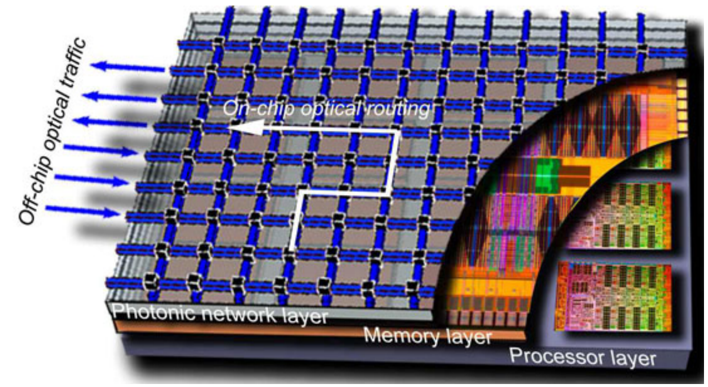
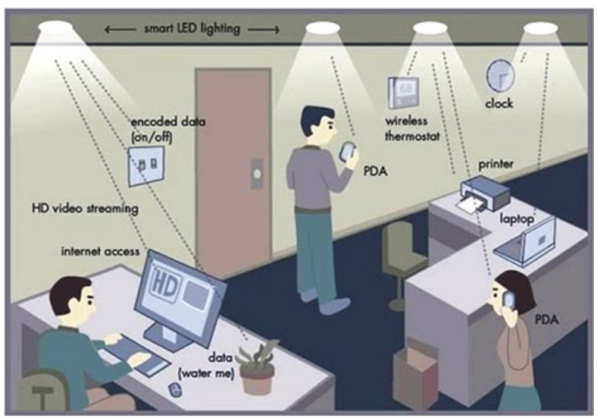
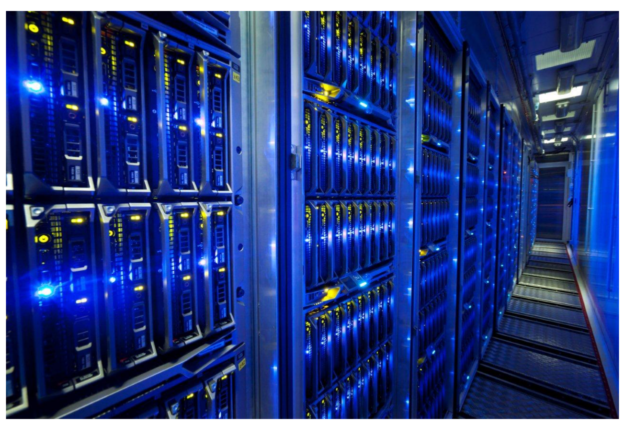
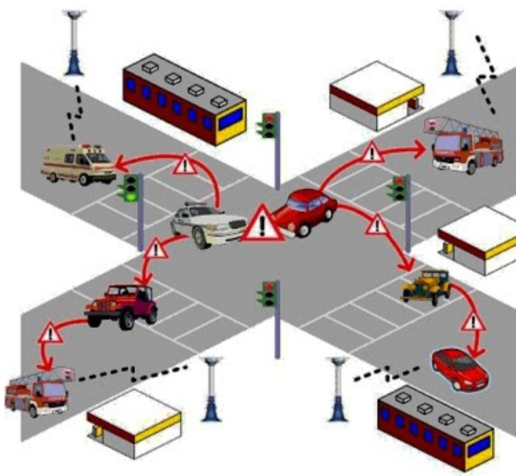


COSC 402

Advanced Computer Networks







About This Course

- Lectures
  - Internetworking with TCP/IP
  - Programming using Socket API
  - Wireless sensor networks, Internet of Things, Cyber-physical systems
  - Other advanced networking topics
    - 4G & 5G networks
    - Datacenter networks
    - Li-Fi networks & network-on-chips
    - Mobile social networks
    - Vehicular ad-hoc networks
    - Software-defined networks
- Aims
  - Sufficient background in advanced network theory
  - Necessary skills in network programming
  - Practice in creative thinking about computer networks

Teaching Team

- Lecturer and lab demonstrator

Dr Haibo Zhang
Email: haibo@cs.otago.ac.nz
Phone: 479-8534
Office: 2.47 Owheo Building

Dr Yawen Chen
Email: yawen@cs.otago.ac.nz
Phone: 479-5740
Office: 2.46 Owheo Building

Course details

- Lectures
  - Thursday 11:00-12:50am
- Labs
  - Thursday 14:00-16:00pm
  - No labs in the first two weeks
- Textbook (recommended)
  - Part I: Unix Network Programming, Vol. 1, The Sockets Networking API (3rd ed), W.R. Stevens, B. Fenner, A.M. Rudoff, Addison Wesley
  - Part II & Part III: no particular recommended textbook.

Assessments

Internal assessment (40%) + Exam (60%)

- Programming Assignments (20%)
  - Assignment 1: implement a multi-user chat system
    Assignment 2: implement a multi-hop routing scheme for wireless sensor networks

The above assignment needs to be done in C. If you are not familiar with it, please take the first two weeks to learn it.

- Project (20%)
- Exam (60%)
  - 3 hours

Lecture Notes

- No hardcopy lecture handouts will be provided.
- Lecture slides will be available on the course webpage approximately one week before the corresponding lecture.

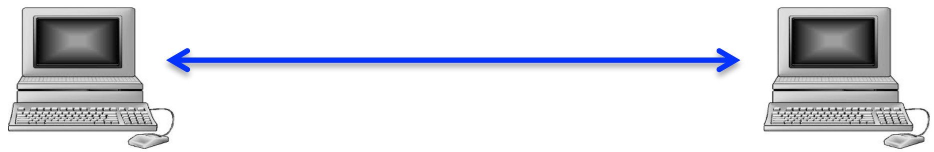
http://www.cs.otago.ac.nz/tele402/schedule.php

Lecture 1 Overview

- This Lecture
  - Protocol design principles
  - TCP and UDP
  - Source: Chapter 2
- Next Lecture
  - Sockets introduction
  - Elementary TCP sockets
  - TCP Client-Server example
  - Source: Chapters 3, 4, and 5

Network Protocols

- Why do we need network protocols?



Allow one to specify or understand communication without knowing the details of the network hardware

- Problems that might arise during communication
  - Hardware failure
  - Network congestion
  - Packet delay or loss
  - Data corruption
  - Data duplication or inverted arrivals

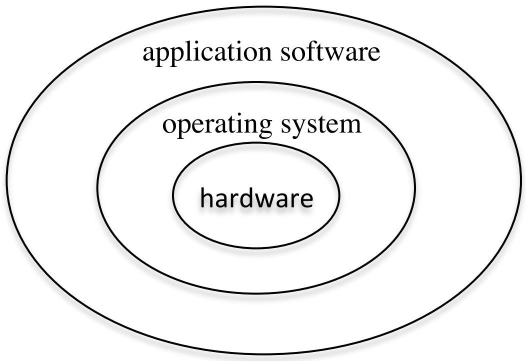
Network Protocols

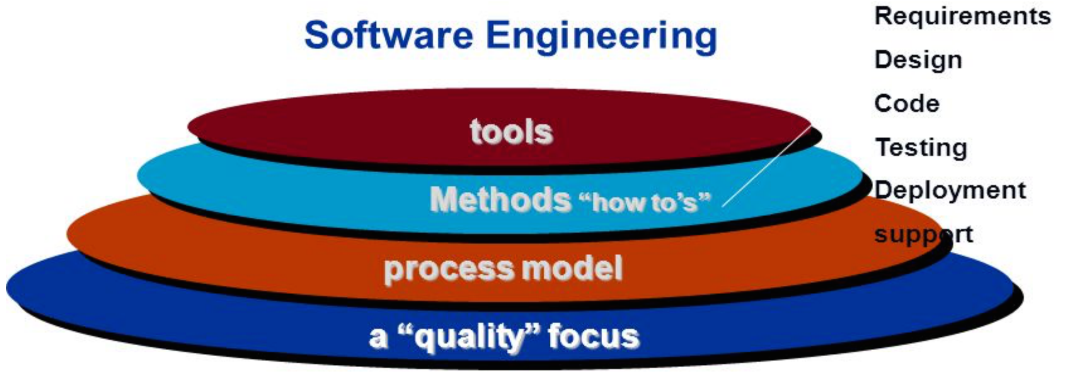
- Is it possible to design a single protocol which handles all problems occurred during data communication?

might be possible, but very difficult

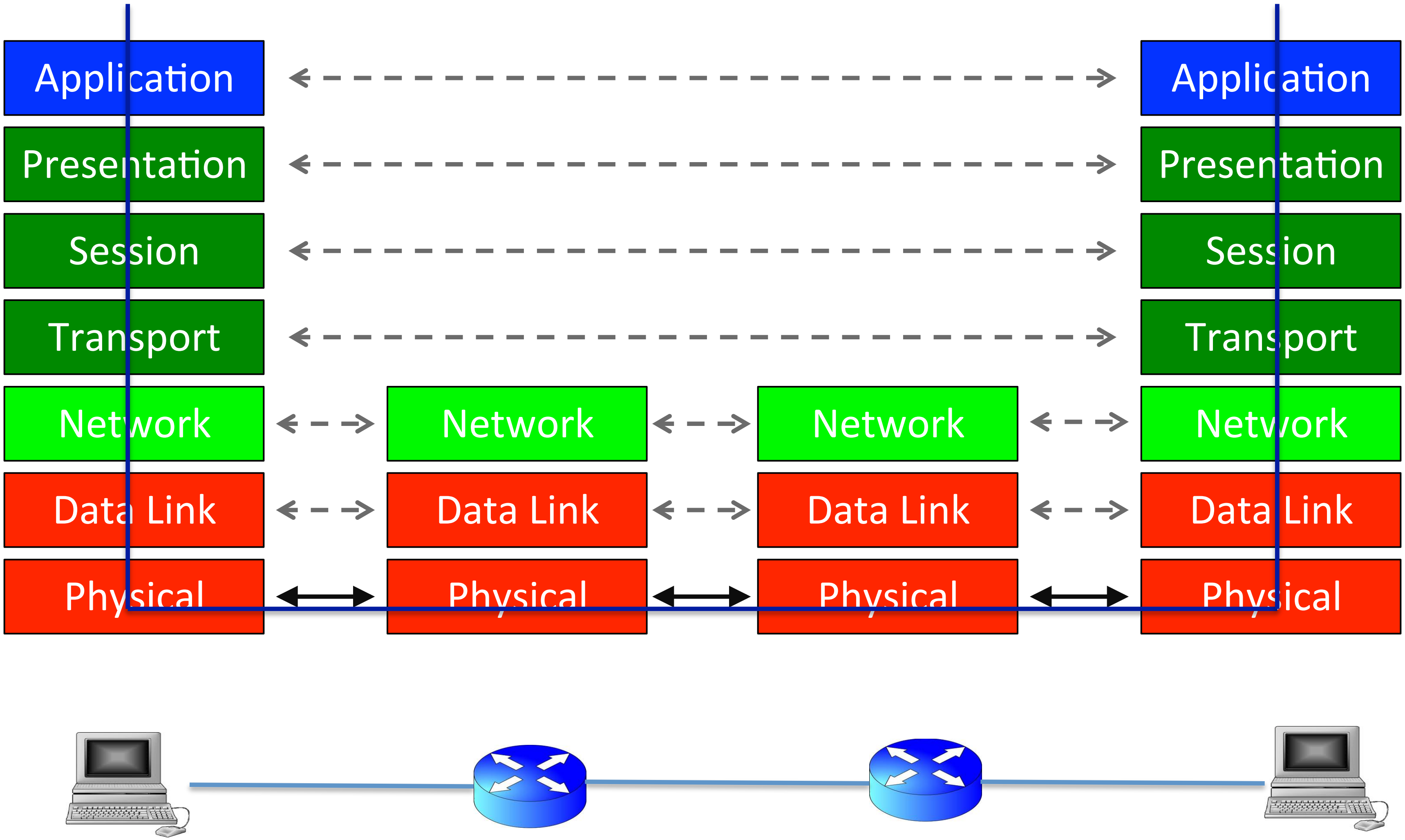
- Layered design approach
  - Not new

Computer system





Open Systems Interconnection (OSI)

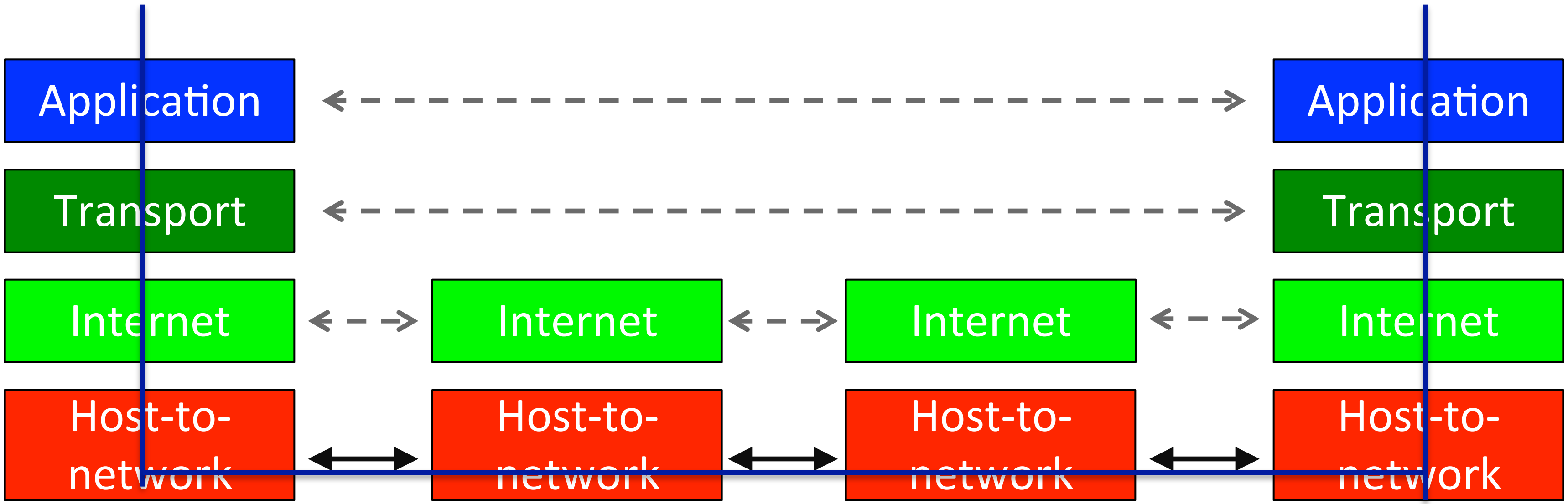


Open Systems Interconnection (OSI)

Application	everything else
Presentation	byte ordering, security
Session	how to tie flows together
Transport	how to send packets end-to-end
Network	how to route packets
Data Link	how to transmit frames
Physical	how to transmit bits

Layering: modular approach to network functionality

TCP/IP Model



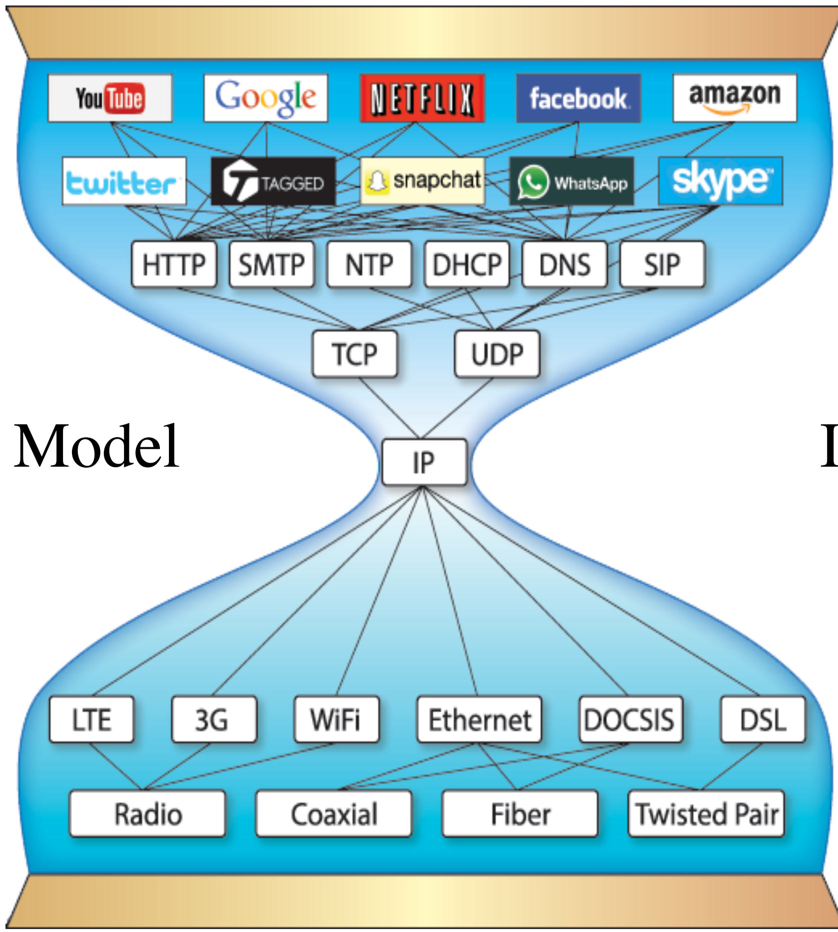
The Protocol Layering Principle

- Each layer is able to perform two opposite tasks for bidirectional communication, e.g. send and receive, encrypt and decrypt.
- Two objects under each layer are identical, i.e., layer n at the destination receives exactly the same object sent by layer n at the source.

Multiplexing/Demultiplexing

- Why are multiplexing and demultiplexing necessary?

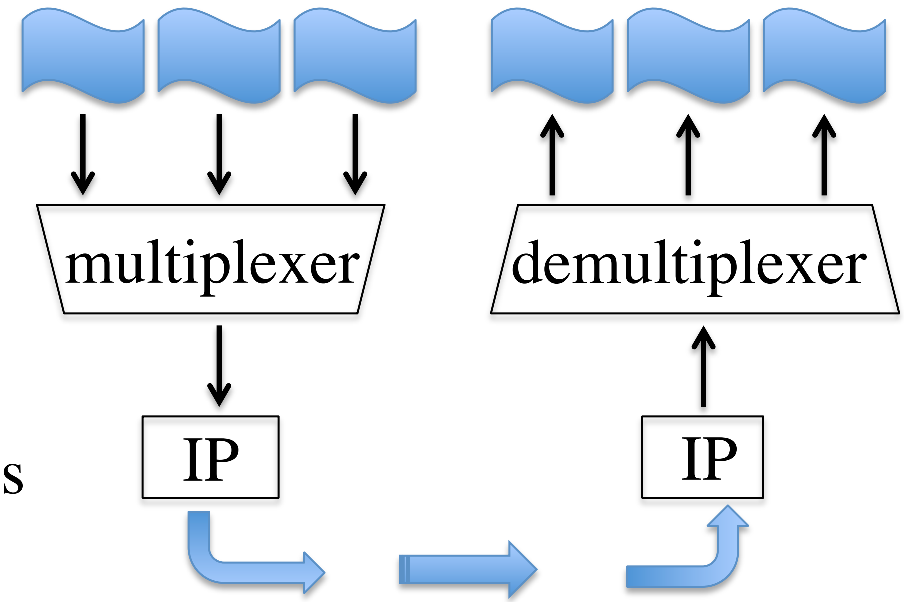
The Hourglass Model



Interoperability

Multiplexing/Demultiplexing

- Occurs at multiple layers
  - TCP
  - IP



- Each header includes some fields used to identify the next layer
  - Filled in by the sender
  - Used by the receiver

<table>
<tr><td>VER
4 bits</td><td>HLEN
4 bits</td><td>Service
8 bits</td><td colspan="2">Total length
16 bits</td></tr>
<tr><td colspan="3">Identification
16 bits</td><td>Flags
3 bits</td><td>Fragmentation offset
13 bits</td></tr>
<tr><td colspan="2">Time to live
8 bits</td><td>Protocol
8 bits</td><td colspan="2">Header checksum
16 bits</td></tr>
<tr><td colspan="5">Source IP address</td></tr>
<tr><td colspan="5">Destination IP address</td></tr>
<tr><td colspan="5">Option</td></tr>
</table>

Protocol Layering: Pros and Cons

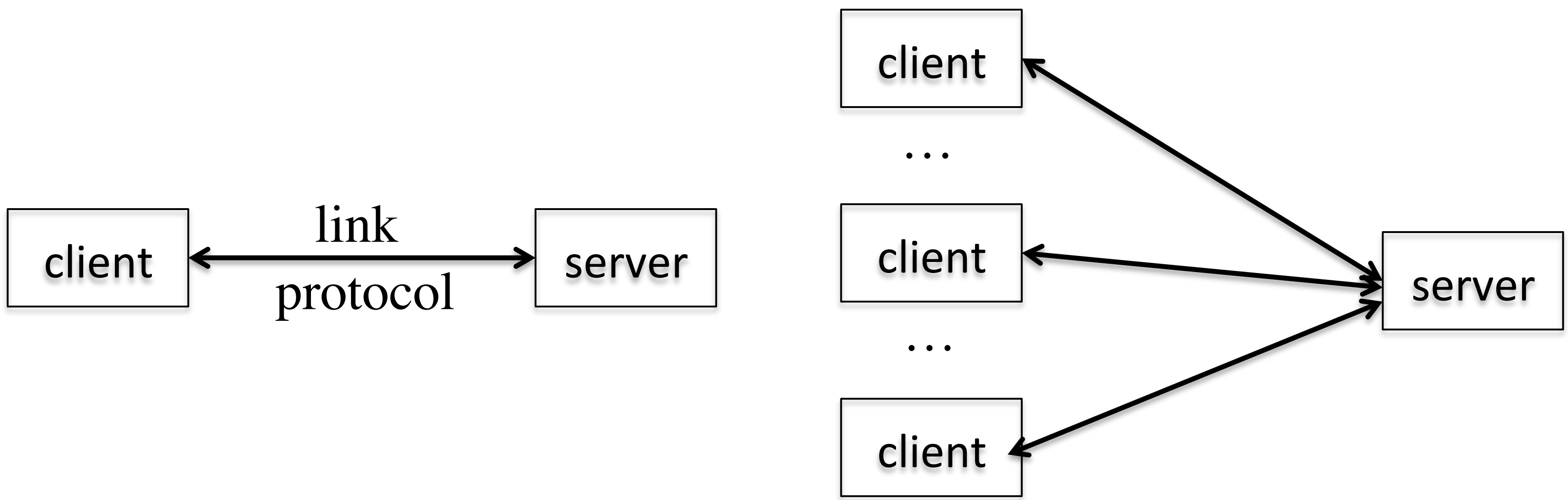
- Pros
  - Modularity, simplicity, interoperability, robustness, security, cost effective

- Cons
  - Complexity, process time, memory usage, prevention from optimization

The TCP/IP Protocol Suit
(The Internet Protocol Suite)

Example: Client and Server (1)

- Simple model
  - One server, multiple clients



- How to make applications robust?

Example: Client and Server (2)

- Local Area Network (LAN) Scenario

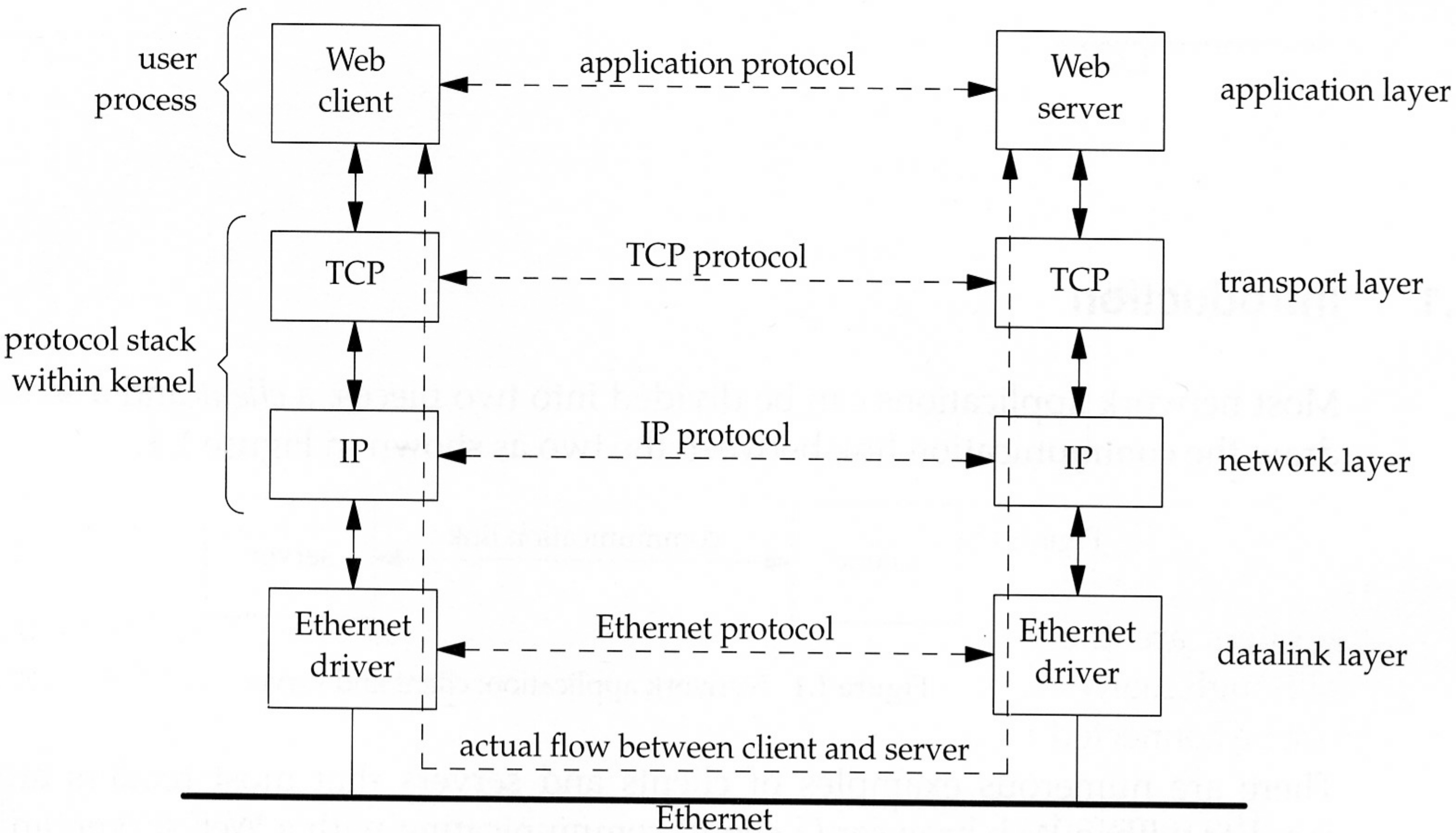


Figure 1.3 Client and server on the same Ethernet communicating using TCP.

Example: Client and Server (3)

- Wide Area Network (WAN) Scenario

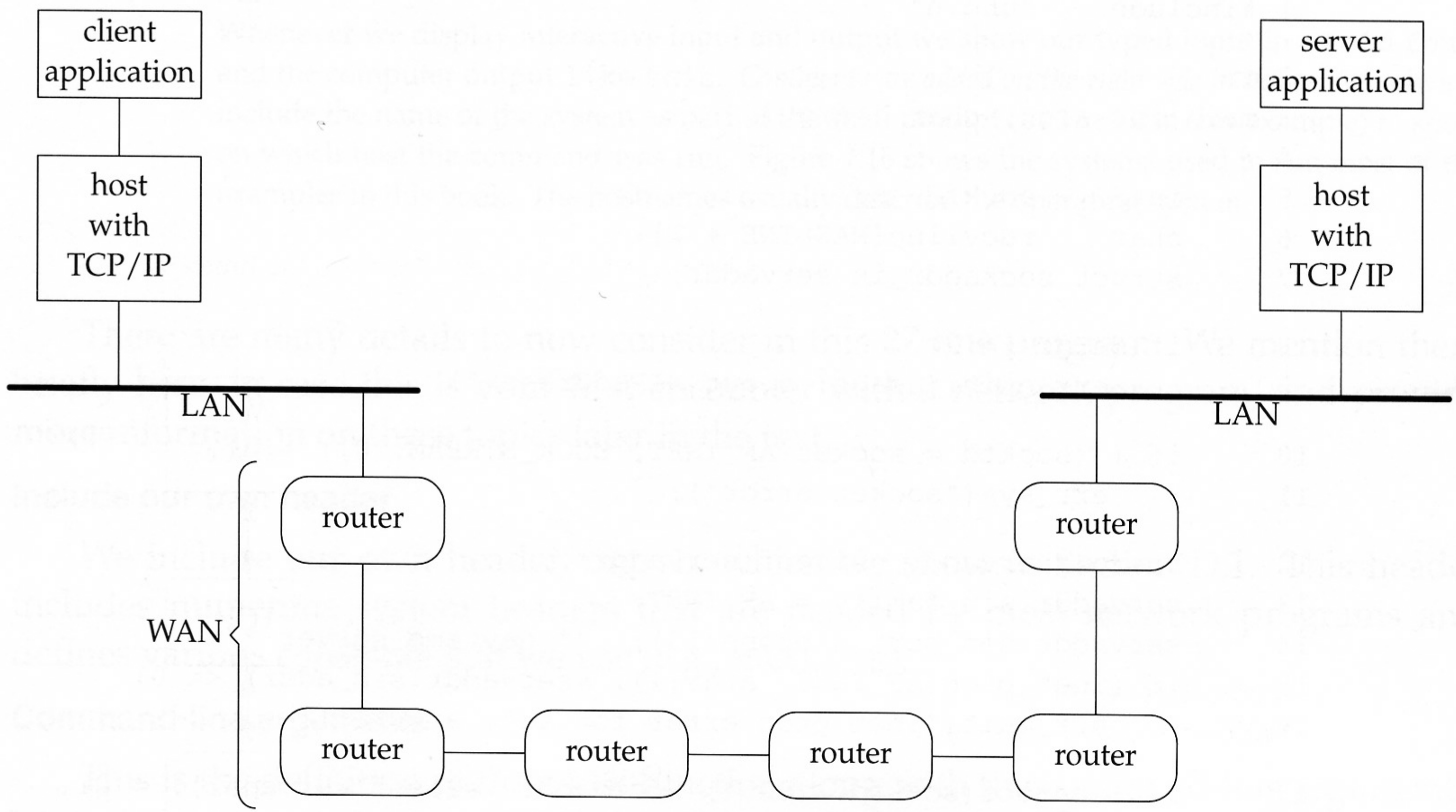


Figure 1.4 Client and server on different LANs connected through a WAN.

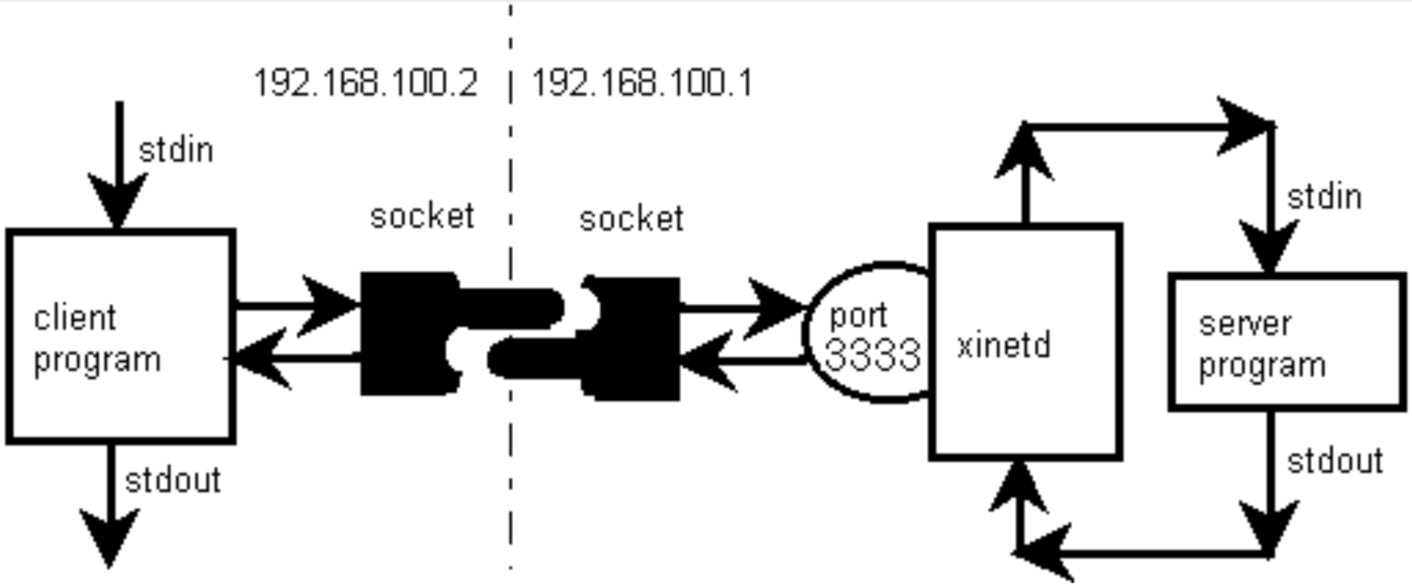
How to Develop the Program?

- Socket Programming
- What is a socket?

Without the telephone network, each endpoint of a telephone line is nothing more than a plastic box.

- Sockets represent endpoints in a line of communication.
- A socket is a **software component** characterized by a unique combination of
  - Local socket address: local IP address and port number
  - Remote socket address: only for TCP sockets
  - Protocol: TCP, UDP

Socket Programming



- Socket Address: the combination of an IP address and a port number (a 16-bit unsigned integer, ranging from 0 to 65535).
- Socket API: an application programming interface, usually provided by the operating system.

A Simple TCP Daytime Client

```
#include    "unp.h"
int
main(int argc, char **argv)
{
    int                       sockfd, n;
    struct sockaddr_in6       servaddr;
    char                      recvline[MAXLINE + 1];
    if (argc != 2)
        err_quit("usage: a.out <IPaddress>");
    if ( (sockfd = socket(AF_INET6, SOCK_STREAM, 0)) < 0)    err_sys("socket error");
    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin6_family = AF_INET6;
    servaddr.sin6_port   = htons(13);/* daytime server */
    if (inet_pton(AF_INET6, argv[1], &servaddr.sin6_addr) <= 0)
        err_quit("inet_pton error for %s", argv[1]);
    if (connect(sockfd, (SA *) &servaddr, sizeof(servaddr)) < 0)    err_sys("connect error");
    while ( (n = read(sockfd, recvline, MAXLINE)) > 0) {
        recvline[n] = 0;       /* null terminate */
        if (fputs(recvline, stdout) == EOF)
            err_sys("fputs error");
    }
    if (n < 0)     err_sys("read error");
    exit(0);
}
```

Create a TCP socket (`socket`)

Specify server's IP address and port

Connect to the server (`connect`)

Send request or receive reply (`send & recv`)

Terminate program (close socket)

A Simple TCP Server

```
#include"unp.h"
#include<time.h>
int main(int argc, char **argv)
{
 intlistenfd, connfd;
 struct sockaddr_inservaddr;
 charbuff[MAXLINE];
 time_tticks;
 listenfd = Socket(AF_INET, SOCK_STREAM, 0);
      bzero(&servaddr, sizeof(servaddr));
 servaddr.sin_family= AF_INET;
 servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
 servaddr.sin_port= htons(13);   /* daytime server */
 Bind(listenfd, (SA *) &servaddr, sizeof(servaddr));
 Listen(listenfd, LISTENQ);
 for ( ; ; ) {
  connfd = Accept(listenfd, (SA *) NULL, NULL);
  ticks = time(NULL);
  snprintf(buff, sizeof(buff), "%.24s\r\n", ctime(&ticks));
  Write(connfd, buff, strlen(buff));
  Close(connfd);
      }
}
```

- Create a TCP socket (`socket`)
- Specify server's IP address and port
- Bind socket with local port (`Bind`)
- Convert the socket to listening socket (`Listen`)
- Accept client connection (`Accept`)
- Receive or reply (`send & recv`)
- Terminate connection (`Close`)

Discovering Details of Your Local Network

- To find out interfaces: netstat -ni
- To find out routing table: netstat -rn
- To find out details of an interface: ifconfig
- To discover hosts on a LAN: ping

TCP/IP vs OSI

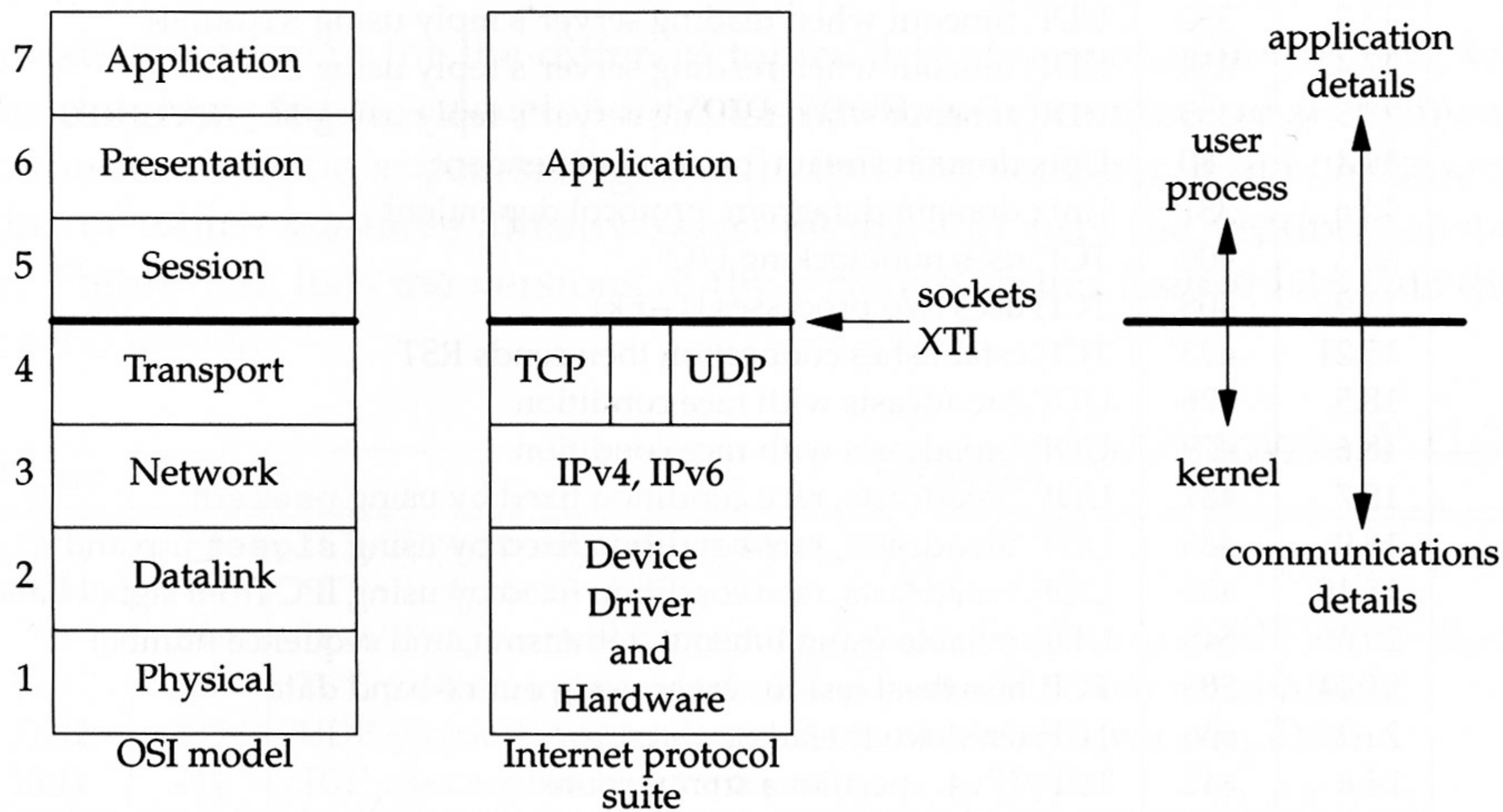


Figure 1.14 Layers in OSI model and Internet protocol suite.

TCP/IP Protocol Suite

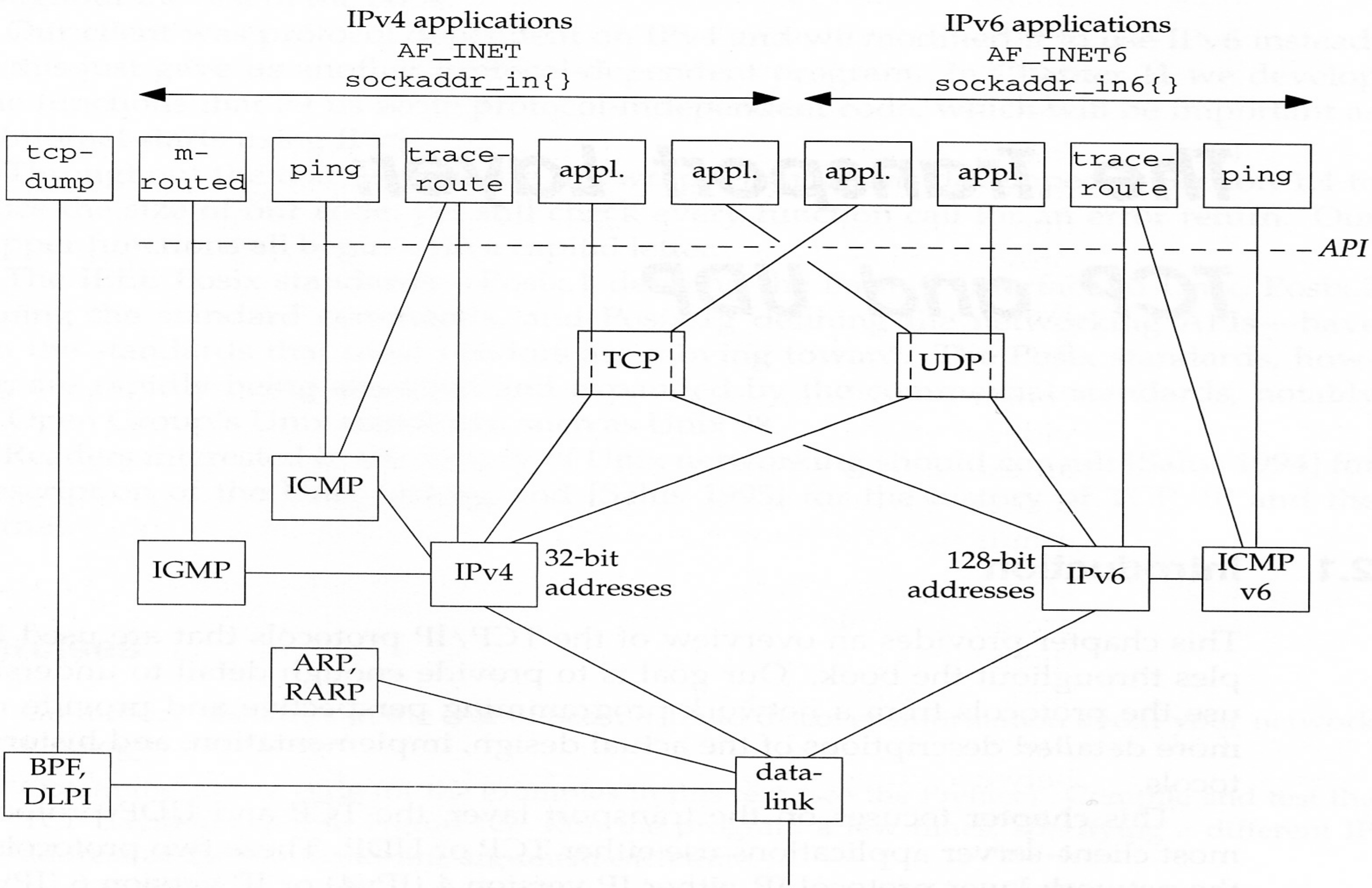
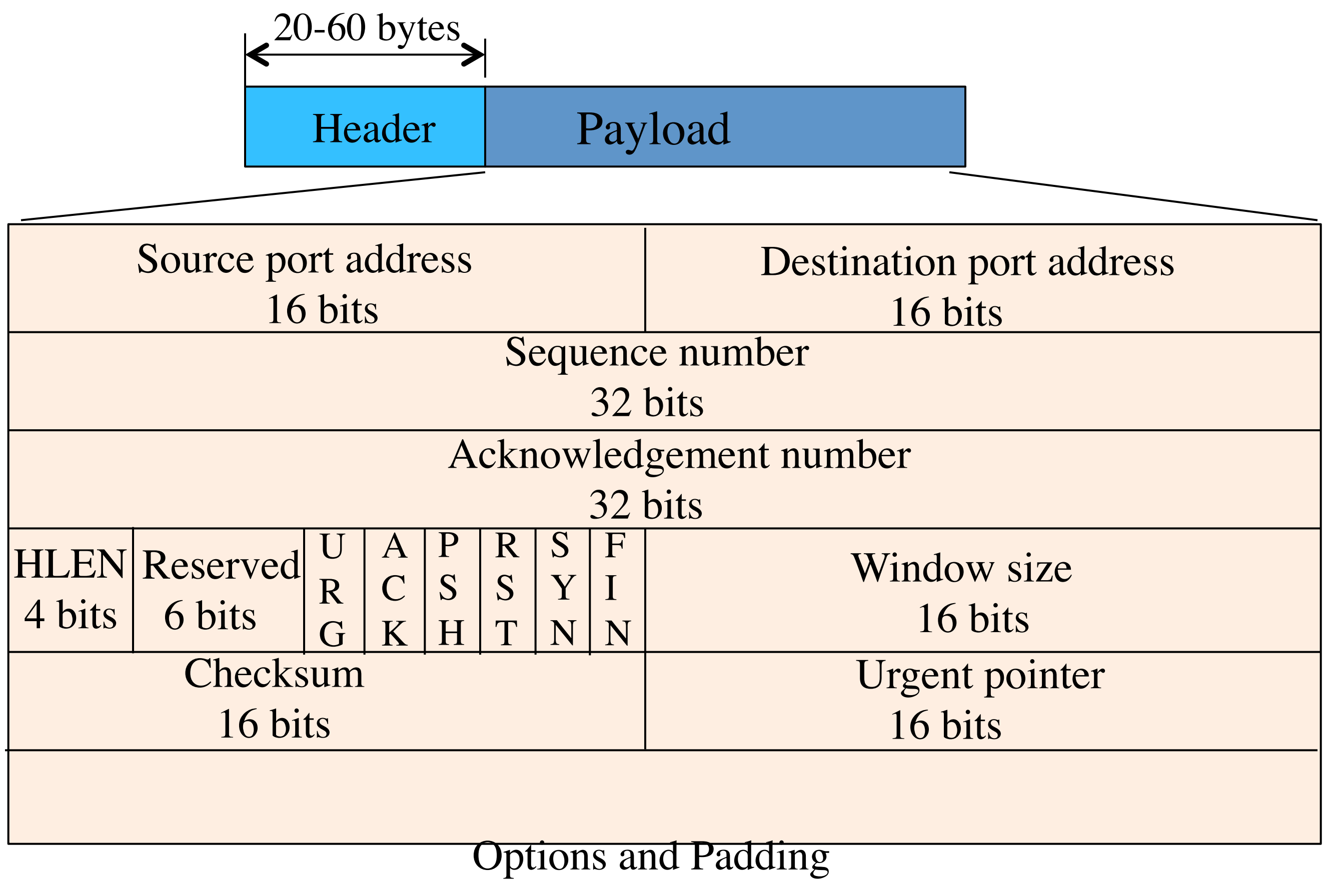


Figure 2.1 Overview of TCP/IP protocols.

TCP Segment

- A segment consists of a fixed 20- to 60-byte header, followed by zero or more data bytes



Connection establishment

- Three-way handshake

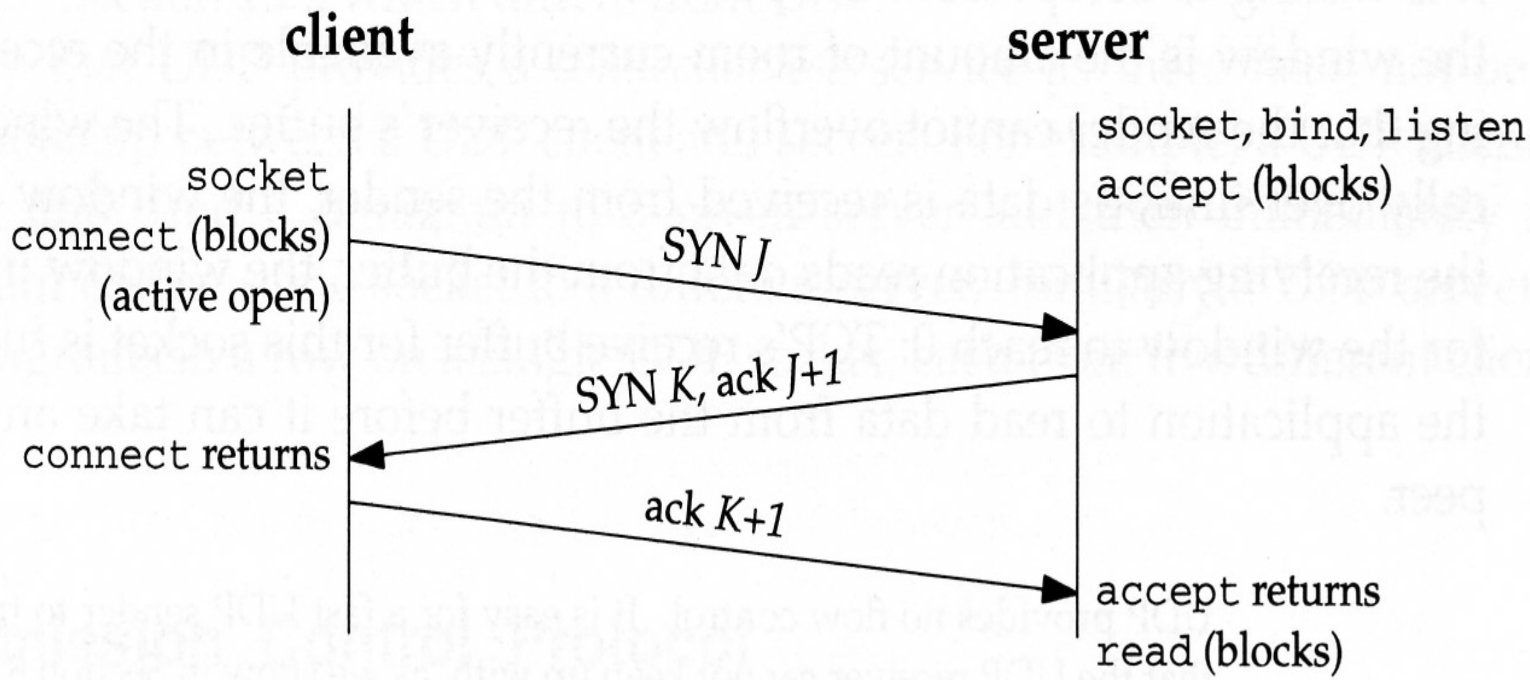


Figure 2.2 TCP three-way handshake.

TCP options

Each SYN message can carry TCP options.

- MSS option: Maximum Segment Size
  - With this option the TCP sending the SYN announces the maximum amount of data that it is willing to accept in each TCP segment
- Window scale option
  - The maximum window that either TCP can advertise to the other TCP is 65535 (16 bits for window size)
- Timestamp option
  - New option needed for high-speed connections to prevent possible data corruption caused by lost packets that then reappear. No worries for network programmers.

Connection termination

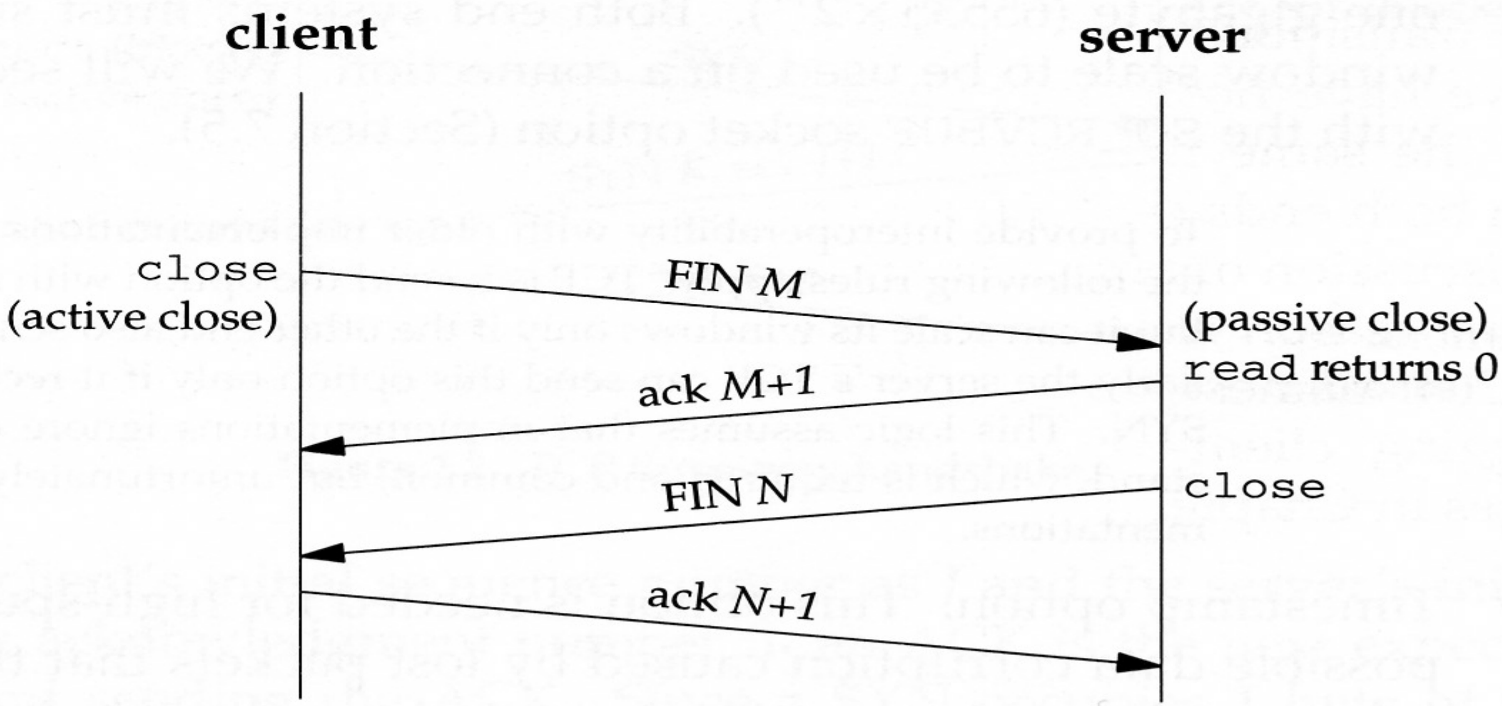


Figure 2.3 Packets exchanged when a TCP connection is closed.

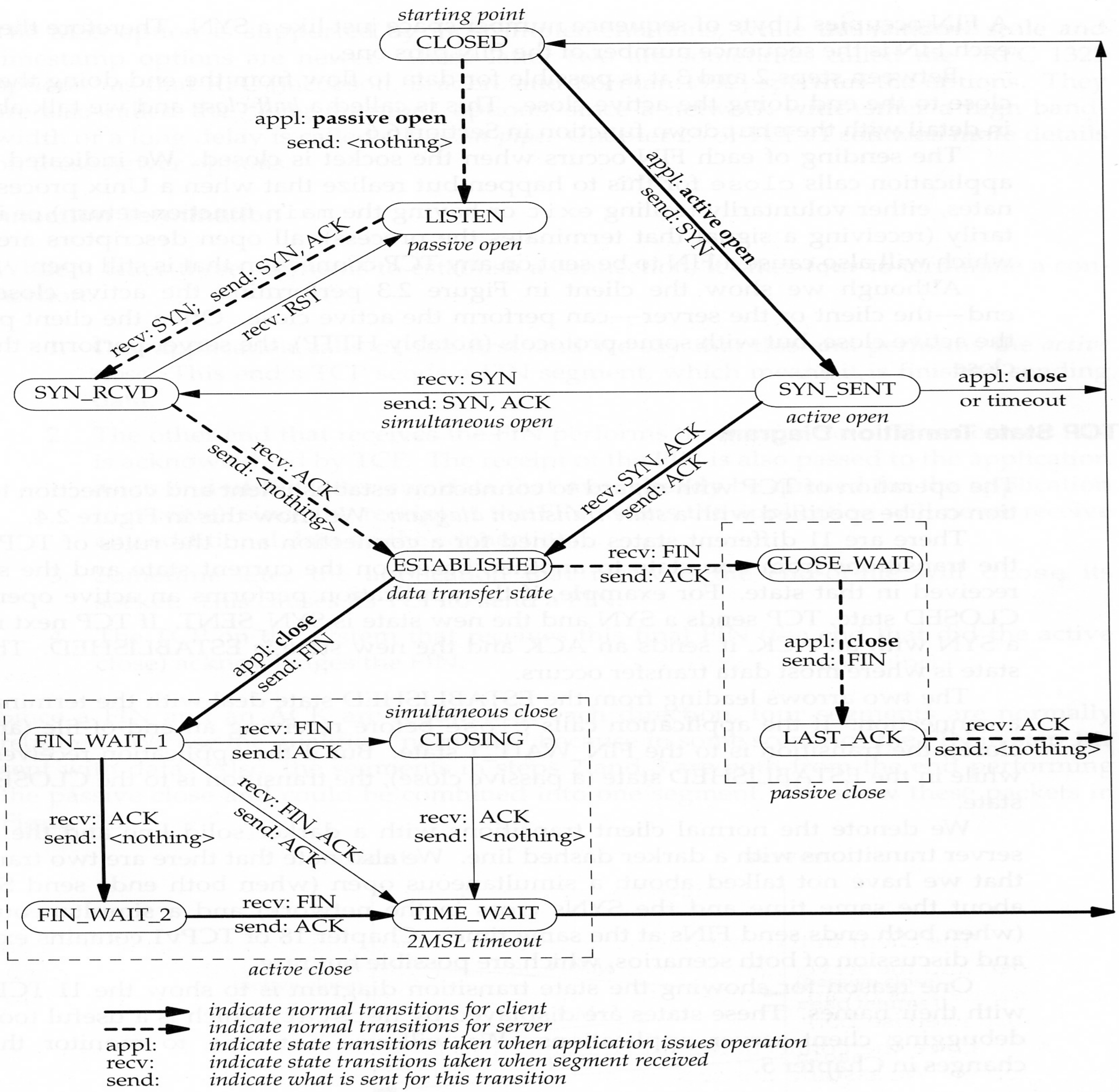


Figure 2.4 TCP state transition diagram.

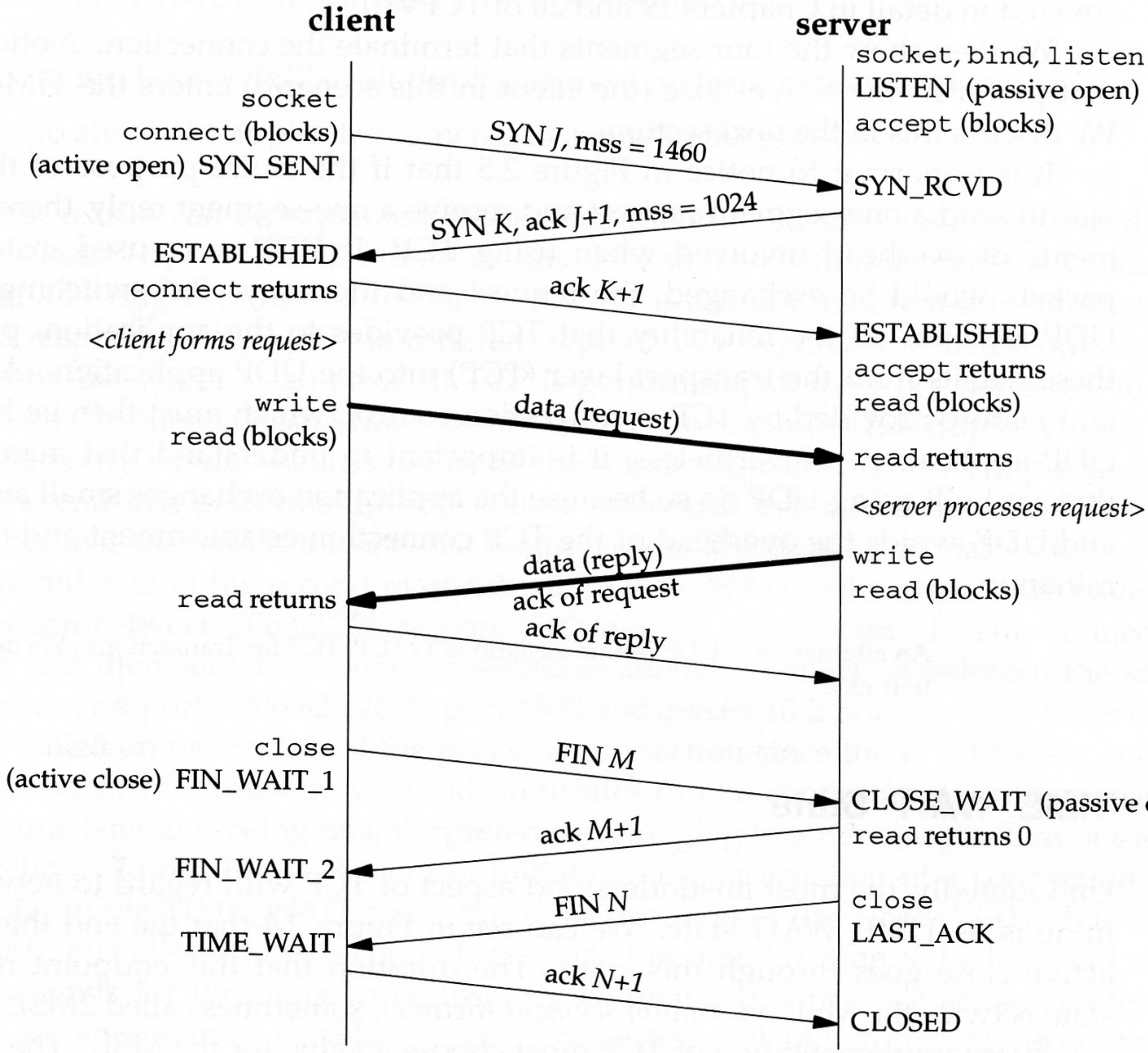


Figure 2.5 Packet exchange for TCP connection.

TIME_WAIT state

- Why need TIME_WAIT state?
  - To implement TCP's full-duplex connection termination reliably
  - To allow old duplicate segments to expire in the network
- The time to remain in this state is 2*MSL
  - MSL is Maximum Segment Lifetime (the maximum amount of time that any given IP datagram can live in an Internet)
  - The recommended value for MSL is 2 minutes in RFC 1122, though BSD used a value of 30 seconds
  - So the time for TIME_WAIT state is between 1 and 4 minutes

Port numbers

- Well-known ports
  - 0-1024
  - Controlled and assigned by IANA (Internet Assigned Number Authority)
- Registered ports
  - 1024-49151
  - Not controlled by IANA, but IANA registers and lists the uses of these ports as a convenience to the community
- Dynamic (or private) ports
  - 49152-65535, also called ephemeral ports
- Reserved (privileged) ports in Unix, 0-1024

Concurrent servers and port

- Socket pair
  - A 4-tuple for a TCP connection, which uniquely identifies the TCP connection
  - local IP address, local TCP port, foreign IP address, and foreign TCP port

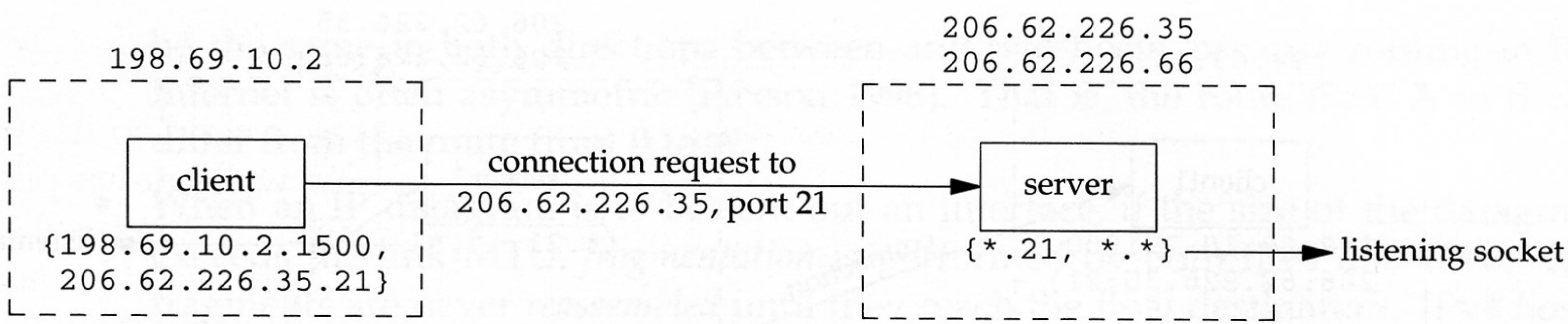


Figure 2.8 Connection request from client to server.

Concurrent servers and port (cont.)

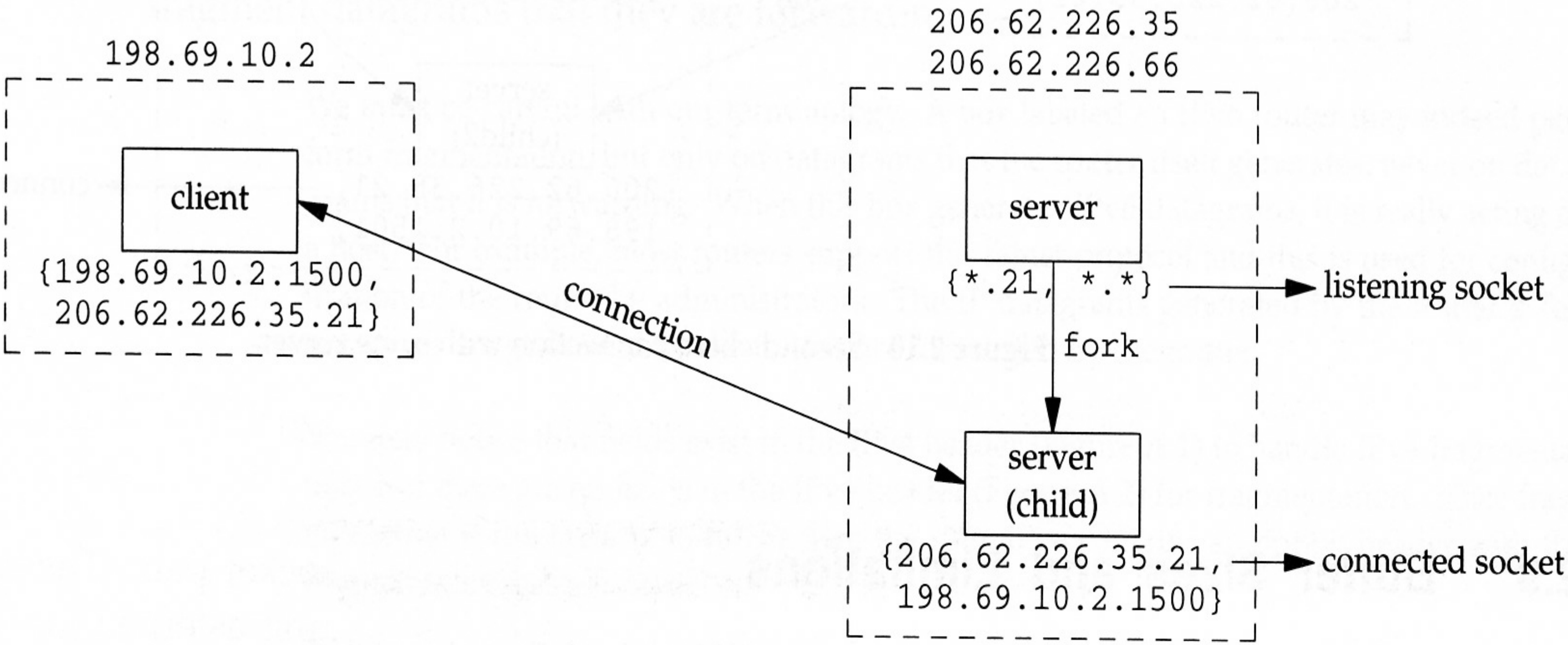


Figure 2.9 Concurrent server has child handle client.

Concurrent servers and port (cont.)

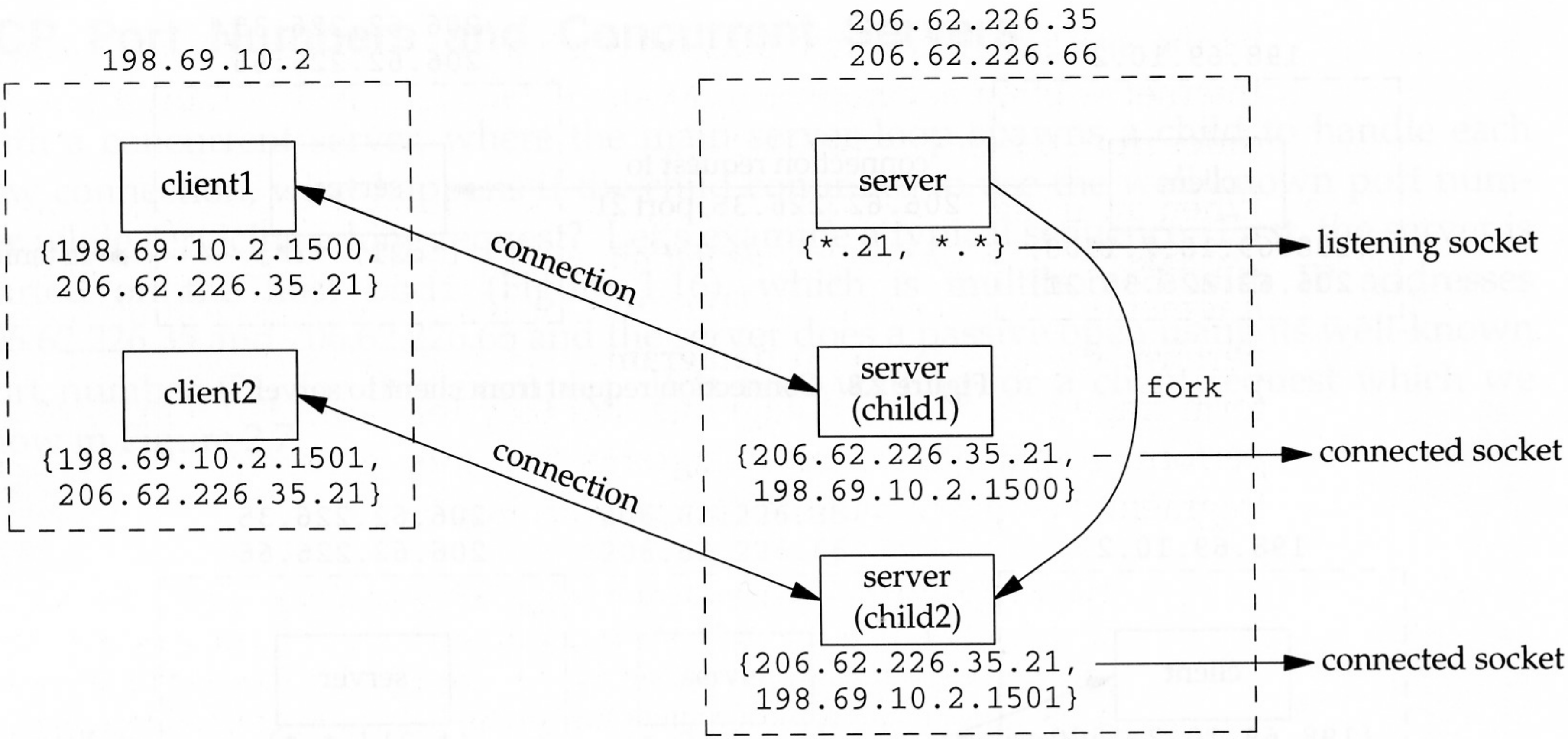
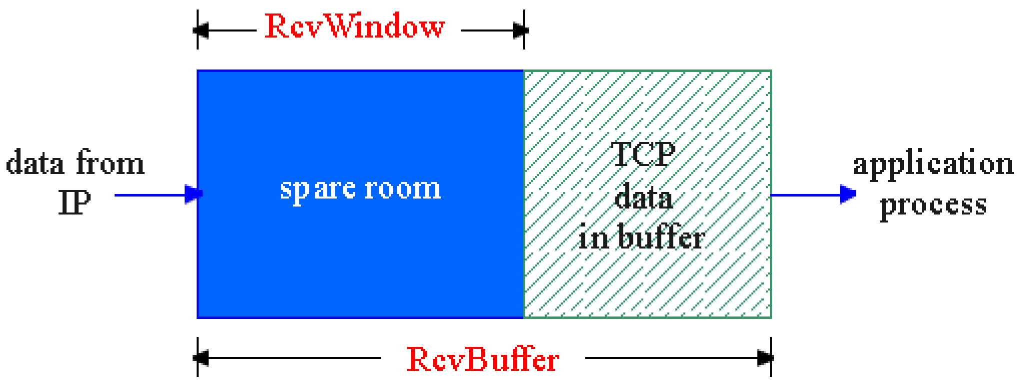


Figure 2.10 Second client connection with same server.

TCP Flow Control

- Receiving side of a TCP connection has a receive buffer.

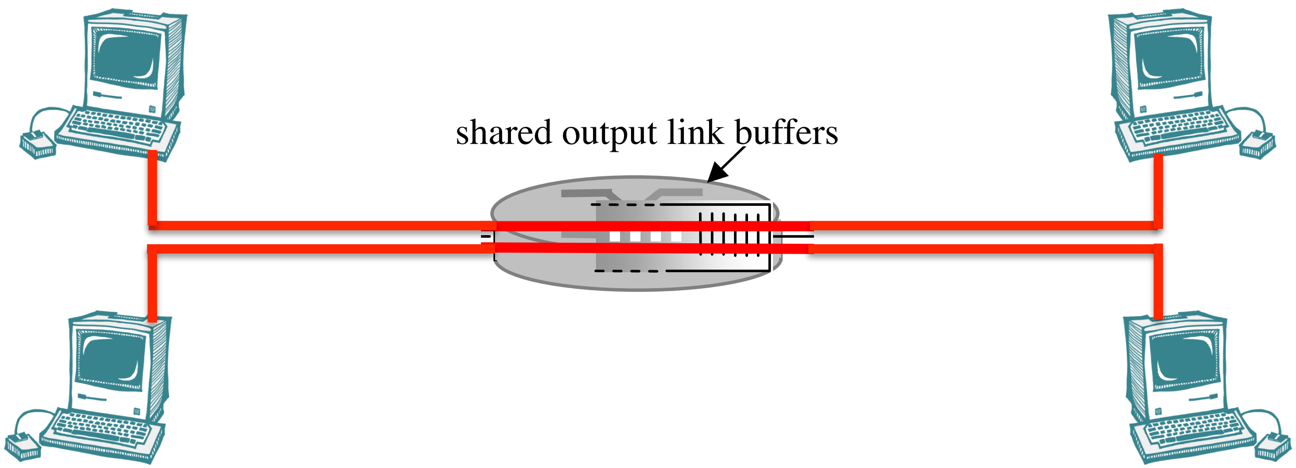


Sender won't overflow receiver's buffer by transmitting too much and too fast.

– Receiver advertises spare room via the "Window Size" field in the header of TCP segment.
– Sender keeps the unacknowledged data in case that retransmission is needed.

TCP Congestion Control

- Congestion: too many sources send too much data for network to handle



- Manifestations:
  - Lost packets (buffer overflow at routers)
  - Long delay (queuing in router buffers)

TCP Congestion Control

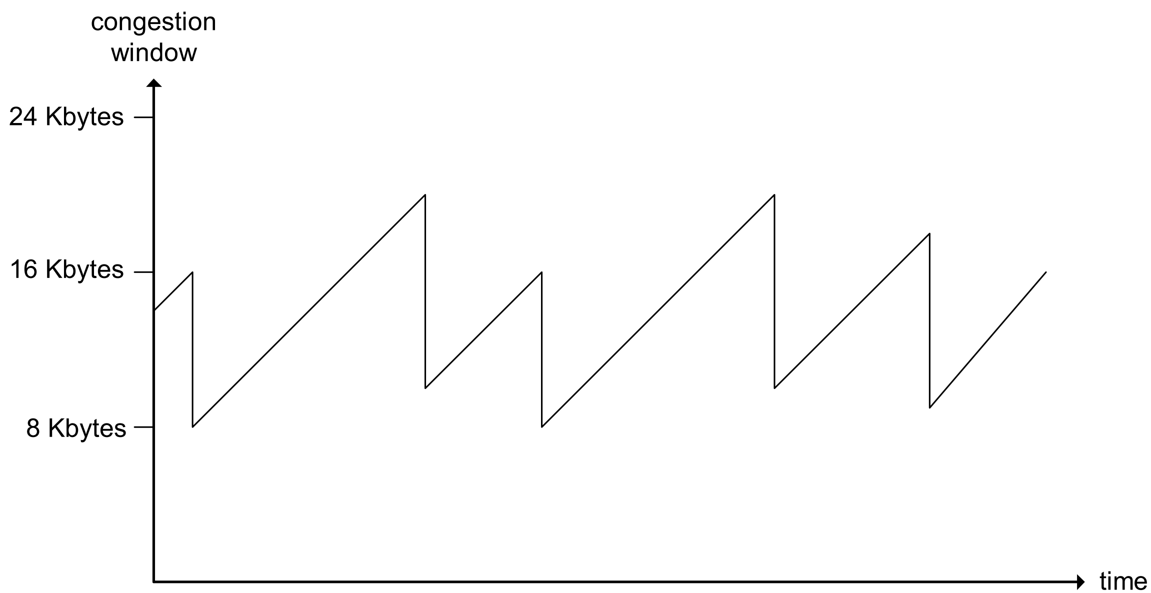
- End-to-end control
  - Congestion window at the sender
  - Sender limits transmission rate:

    LastByteSent – LastByteAcked <= Congestion Window

- Mechanisms
  - AIMD
  - Slow start
  - Refinement

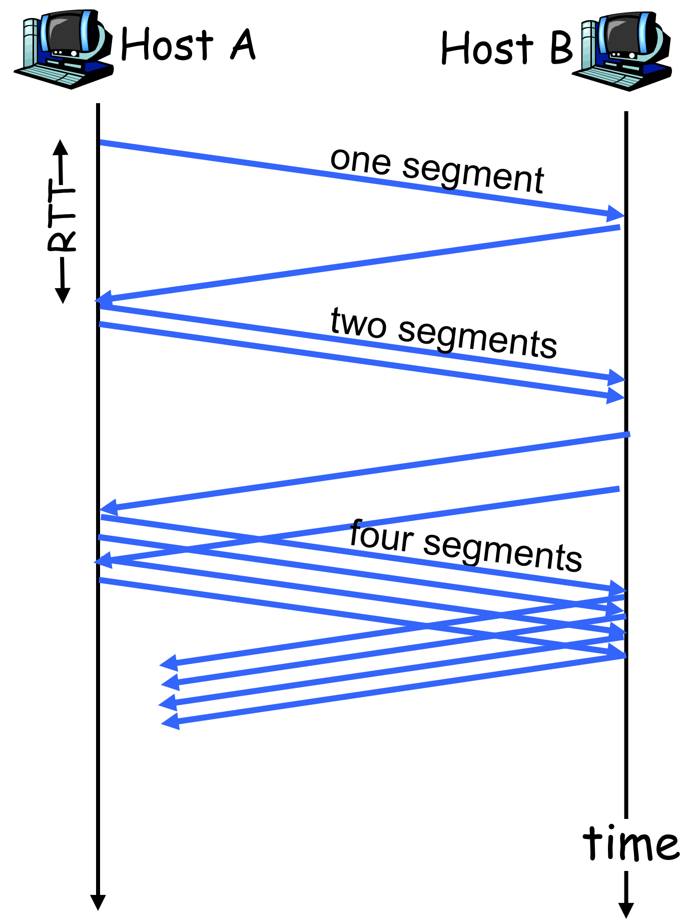
TCP AIMD

- Additive Increase
  - Increase congestion window by 1 MSS every RTT in the absence of loss
- Multiplicative Decrease
  - Cut congestion window in half after loss event



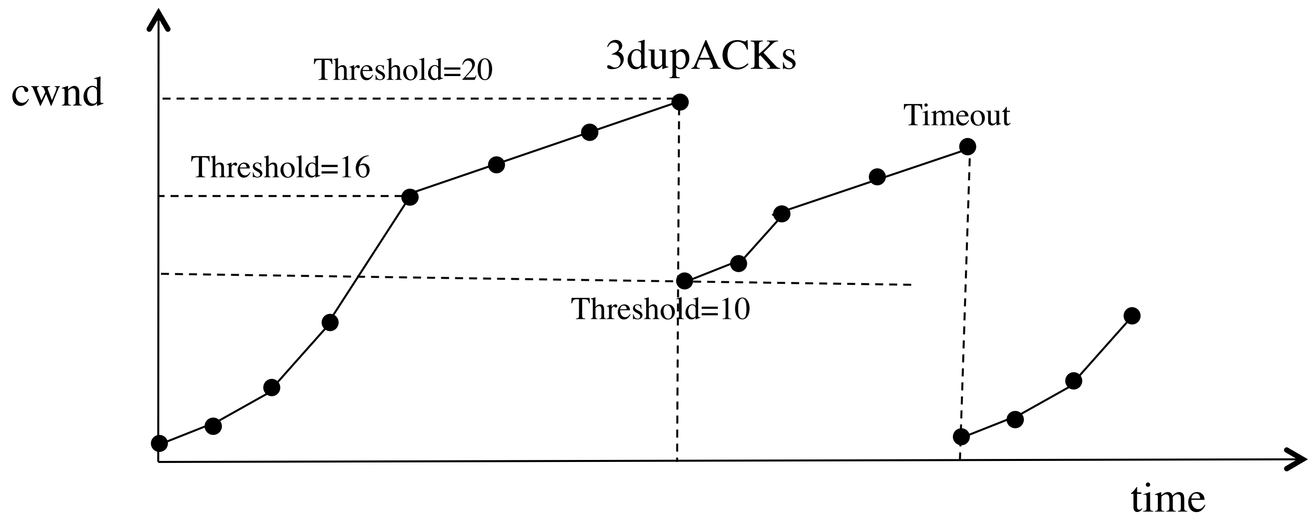
TCP Slow Start

- When connection begins, congestion window is set to 1 MSS.
- Double the congestion window every RTT if there is no loss event.
- Initial rate is slow but ramps up exponentially fast.



Refinement

- After 3 duplicated ACKs
  - Congestion window is cut in half
  - Window then grows linearly
- After timeout event
  - Congestion window is reset to 1 MSS
  - Slow start
  - Additive increase



Fast TCP

- Wei et al. Fast TCP: motivation, architecture, algorithms, performance, IEEE/ACM Transactions on Networking, 2006.
  - Use queueing delay as a congestion measure

$$\mathtt{w} \quad \longleftarrow \quad \min\left\{2\mathtt{w},\ (1-\gamma)\mathtt{w} + \gamma\left(\frac{\mathtt{baseRTT}}{\mathtt{RTT}}\mathtt{w} + \alpha\right)\right\}$$

    - baseRTT: the minimum RTT observed
    - α : a constant incremental factor

Summary

- The layered design approach for network protocols
- TCP connection setup and termination
  - Transition between different states
  - TIME_WAIT state
- Port numbers & socket
- TCP flow control and congestion control