Transport Layer

TCP / UDP

Chapter 6

section 6.5 is TCP

12 Mar 2012

Layers

Application

Transport

Network

Host-to-Network/Physical/DataLink

Why do we need the Transport Layer?

High Level Overview

- TCP (RFC 793) UDP

- The network can fail in many ways

We'll talk about the Transport Layer via TCP

Transport Layer

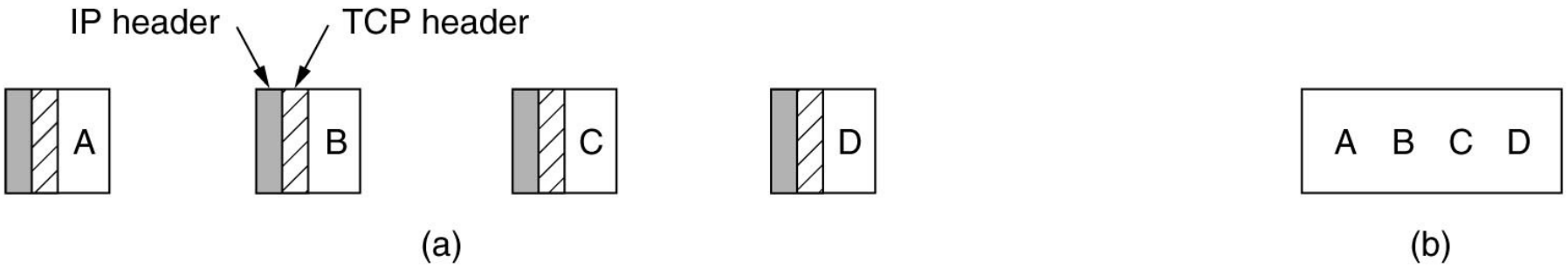
- Connection / Connectionless

- Completely on endpoint comptuers

- Three phases

Segments

- Frames
- Packets
- Segments



Transport Layer

- Addressing

- Connection setup

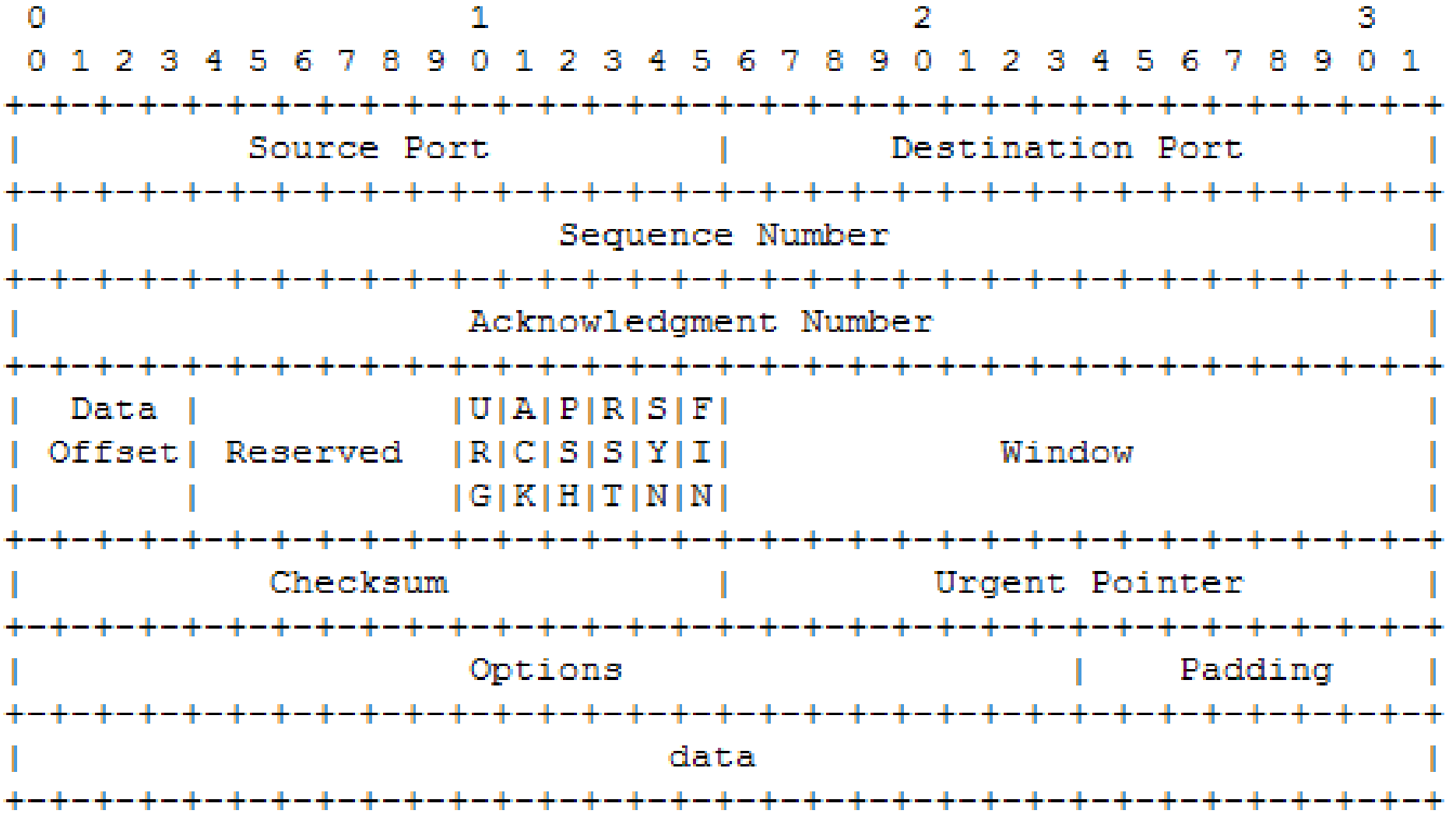
- Storage in the network

- Buffering/performace/expectations

TCP Protocol Basics

- Each byte in the stream has its own 32-bit sequence number

- Basic unit of transfer is the *TCP segment*



TCP Header Format

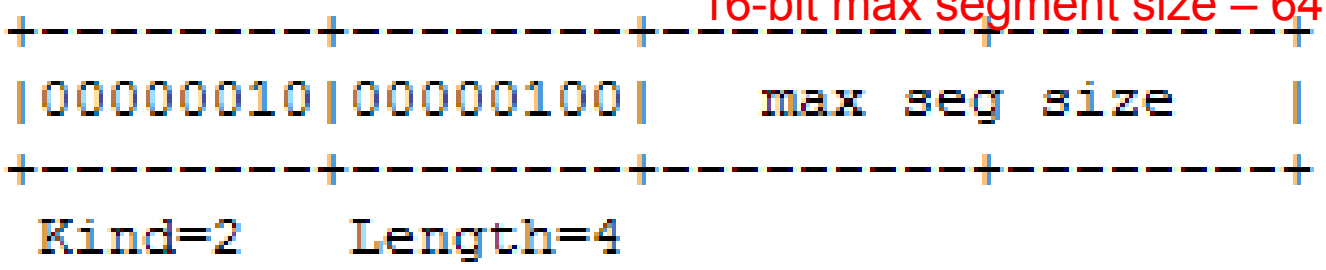
Note that one tick mark represents one bit position.

Figure 3.

TCP Options

Maximum Segment Size

16-bit max segment size – 64KB



Maximum Segment Size Option Data: 16 bits

If this option is present, then it communicates the maximum receive segment size at the TCP which sends this segment. This field must only be sent in the initial connection request (i.e., in segments with the SYN control bit set). If this option is not used, any segment size is allowed.

- If this is not specified the default size is 536 bytes
  - RFC 879
- Both sides send this option to determine segment size
  - both sides must use the same segment size

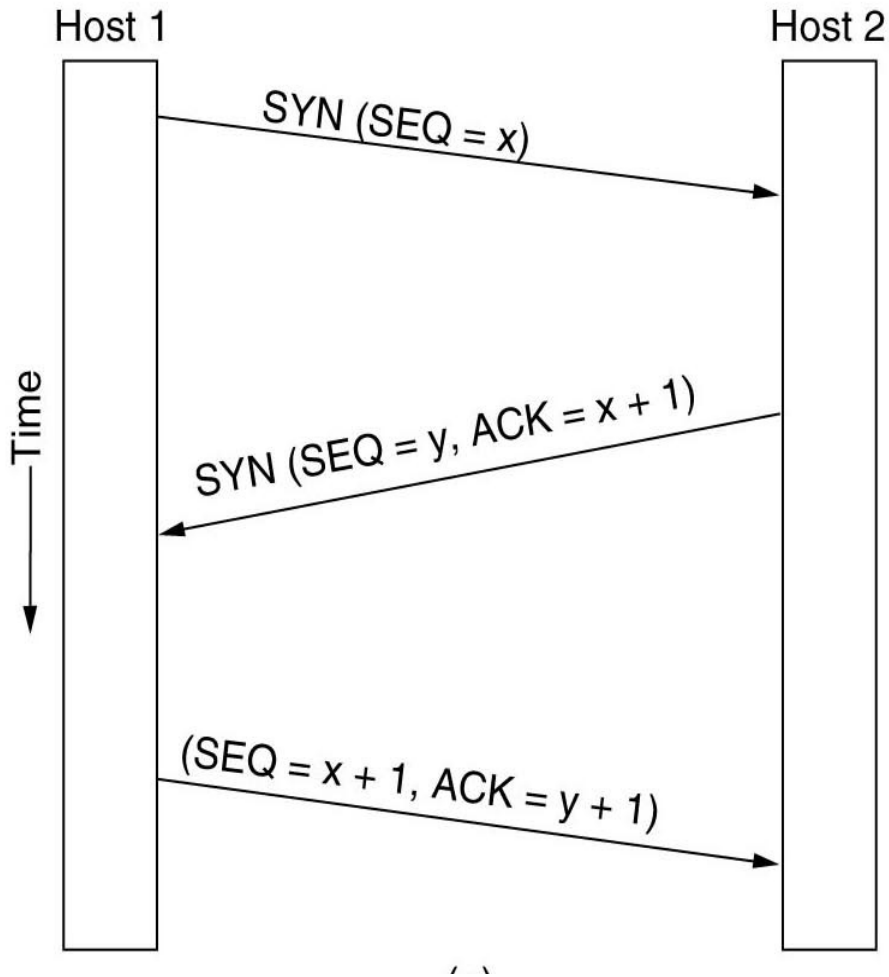
Connection Setup

- What needs to happen?

- What does a reliable byte stream need?

Connection Establishment

- 3-way handshake
- Uses the SYN and ACK bit fields in the TCP header
- Security



Tanenbaum, Figure 6-31, p540

Initial Sequence Number

- Initial sequence number is not zero

Connection Release

- What bad things can happen?

- Two Army Problem

Connection Release

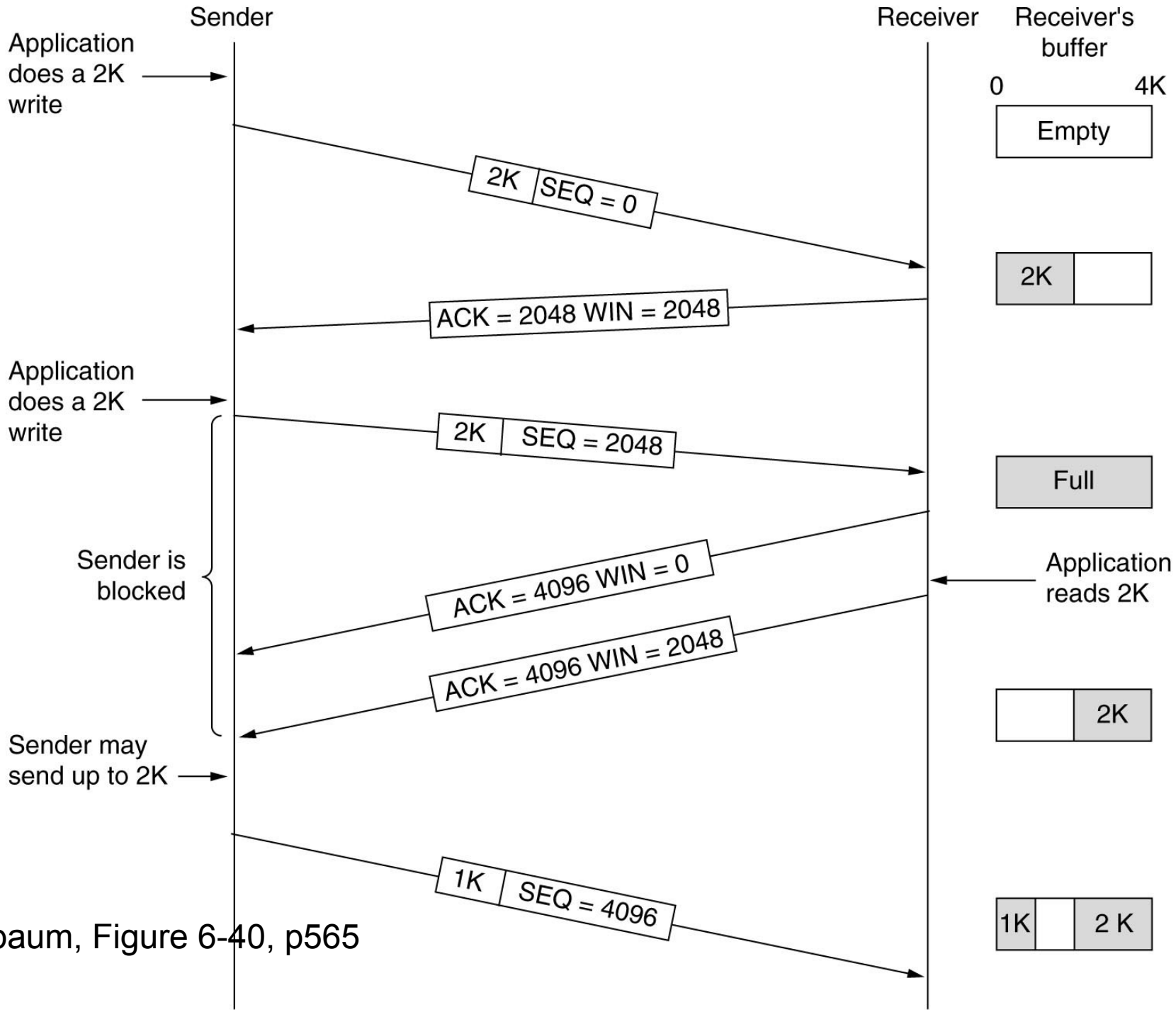
- FIN bit

Sliding Window

- Sliding window protocol used to manage flow/retransmission

- Acknowledgement number denotes the **next** byte to be received

- Window size denotes how many more bytes may be sent



Tanenbaum, Figure 6-40, p565

Initial Sequence Number Attack

- Kevin Mitnick

Buffers

-

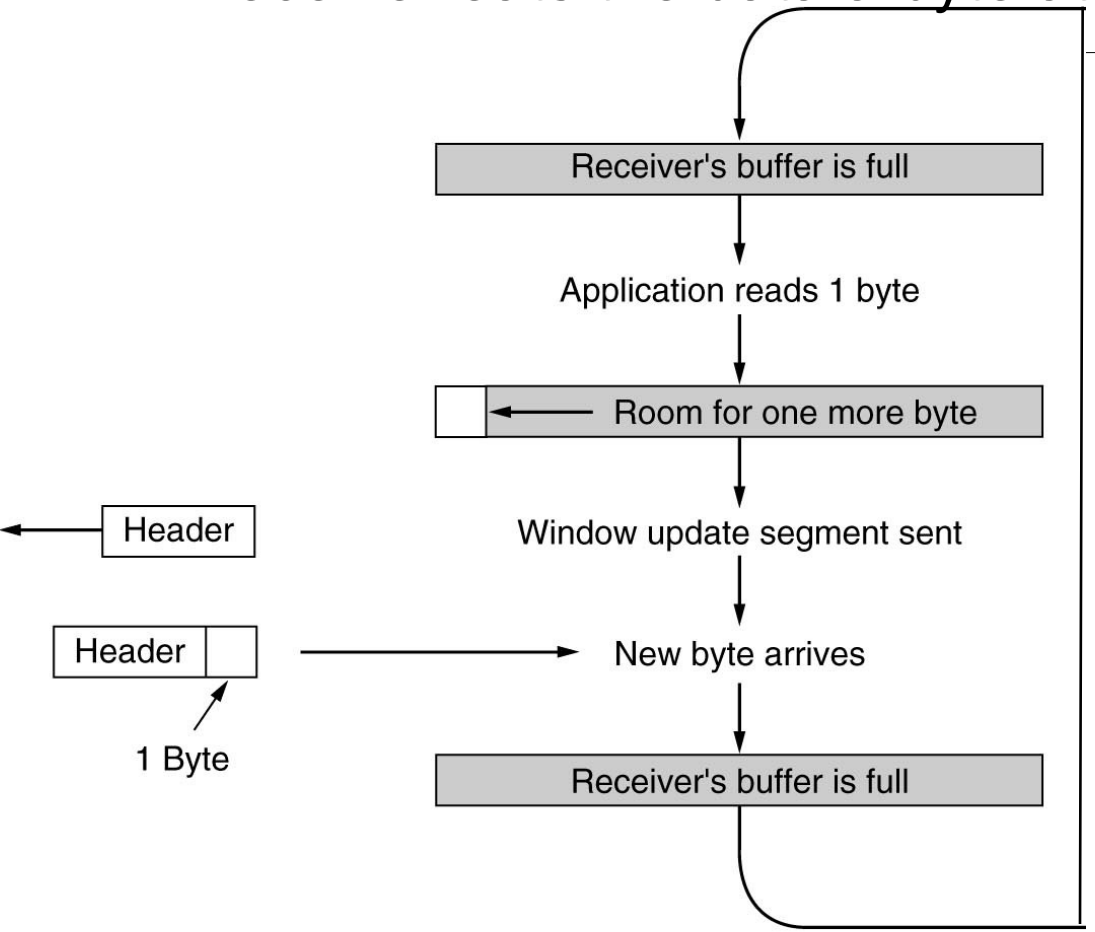
Nagle

- When data comes in to the buffer one byte at a time

- Sometimes needs to be disabled
  - when might this be bad?

Silly Window Syndrome

- Sender produces large chunks of data
- Receiver eats the data a byte at a time



What is the problem?

How can we fix this?

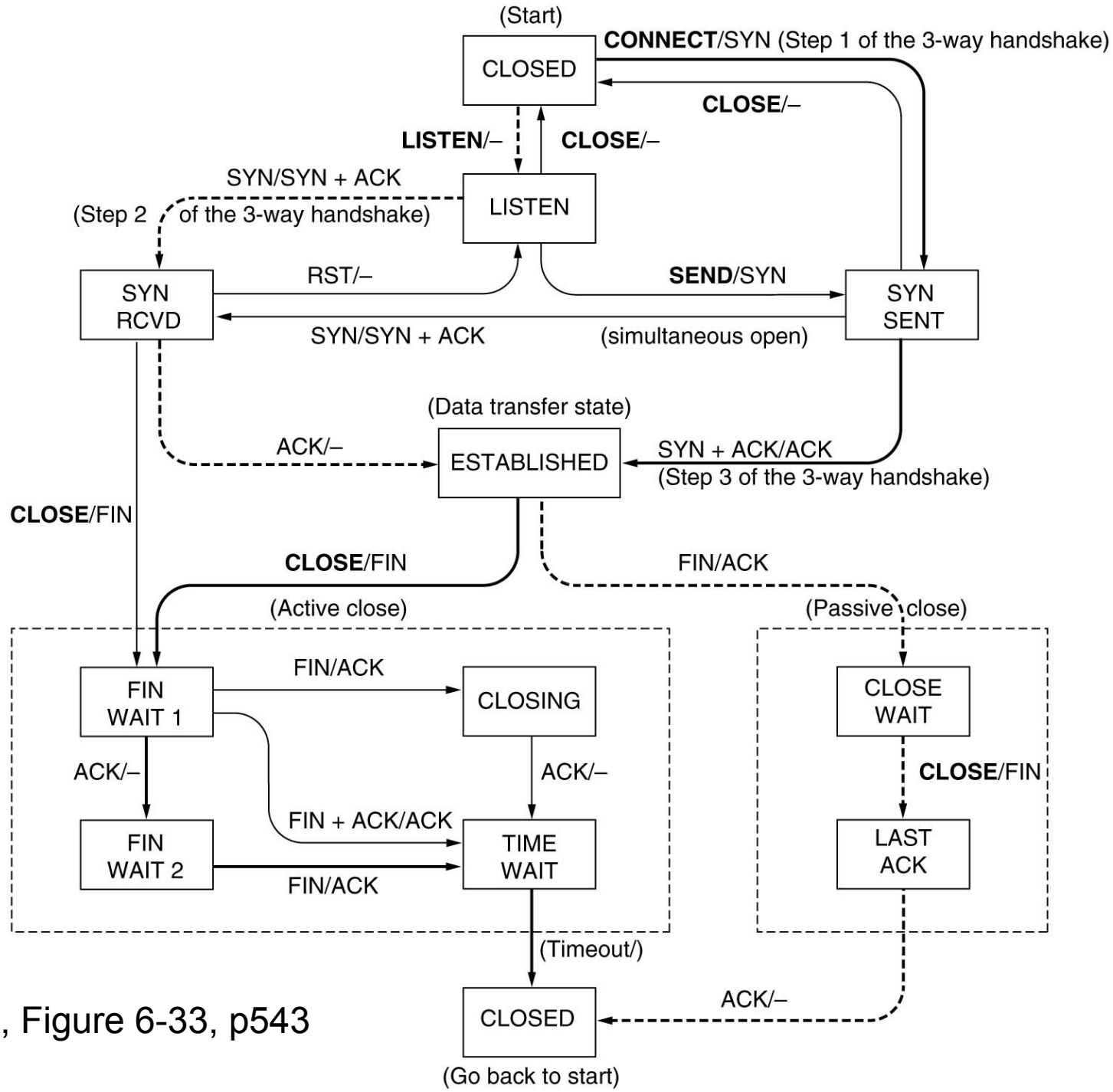
Tanenbaum, Figure 6-41, p568

No one will ever need more than 64KB!

- Max window size: 64 KB.
  - Goal:

- On a T3 line (44.736 Mbps)

- Hack (Fix) RFC 1323



Tanenbaum, Figure 6-33, p543

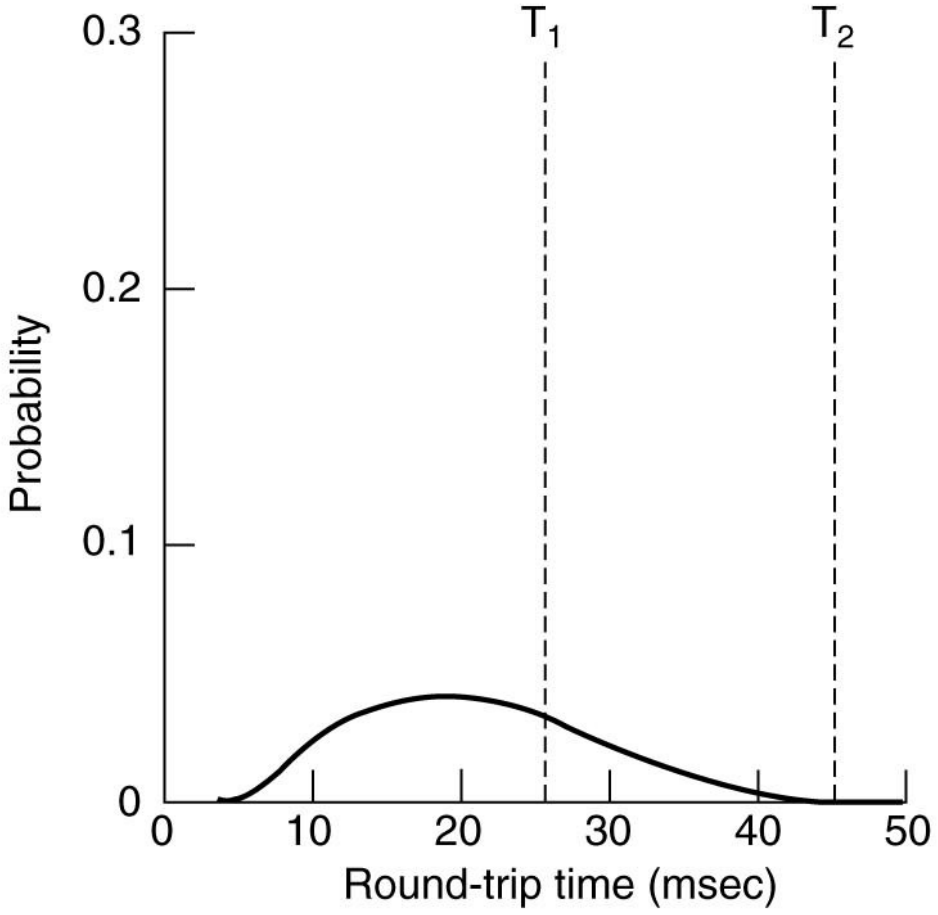
Reliability

- Retransmit!

Timers & Retransmission

- Send segment & start timer

- How long should the timeout interval be?
  - what factors do we need to consider?
  - what makes this hard?
  - what happens if we set the timerout to T1? T2?



Tanenbaum, Figure 6-38, p551

Timers

- Each connection keeps a value RTT

- RTT = α*RTT + (1 - α)*M

- Timeout: β*RTT

- Jacobson proposed making β roughly proportional to standard deviation of the ACK time
  - use mean deviation as cheap estimator
  - variable named D

Timers

- $D = \alpha * D + (1- \alpha) * | RTT - M|$

- Timeout value: RTT + 4 * D

- When might you get two ACKs for the same segment?
  - why might this cause problems?
  - solution?

Other Timers

- Persistence Timer

- Keepalive Timer

ACKs & Retransmission

- When does the receiver ACK?

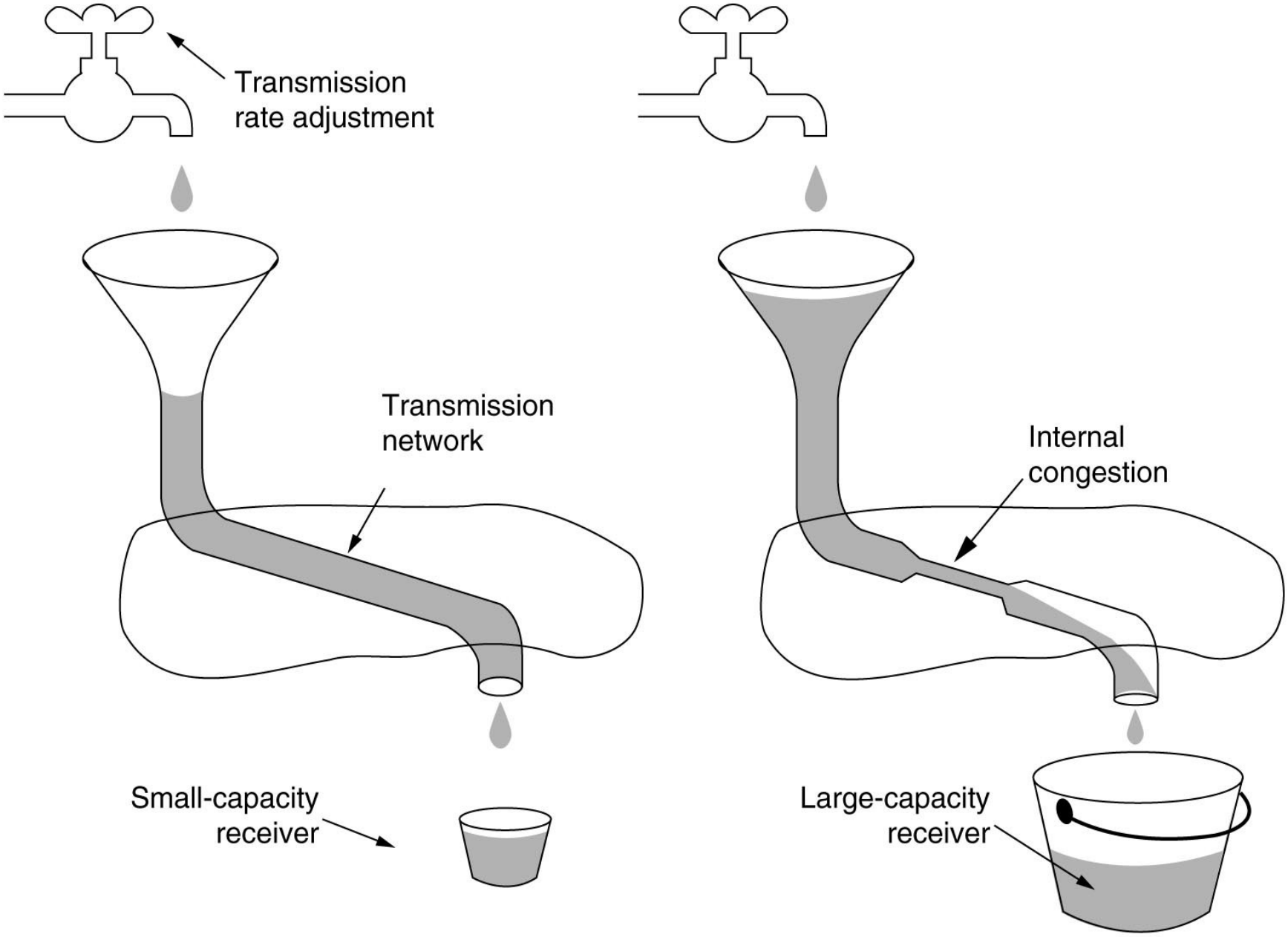
- Fast Retransmit (RFC 2581)

Congestion Control

- Some done in Network Layer
- Most done in Transport Layer
  - how do you control congestion?

- Law of conservation of packets:
  - don't send new packets until old ones are received

- Dynamically manipulate window size to control how much data is in the network
- How can we detect congestion in the network?



Tanenbaum, Figure 6-36, p548

Implementation Details

- Receiver Capacity
  - receiver window (based on receiver buffer size, load, etc)

- Network Capacity
  - congestion window (sender window)
- Use the minimum of the two values
- How do we find the Congestion Window Size?

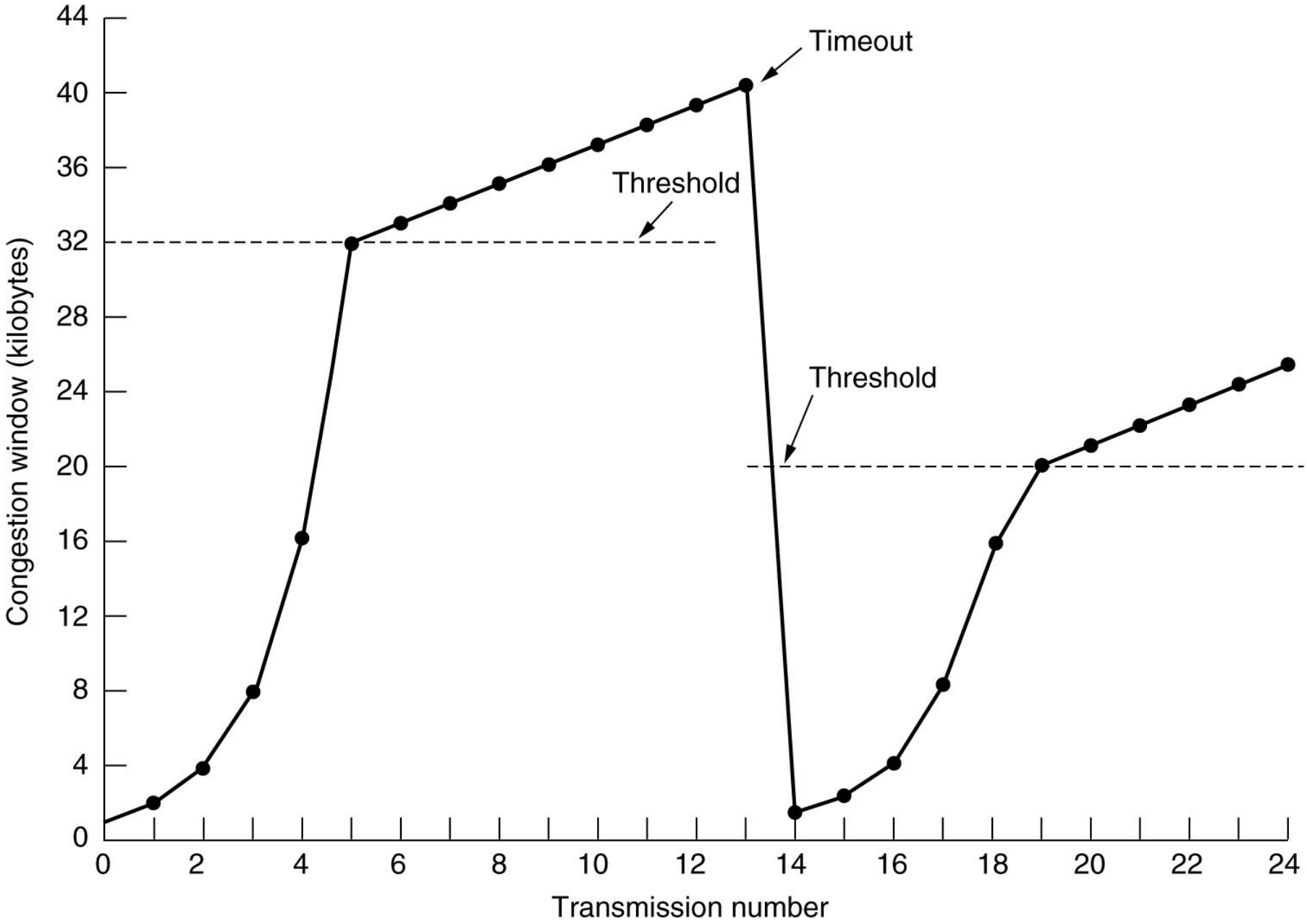
A Loss

- What happens when there is a lost packet?
  - what does that loss tell us?
  - what should we do in response to that?

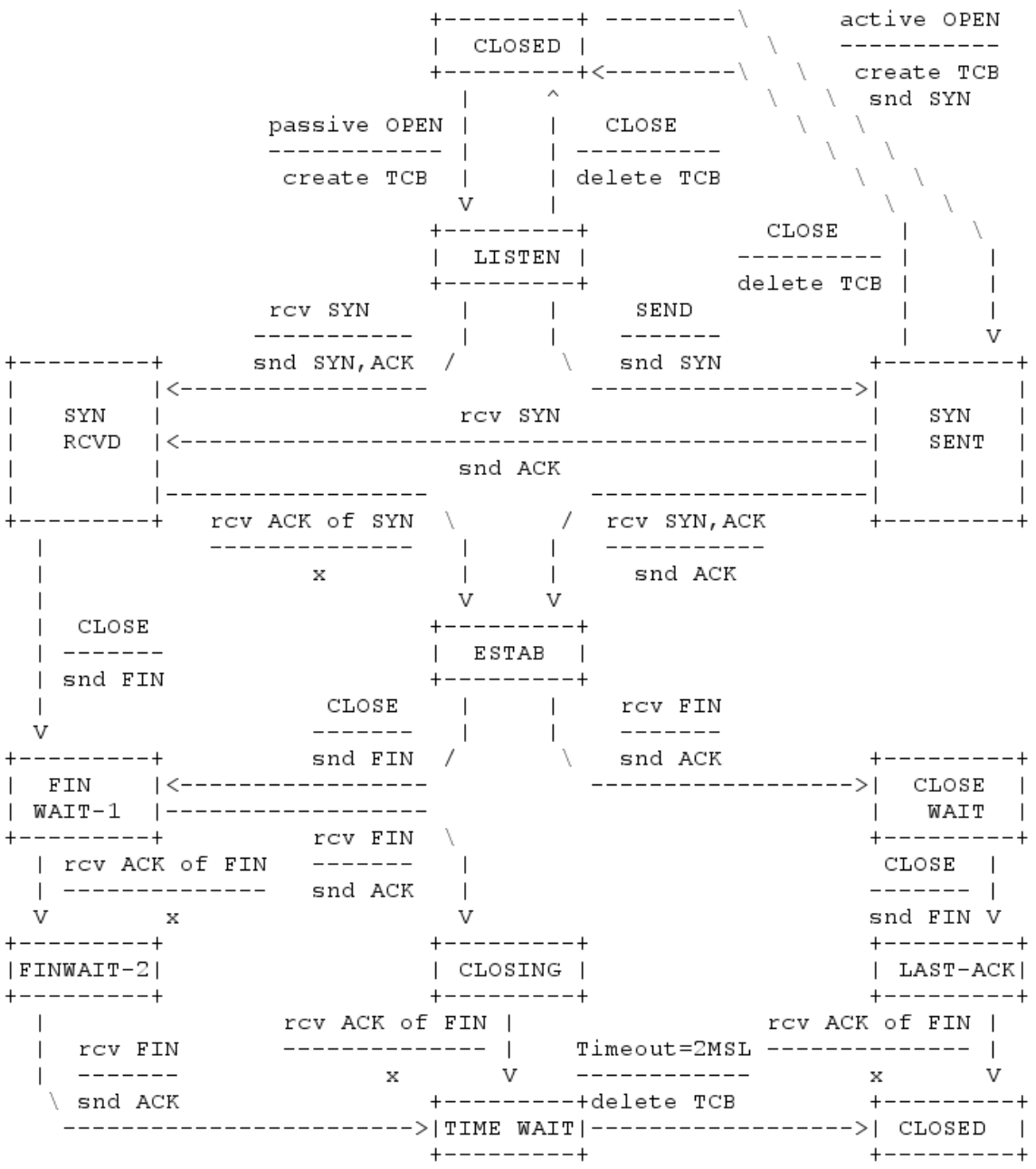
More Details

- When a timeout occurs, we reduce the congestion window (TCP Tahoe)

- TCP Reno (Fast Recovery)



Tanenbaum, Figure 6-37, p550



TCP Connection State Diagram
Figure 6.

http://www.ietf.org/rfc/rfc793.txt

Congestion, RFC 3168 & 5681

- ECN: Explicit Congestion Notification
  - TCP Flag: ECE/CWR
  - Random Early Detection (RED)

In the Network

- Accept packets until buffers fill
  - Tail drop
  - TCP Global Sync
- Random Early Detection
  - as buffer fills, the probability of dropping/marking a packet increases
  - full buffer, probability is 1
  - does not penalize bursty traffic

PSH

A sending TCP is allowed to collect data from the sending user and to send that data in segments at its own convenience, until the push function is signaled, then it must send all unsent data. *When a receiving TCP sees the PUSH flag, it must not wait for more data from the sending TCP before passing the data to the receiving process.* RFC 793

socket.receive(...); // may block

Congestion & Goodput

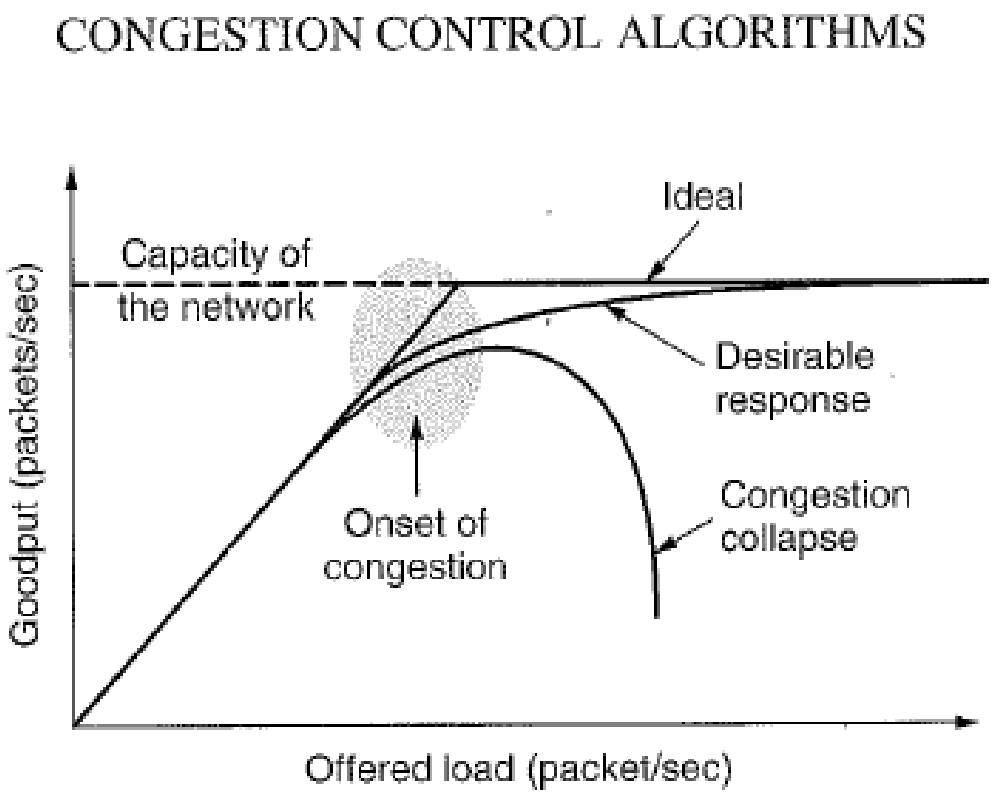


Figure 5-21. With too much traffic, performance drops sharply.

Congestion (6.3)

- Goal: keep the pipe full!

- 100Mbps link, with 5 senders, how much bandwidth each?

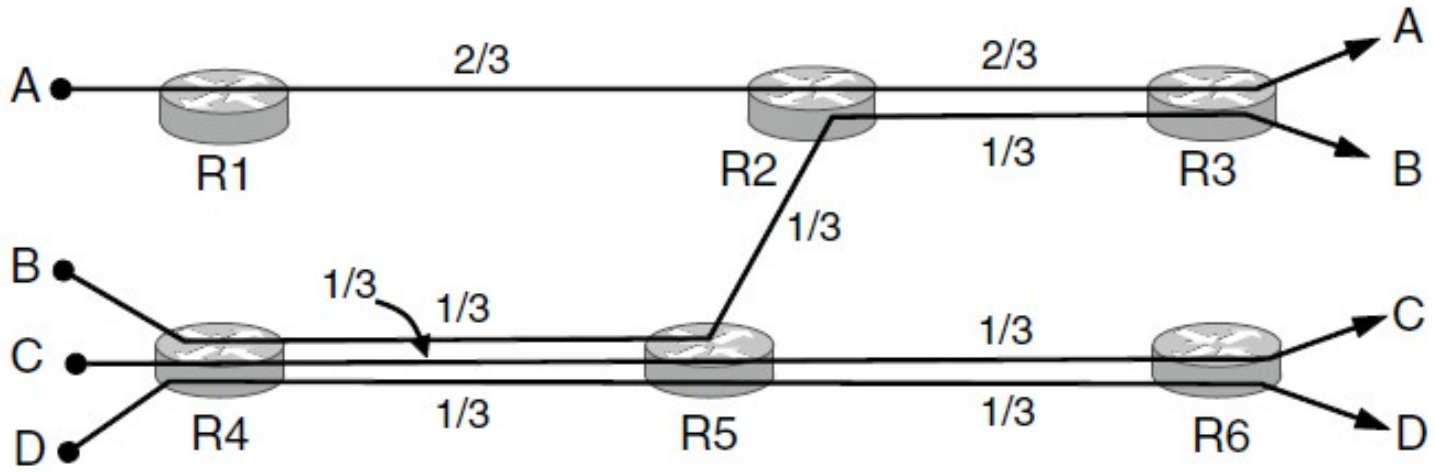
- Traffic is bursty

- How do we divide up the bandwidth?
  - what determines how much bandwidth a connection can fully take advantage of?

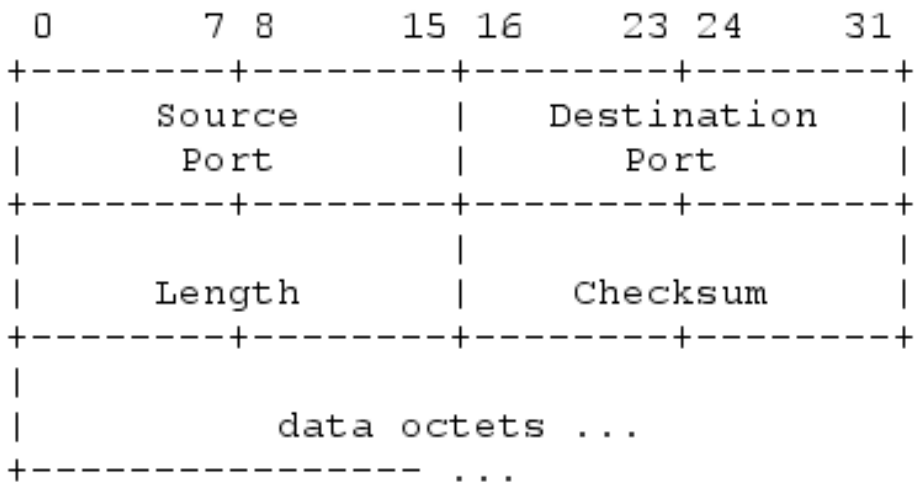
Power

- power = load / delay Kleinrock

- max-min fairness



UDP RFC 768



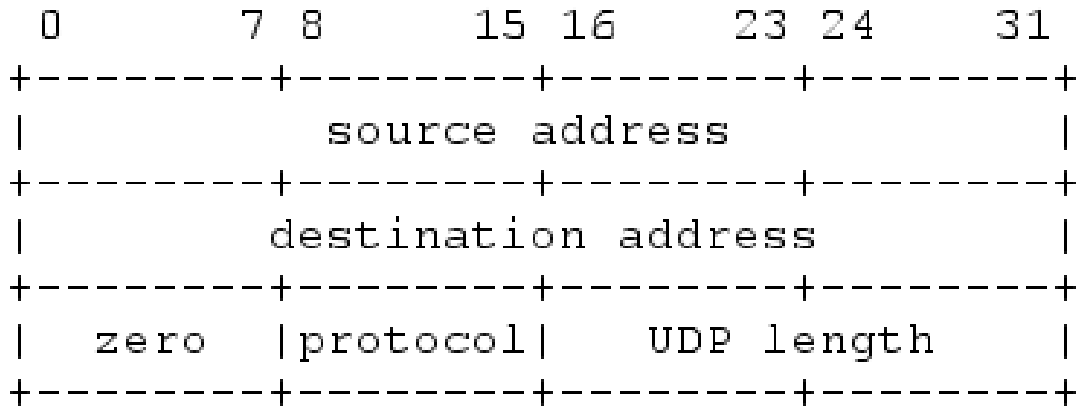
User Datagram Header Format

UDP

- Does ***not***
  - flow control
  - congestion control
  - retransmission
- Interface to IP with ports added
- Why?

Checksum

Protocol = 17



Checksum is the 16-bit one's complement of the one's complement sum of a pseudo header of information from the IP header, the UDP header, and the data, padded with zero octets at the end (if necessary) to make a multiple of two octets. (RFC 768)

This is exactly how the TCP (IPv4) checksum works. Protocol = 6

1's Complement

- Invert all the bits

  1 and -1 are inversions of each other

- Addition

negative zero!

- Subtraction