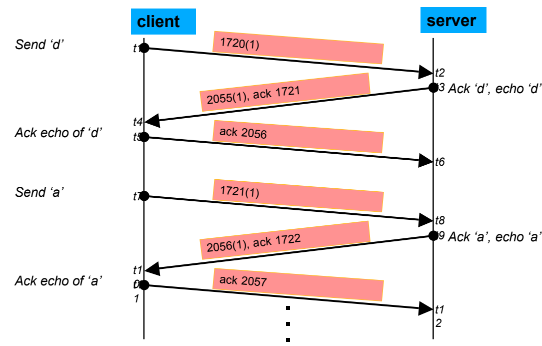
Timer Implementation Details

- To reduce overhead, timeout conditions are often evaluated on half-second (500 ms) boundaries, or clock *ticks*
  – i.e., a timeout scheduled for x seconds in the future will be processed at $\lfloor x / .5 \rfloor^{th}$ tick in the future, which is in the interval $[x-.5, x]$



INTERACTIVE DATA TRANSFER

Character-by-Character Transfer (Telnet)



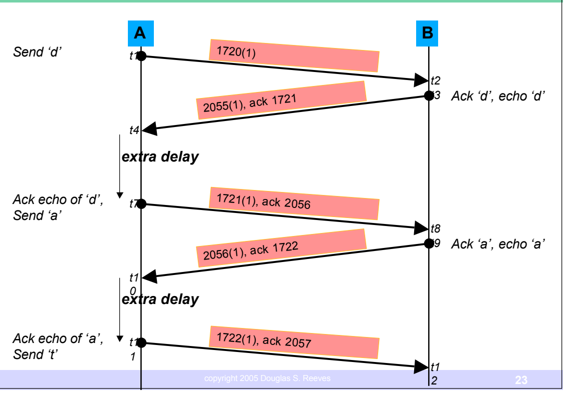
Performance?

- Each data packet has 1 character
  - 40 bytes of overhead (IP+TCP), 1 byte of payload!
- Some TCP segments are just ACKs
  - 40 bytes of overhead, no payload!
- Improvements possible? Does it matter?

Improvement #1: Delayed ACKs

- Purpose: by waiting, there may be more data to send
  - i.e., ACK can be “piggybacked” with data
- Maximum amount of time to wait is implementation dependent
  - Shorter better? Longer better?
  - (ex.: Linux= min of 40ms, max of 200ms)
- Good: reduces overhead
  - any drawbacks???

Delayed ACK Example



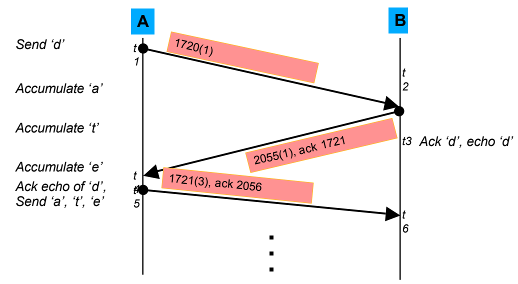
Improvement #2: “Nagle’s Algorithm” (RFC 896)

- Idea: “accumulate” data before sending a data segment
- When application generates data slowly, send the first byte and buffer the rest until…
  1. a maximum sized segment is filled, *or*
  2. an ACK is received, *or*
  3. half the current window size is filled

Nagle's Algorithm (cont'd)

- Benefits
  - a *self-clocking* algorithm (i.e., no timers needed)
  - useful for paths with long round-trip times
- Effect on throughput …
  - for low data rate applications?
  - for high data rate applications?
- Any drawbacks?

Example

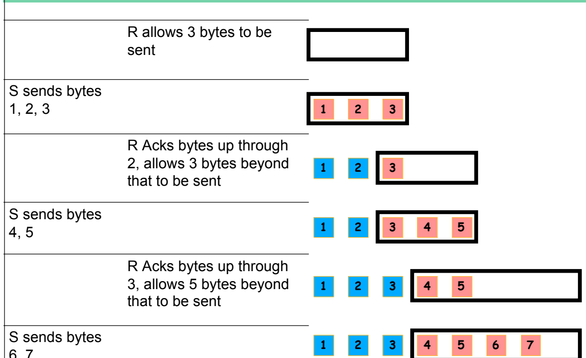


FLOW CONTROL AND OPTIMAL WINDOW SIZES

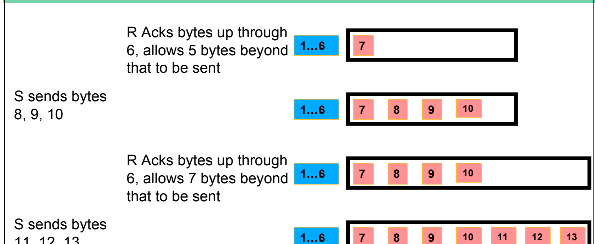
Flow Control in TCP

- *Flow control*: making sure the sender doesn't overrun the receiver's buffer
  - buffer contains data accepted from TCP, not yet processed by application
- Each ACK from receiver carries a *window advertisement* (Window Size)
  - # of additional bytes the sender is authorized to send before it must wait for an acknowledgment

Sliding Window, Again



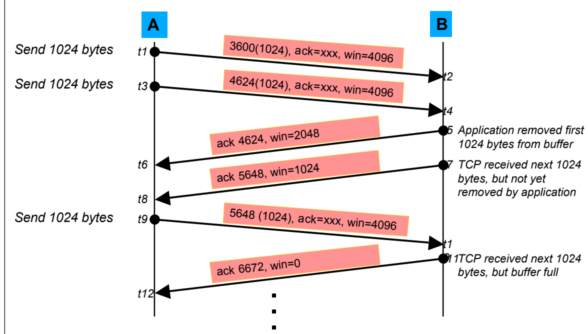
Sliding Window, Again (cont'd)



Rules for Sliding Window

- The "left edge" is shifted to right by acknowledgments
  - if "right edge" doesn't move, means window gets smaller
- The "right edge" is shifted to the right by window advertisements
  - if "left edge" doesn't move, means window gets larger
- Should never see…
  - "left edge" moving to left (why not??)
  - "right edge" moving to left (why not??)

Window Advertisements Example



How Big Should the Window Be?

- Example
  - the receiver can process data at (i.e., it's "bandwidth" is) a maximum rate of **100,000 bits/s**
  - the round-trip time from S to R and back is .4 s
  - very high-bandwidth (1 Gb/s or greater) network
  - ideal solution: sender sends at exactly the rate the receiver can receive

How Big Should the Window Be? (cont'd)

- Too **small**: 10,000 bit window (1250 bytes)
  - S sends 10,000 bits, waits .4s, sends 10,000 bits, …
  - rate: 10,000 bits/.4s = **25,000 bits/s**
- Too **large**: 100,000 bit window (12,500 bytes)
  - S sends 100,000 bits, waits .4s, sends 100,000 bits, …
  - rate: 100,000 bits/.4s = **250,000 bits/s**

How Big Should the Window Be? (cont'd)

- **Just right**: 40,000 bit window (5,000 bytes)
  - S sends 40,000 bits, waits .4s, sends 40,000 bits, …
  - rate: 40,000 bits/.4s = **100,000 bits/s**!

How Big Should the Window Be? (cont'd)

- The optimum window size = receiver bandwidth * round-trip time
  - this is called the *bandwidth-delay product*
  - e.g., .4*100,000=40,000 bits
- Challenges
  - how determine the maximum receiver rate?
  - how determine the round-trip time?
  - what if either or both changes frequently?

Other Examples

Receiver BW	RTT	Window Size
12.5 MB/s (100 Mb/s)	500 ms (.5 s)	6.25 MB (50 Mb)
1000 KB (8000 Kb)	5 ms (.005 s)	5 KB (40 Kb)

Summary

1. The state diagram captures all the behavior of connection state
2. The 2MSL timer and long sequence numbers help avoid confusion between independent connections
3. Delayed ACKs and Nagle's algorithm improve the efficiency of low-data-rate, high-RTT transfers

Summary (cont'd)

4. The receiver limits the sending rate through window advertisements
   - the sliding window size can change over time
5. the optimal window size is based on the network delay-receiver bandwidth product

Next Lecture

- TCP, lecture 3