Introduction to Computer Networks

COSC 4377

Lecture 8

Spring 2012

February 13, 2012

Announcements

- HW4 due this week
- Start working on HW5
- In-class student presentations
- TA office hours this week
  - TR 1030a – 100p

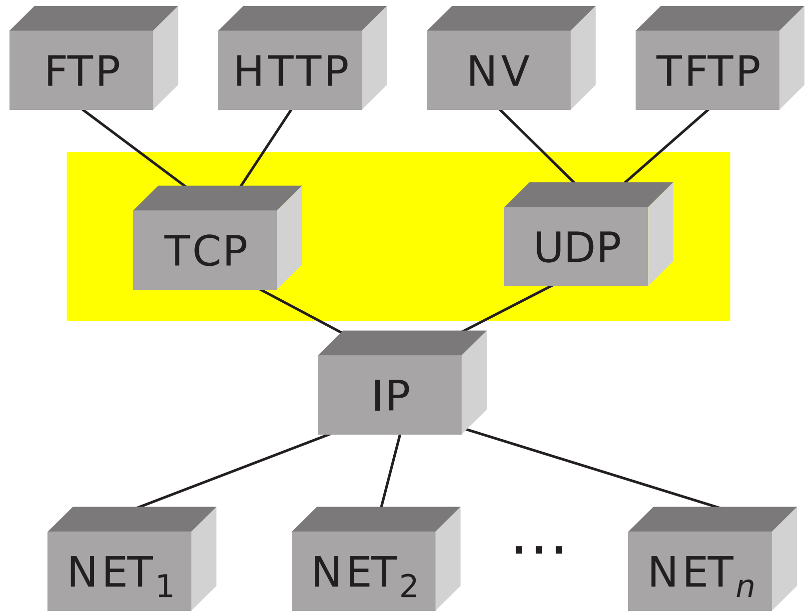
Today's Topics

- HW4 discussions
- Transport Protocols
  - Flow Control
  - Congestion Control

HW4

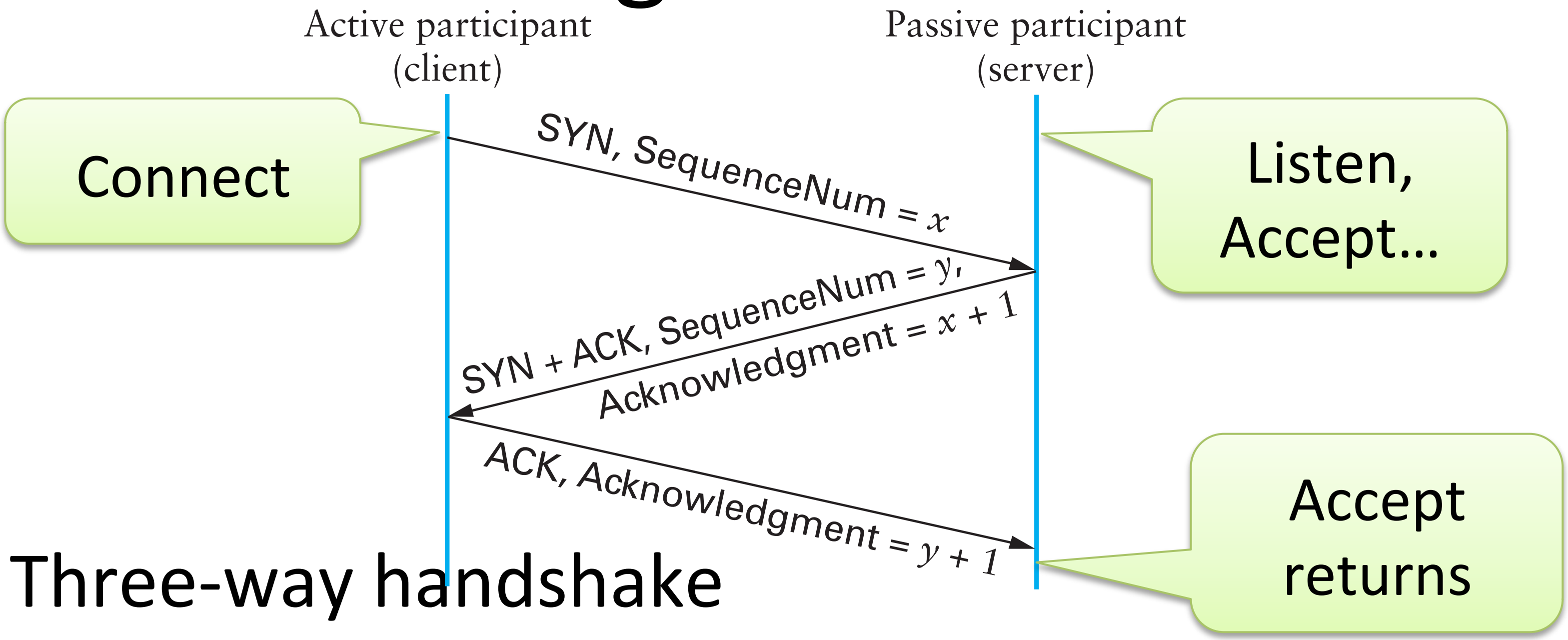
- Multiple clients connect to a single server
  - Limit the level of concurrency
- Keep track of unique IP and clients
- Testing easy if you have a way to create "slow" clients
  - Can use --limit-rate flag in wget
- Basic HTTP server code required

Transport Layer



- Transport protocols sit on top of network layer and provide
  - Application-level multiplexing ("ports")
  - Error detection, reliability, etc.

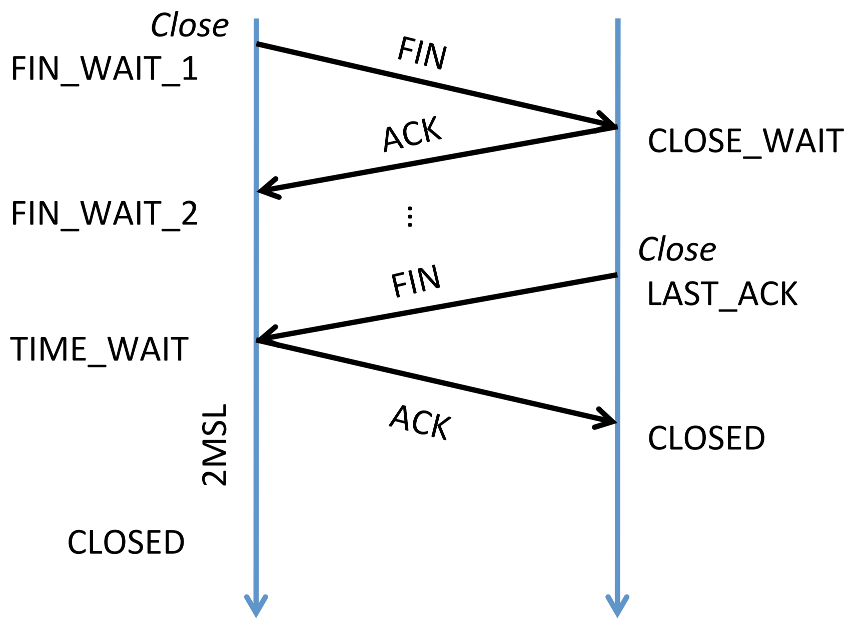
Establishing a Connection



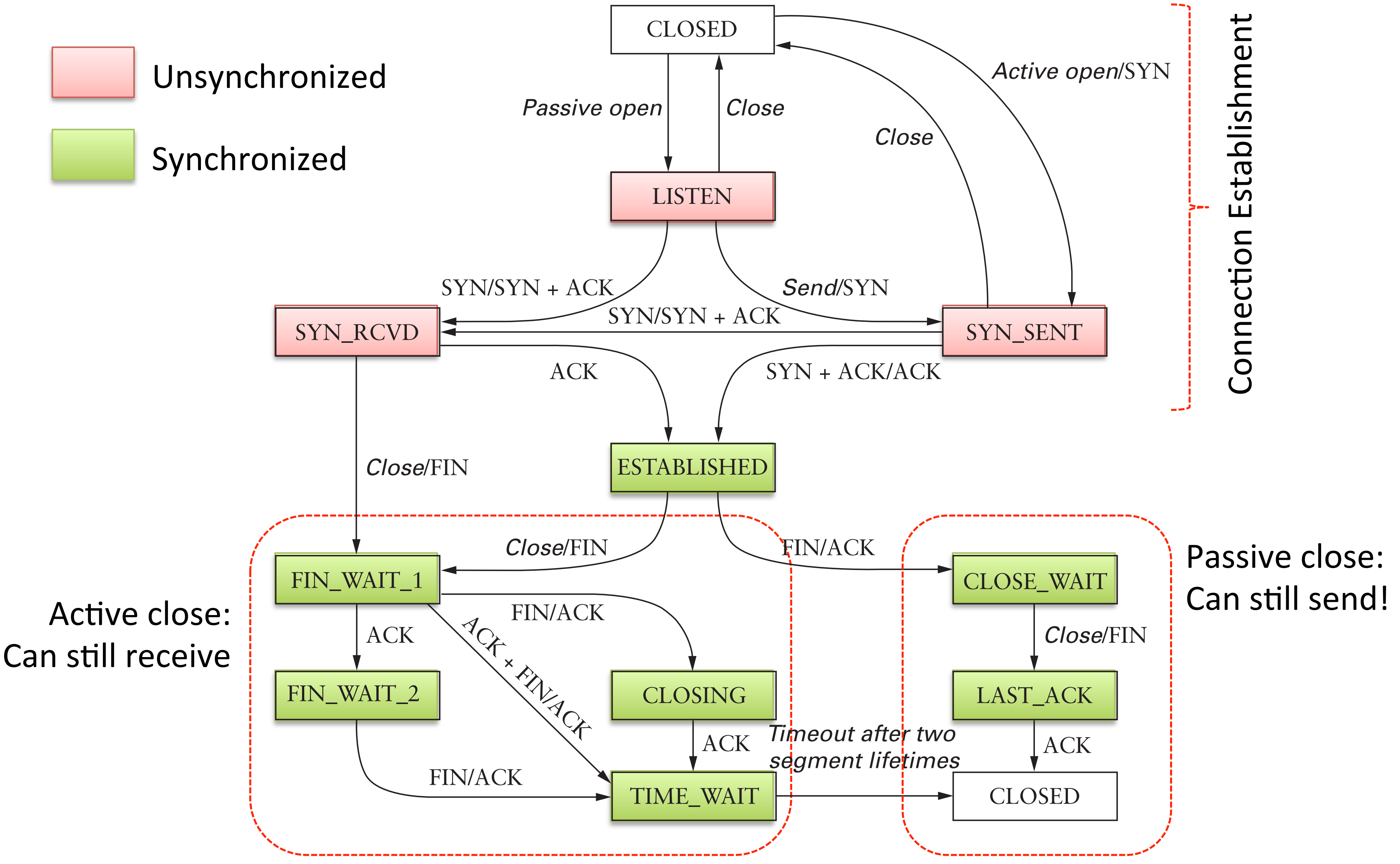
- Three-way handshake
  - Two sides agree on respective initial sequence nums
- If no one is listening on port: server sends RST
- If server is overloaded: ignore SYN
- If no SYN-ACK: retry, timeout

Connection Termination

- FIN bit says no more data to send
  - Caused by close or shutdown
  - Both sides must send FIN to close a connection
- Typical close



Summary of TCP States



EWMA

- Estimate RTT
- RTT(t) = $\alpha \times$ RTT(t-1) + (1-α) $\times$ newEst

$\alpha = 0.8$

Time	RTT	newEst
0	-	10
1	8.0	12
2	6.4+2.4=8.6	10
3	6.9+2=8.9	

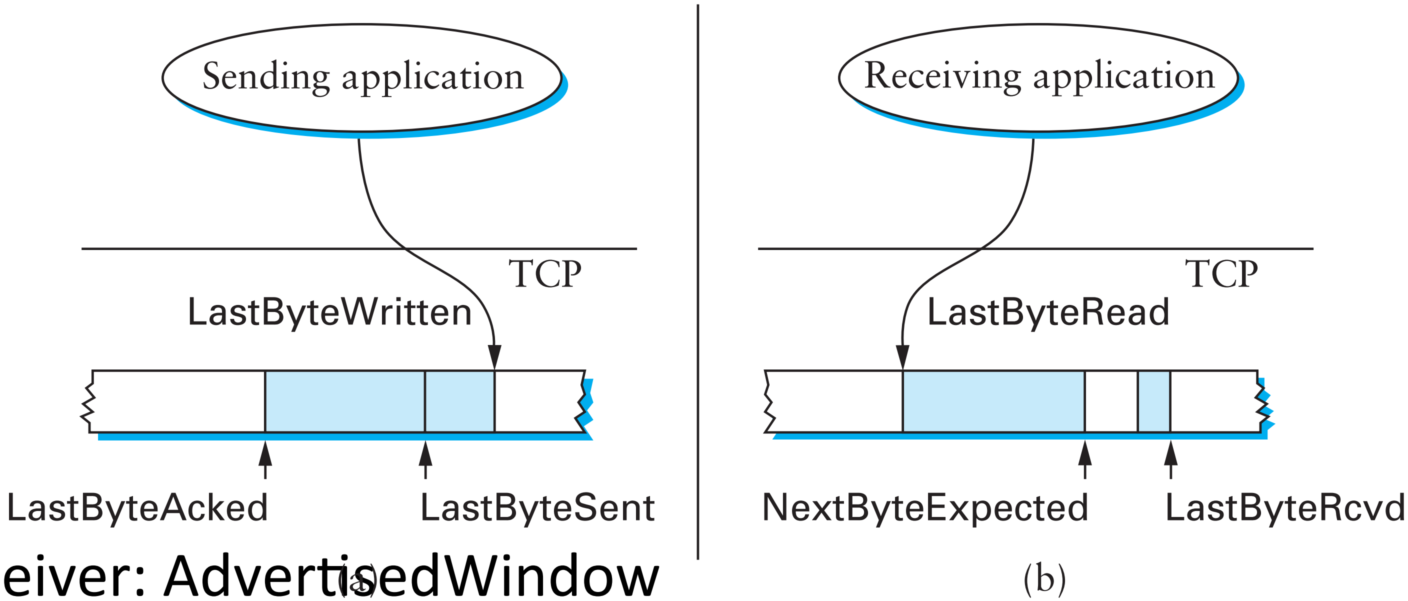
First Goal

- We should not send more data than the receiver can take: *flow control*
- Data is sent in MSS-sized segments
  - Chosen to avoid fragmentation
- Sender can delay sends to get larger segments
- When to send data?
- How much data to send?

Flow Control

- Part of TCP specification (even before 1988)
- Goal: not send more data than the receiver can handle
- Sliding window protocol
- Receiver uses window header field to tell sender how much space it has

Flow Control

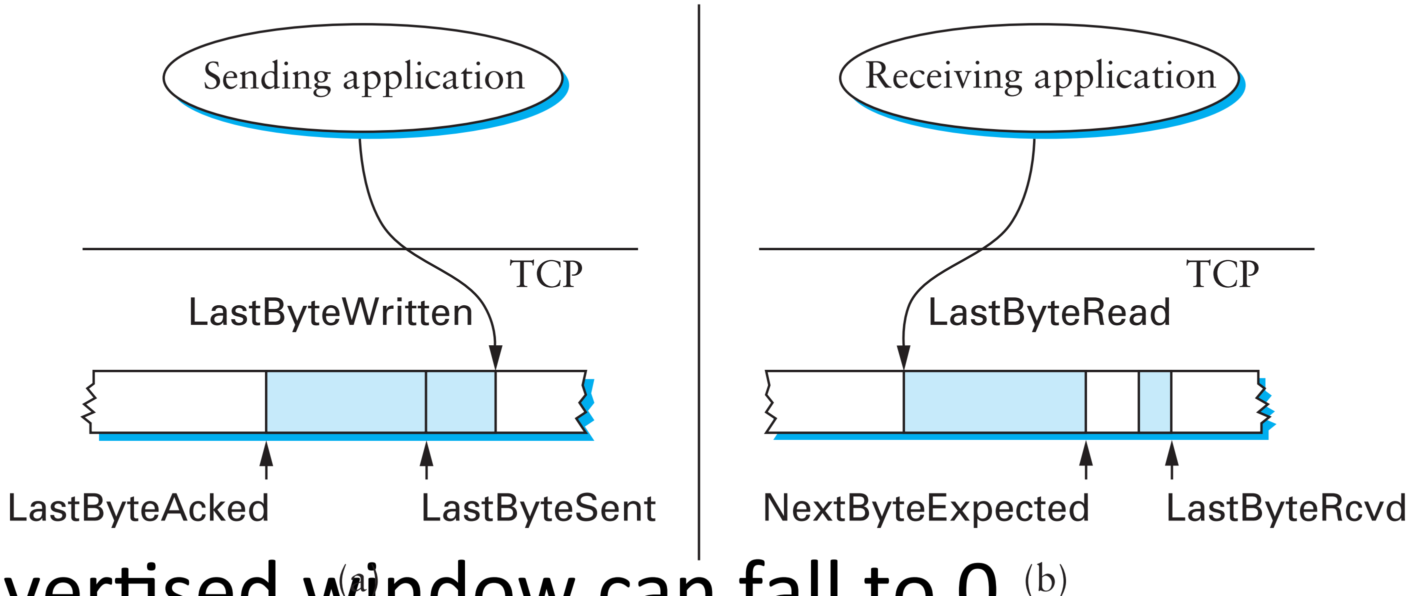


- Receiver: AdvertisedWindow
  = MaxRcvBuffer – ((NextByteExpected-1) – LastByteRead)
- Sender: LastByteSent – LastByteAcked <= AdvertisedWindow

  EffectiveWindow = AdvertisedWindow – (BytesInFlight)

  LastByteWritten – LastByteAcked <= MaxSendBuffer

Flow Control



- Advertised window can fall to 0
  - How?
  - Sender eventually stops sending, blocks application
- Sender keeps sending 1-byte segments until window comes back > 0

- 50 students have ssh window open to bayou and are typing 1 character per second

- How many packets are read and written by bayou per second?
  - Consider minimum frame size

When to Transmit?

- Nagle's algorithm
- Goal: reduce the overhead of small packets

```
If available data and window >= MSS
    Send a MSS segment
else
    If there is unAcked data in flight
        buffer the new data until ACK arrives
    else
        send all the new data now
```

- Receiver should avoid advertising a window <= MSS after advertising a window of 0

http://tools.ietf.org/html/rfc896

Delayed Acknowledgments

- Goal: Piggy-back ACKs on data
  - Delay ACK for 200ms in case application sends data
  - If more data received, immediately ACK second segment
  - Note: never delay duplicate ACKs (if missing a segment)
- Warning: can interact *very* badly with Nagle
  - Temporary deadlock
  - Can disable Nagle with TCP_NODELAY
  - Application can also avoid many small writes

http://en.wikipedia.org/wiki/TCP_delayed_acknowledgment
http://developers.slashdot.org/comments.pl?sid=174457&cid=14515105

Turning Nagle’s Algorithm Off

“In general, since Nagle's algorithm is only a defense against careless applications, it will not benefit a carefully written application that takes proper care of buffering; the algorithm has either no effect, or negative effect on the application.”

- Who wants to turn the algorithm off?
  - Search on Google and find out.

http://en.wikipedia.org/wiki/Nagle's_algorithm

Limitations of Flow Control

- Network may be the bottleneck
- Signal from receiver not enough!
- Sending too fast will cause queue overflows, heavy packet loss
- Flow control provides *correctness*
- Need more for performance: congestion control

A Short History of TCP

- 1974: 3-way handshake
- 1978: IP and TCP split
- 1983: January 1st, ARPAnet switches to TCP/IP
- 1984: Nagle predicts congestion collapses
- 1986: Internet begins to suffer congestion collapses
  - LBL to Berkeley drops from 32Kbps to 40bps
- 1987/8: Van Jacobson fixes TCP, publishes seminal paper: (TCP Tahoe)
- 1990: Fast transmit and fast recovery added (TCP Reno)

Second goal

- We should not send more data than the network can take: *congestion control*

TCP Congestion Control

- 3 Key Challenges
  - Determining the available capacity in the first place
  - Adjusting to changes in the available capacity
  - Sharing capacity between flows
- Idea
  - Each source determines network capacity for itself
  - Rate is determined by window size
  - Uses implicit feedback (drops, delay)
  - ACKs pace transmission (self-clocking)

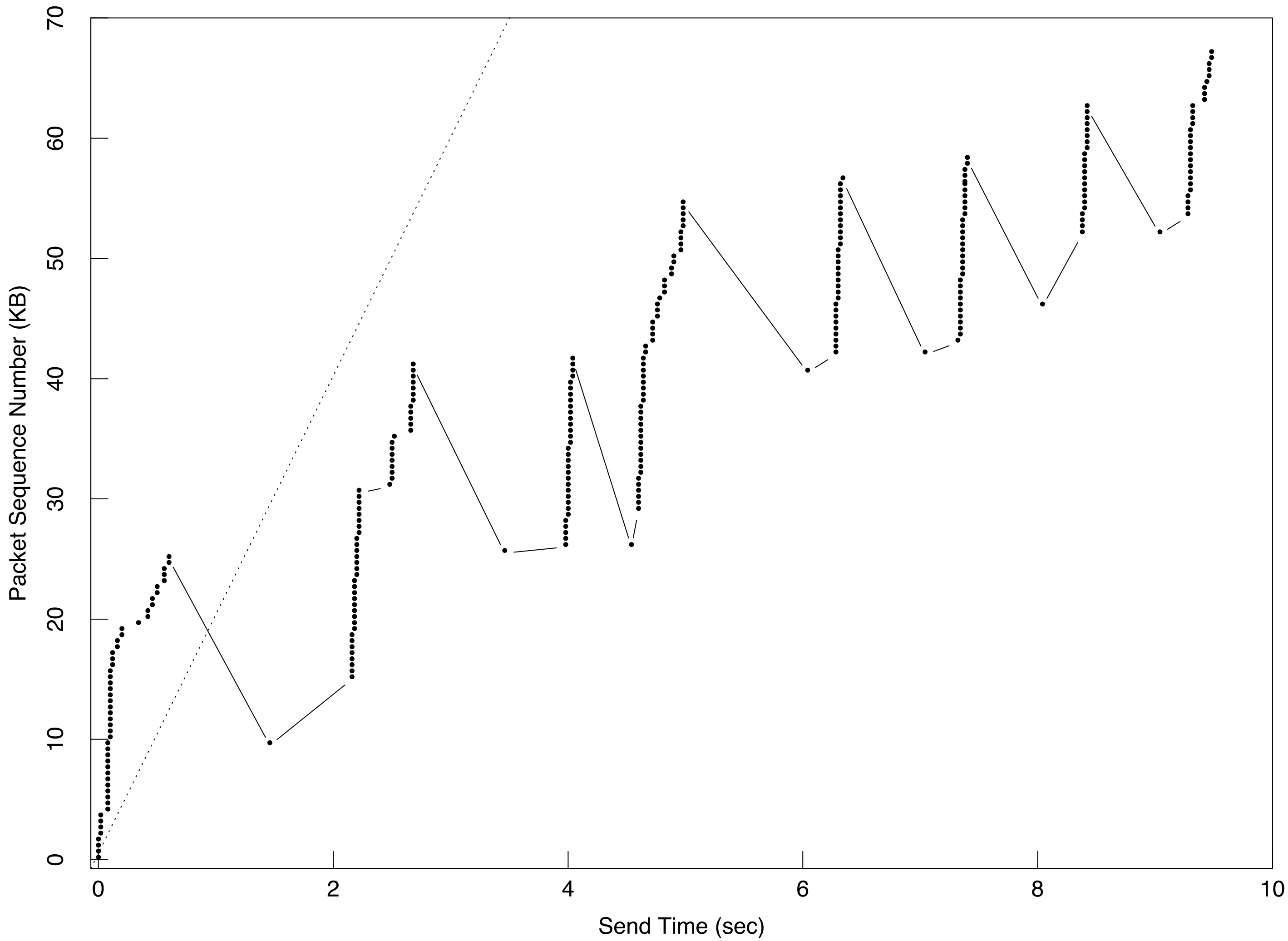
Dealing with Congestion

- TCP keeps congestion and flow control windows
  - Max packets in flight is lesser of two
- Sending rate: ~Window/RTT
- The key here is how to set the congestion window to respond to congestion signals

Starting Up

- Before TCP Tahoe
  - On connection, nodes send full (rcv)window of packets
  - Retransmit packet immediately after its timer expires
- Result: window-sized bursts of packets in network

Bursts of Packets



Graph from Van Jacobson and Karels, 1988

Determining Initial Capacity

- Question: how do we set w initially?
  - Should start at 1MSS (to avoid overloading the network)
  - Could increase additively until we hit congestion
  - May be too slow on fast network
- Start by doubling w each RTT
  - Then will dump at most one extra window into network
  - This is called *slow start*
- *Slow start*, this sounds quite fast!
  - In contrast to initial algorithm: sender would dump entire *flow control* window at once

Startup behavior with Slow Start