Computer Networks - Overview

- Computer Network Terms
- Common functions and devices in computer networks
- Structure of computer networks
- Many examples with some detailed information

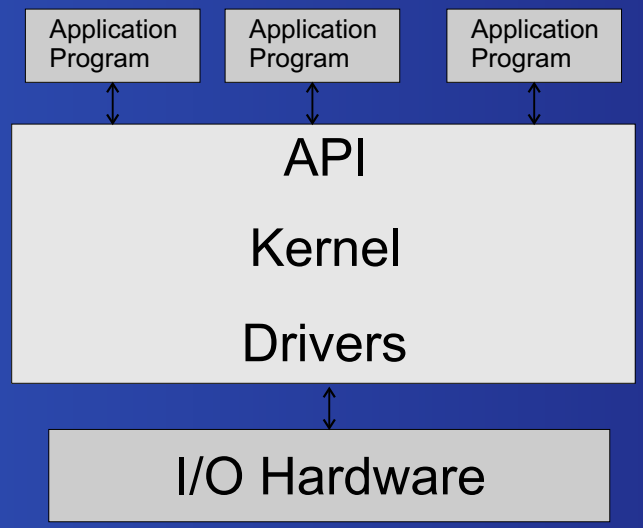
Computer Networks - The Plan

- now - spring break: HDLC, Ethernet (“Low-Level”)
- spring break - end: IP, TCP and applications (“High-Level”)

Operating Systems: Missing things

- API: Application Programming Interface
- Files, Filedescriptors

APIs



- Operating systems define an API
- Application programs communicate with the OS only by using the API

APIs

- APIs can be very different in size:
  - Win32: 2016 Functions
  - Linux: 190 System Calls (Linux 2.2)
  - Minix: 53 System Calls
- Some common UNIX system calls: read, write, open, close, fork, kill, sbrk
- System calls are initiated in the application program but executed by the operating system: context switch (implemented through trap)

Examples for (unix) API functions

- Memory management: malloc, free, sbrk
- I/O: open, close, read, write, fstat, ioctl
- Network: socket, bind, connect, shutdown, send, recv
- signals and process control: signal, kill, fork, execvp
- synchronisation: sema_post, sema_wait, sema_trywait

File Systems and APIs

- Almost all general purpose OS control disk devices through file systems
- Exceptions: Some database server systems access disk devices directly
- A file system is a method of assigning names to sequences of disk blocks and controlling access to them.
- Examples: Dos Fat32, Linux Ext3, ReiserFS

How does it work ?

```
1  h = open("thefile",O_RDWR|O_CREAT);
2  write(h,buffer,length_of_buffer);
3  close(h);
```

How does it really work ?

```
1  h = open("thefile",O_RDWR|O_CREAT);
2  if (h==-1) {printf("Error␣%d\n",errno);
3    perror("FileIO");exit(-1);}
4  n = write(h,buffer,length_of_buffer);
5  if (n==length_of_buffer)
6    printf("successful␣write␣of␣buffer\n");
7  if (n==0) printf("nothing␣written\n");
8  if (n==-1) { printf("Error␣%d\n",errno);
9    perror("FileIO"); }
10 n==close(h);
11 if (n==-1) { printf("Error␣%d\n",errno);
12   perror("FileIO");}
```

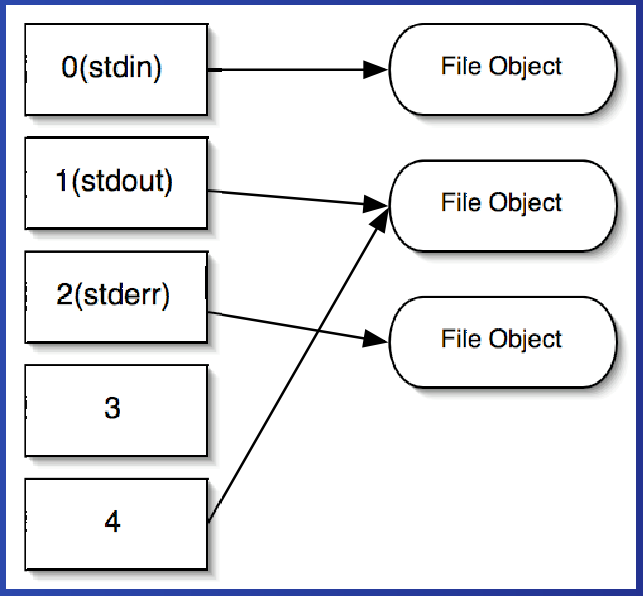
A look inside: Linux

```
1   struct task_struct {
2           volatile long state;    /* -1 unrunnable,
3                                      0 runnable, >0 stopped */
4           unsigned long flags;    /* per process flags,
5                                      defined below */
6           int sigpending;
7   ...
8   /* filesystem information */
9           struct fs_struct *fs;
10  /* open file information */
11          struct files_struct *files;
12  /* namespace */
13          struct namespace *namespace;
14  /* signal handlers */
15  ...
```

from linux 2.4.20 include/linux/sched.h

A look inside: Files of a process

Idea of the `files_struct` for a process...



A look inside: Linux

```
1  struct files_struct {
2          atomic_t count;
3          rwlock_t file_lock;
4          int max_fds;
5          int max_fdset;
6          int next_fd;
7          struct file ** fd;
8          fd_set *close_on_exec;
9          fd_set *open_fds;
10         fd_set close_on_exec_init;
11         fd_set open_fds_init;
12         struct file * fd_array[NR_OPEN_DEFAULT];
13 };
```

from linux 2.4.20 include/linux/sched.h

A look inside: Linux

```
1  struct file {
2          struct list_head         f_list;
3          struct dentry            *f_dentry;
4          struct vfsmount          *f_vfsmnt;
5          struct file_operations   *f_op;
6          atomic_t                 f_count;
7          unsigned int             f_flags;
8          mode_t                   f_mode;
9          loff_t                   f_pos;
10 ...
```

from linux 2.4.20 include/linux/fs.h

A look inside: Linux

```
1  struct file_operations {
2      struct module *owner;
3      loff_t (*llseek) (struct file *, loff_t, int);
4      ssize_t (*read) (struct file *, char *, size_t, loff_t *);
5      ssize_t (*write) (struct file *, const char *, size_t, loff_t *)
6      int (*readdir) (struct file *, void *, filldir_t);
7      unsigned int (*poll) (struct file *, struct poll_table_struct *)
8      int (*ioctl) (struct inode *, struct file *, unsigned int, unsig
9      int (*mmap) (struct file *, struct vm_area_struct *);
10     int (*open) (struct inode *, struct file *);
11     int (*flush) (struct file *);
12 ...
```

from linux 2.4.20 include/linux/fs.h

Conclusion

- Processes keep track of their open files
- Open files are kept in a numbered list
- File operations are abstract
- OS fills file operations with implementation

Network vs. File example: Coffee!

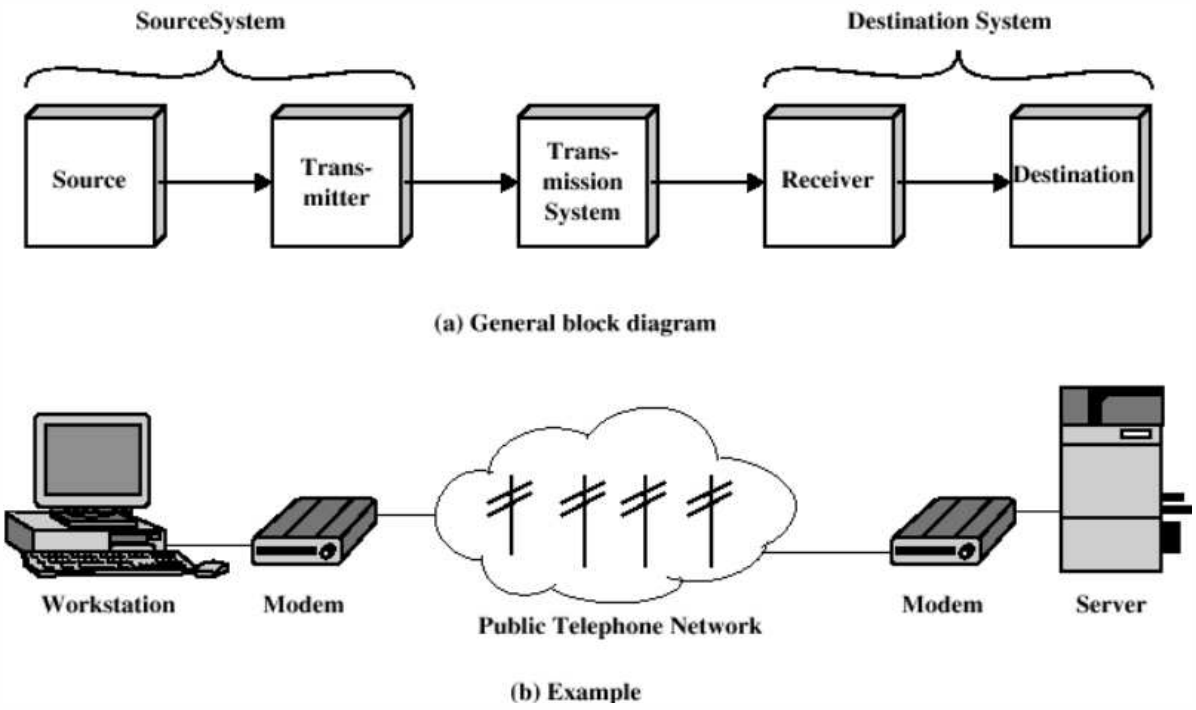
```
1  sprintf(buffer,"There's coffee available!");
2  sock=connectUDP("255.255.255.255","51966");
3  n=send(sock,buffer,strlen(buffer)+1,0);
4  close(sock);
```

```
1  h = open("thefile",O_RDWR|O_CREAT);
2  write(h,buffer,length_of_buffer);
3  close(h);
```

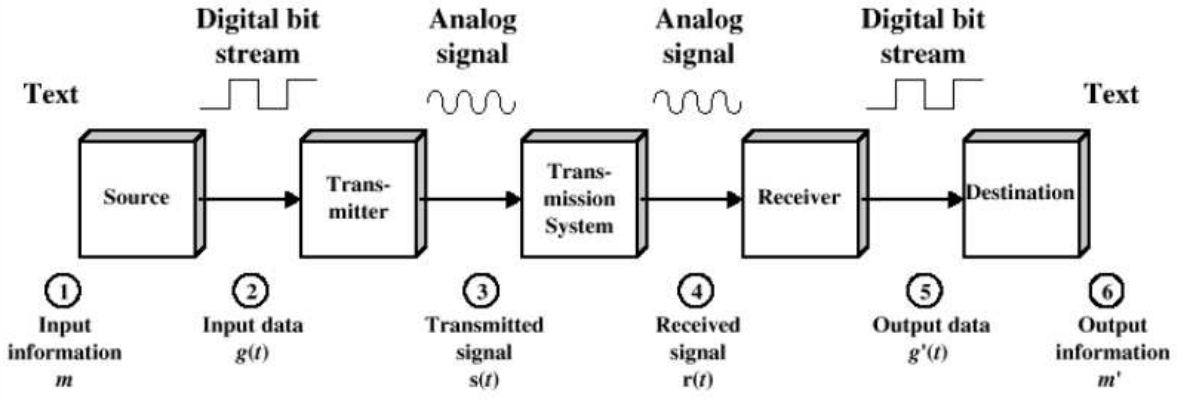
Network example: connectUDP

```
1  int connectUDP(char *host,char *service)
2  {
3  struct hostent  *phe;   /* pointer to host information entry    */
4  struct servent  *pse;   /* pointer to service information entry */
5  struct protoent *ppe;   /* pointer to protocol information entry*/
6  struct sockaddr_in sin; /* an Internet endpoint address         */
7  int     s, type;        /* socket descriptor and socket type    */
8  pse = getservbyname(service, "UDP"); sin.sin_port = pse->s_port;
9  phe = gethostbyname(host);
10 memcpy((char *)&sin.sin_addr,phe->h_addr, phe->h_length);
11 ppe = getprotobyname(protocol); type = SOCK_DGRAM;
12 s = socket(PF_INET, type, ppe->p_proto);
13 connect(s, (struct sockaddr *)&sin, sizeof(sin);
14 return(s);
```

Networks: An Overview



Networks: Communication Model



Network Types:WANs

- Traditional: Circut Switching and Packet Switching
- Frame Relay: less error control, higher data rate
- ATM: fixed cells

Network Types:LANs

- Token Ring
- Ethernet
- Wireless LAN

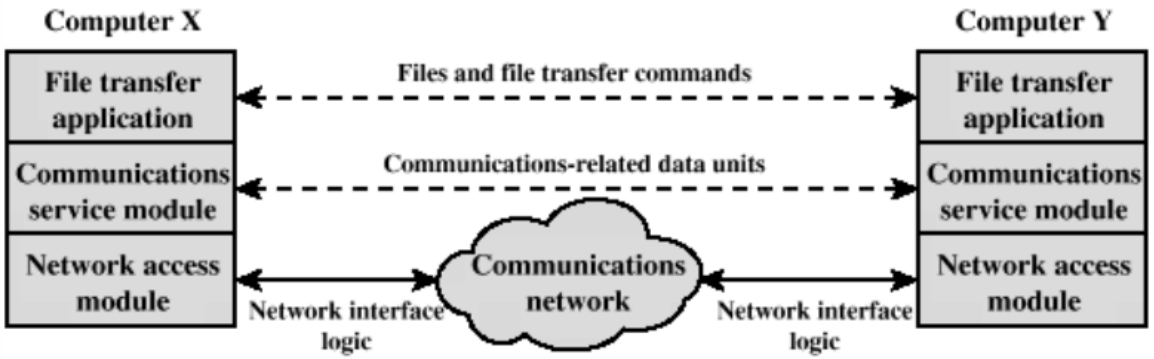
Network Protocols

Protocols define:

- Syntax: Data Format, Signal Levels
- Semantics: Control Information for coordination and error control
- Timings: Sequencing, Transmission Speed, Timeouts

A Protocol Architecture: Set of protocols used together in a modular architecture.

A simple protocol architecture



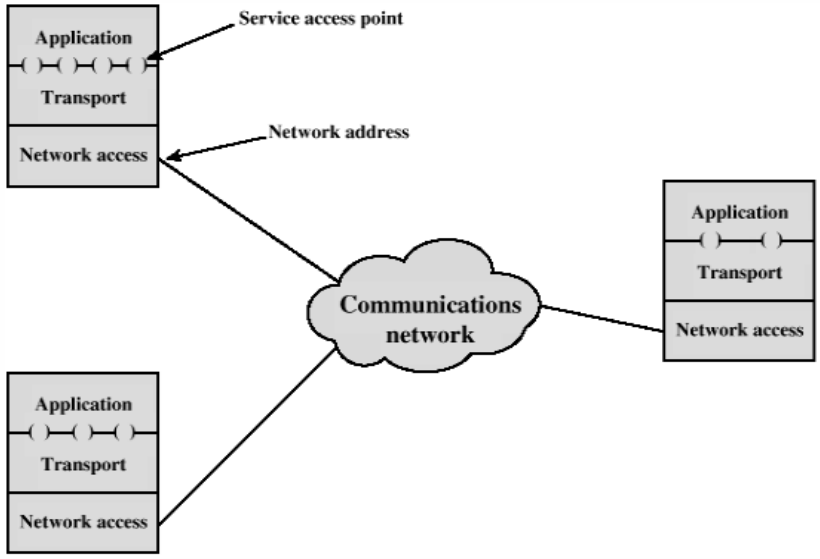
Network Layer Model

A Simple Model:

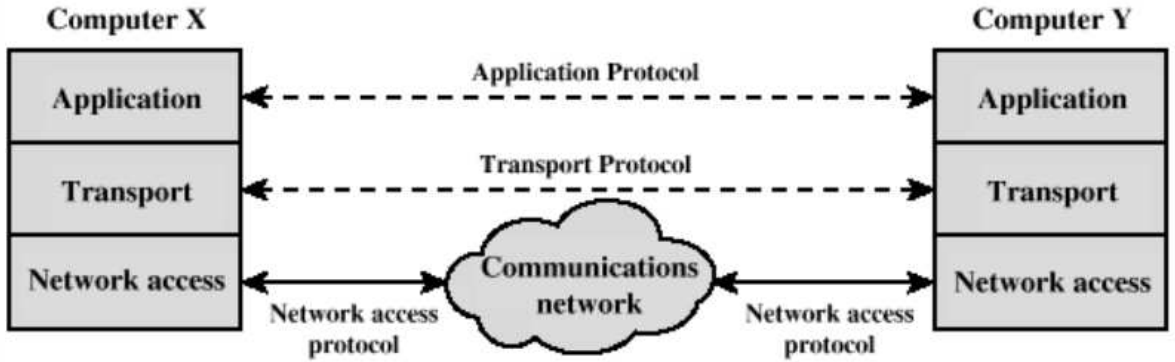
- Network Access Layer: Network (hardware) dependent, adressing
- Transport Layer: Reliable data exchange
- Application Layer: Application dependent (Service Access Points)

Each layer only communicates with adjacent layers

A simple protocol architecture



Protocols


Computer X
Computer Y
Application
Application Protocol
Application
Transport
Transport Protocol
Transport
Network access
Communications network
Network access
Network access protocol
Network access protocol

Constructing Packets

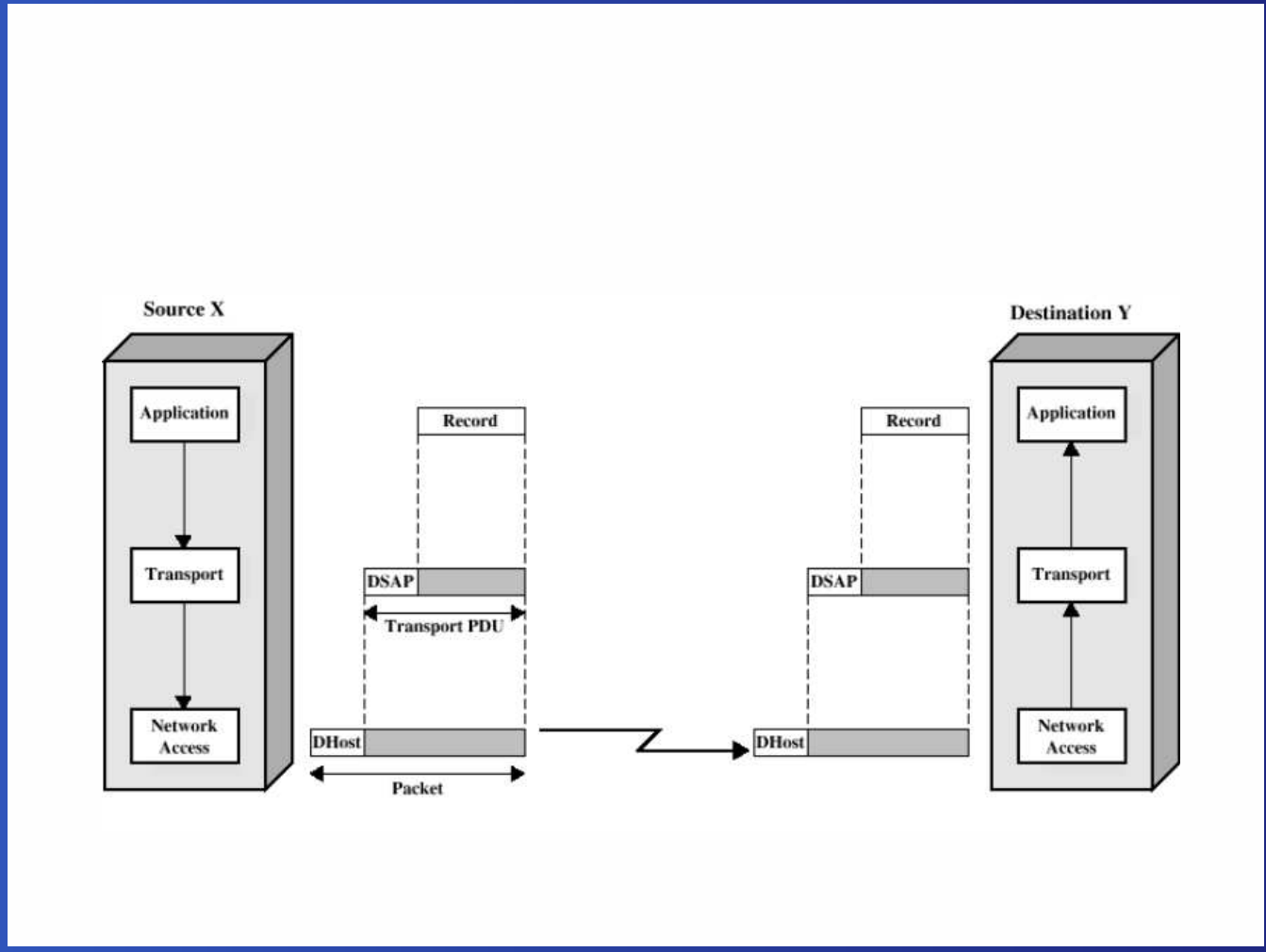
Protocol Data Units:

- Constructed from the data of the next higher layer and additional control information

Example:

- Transport PDU adds SAP identification, sequence number and error detection information to Application PDU.
- Network PDU adds adressing and optional priority information to Transport PDU

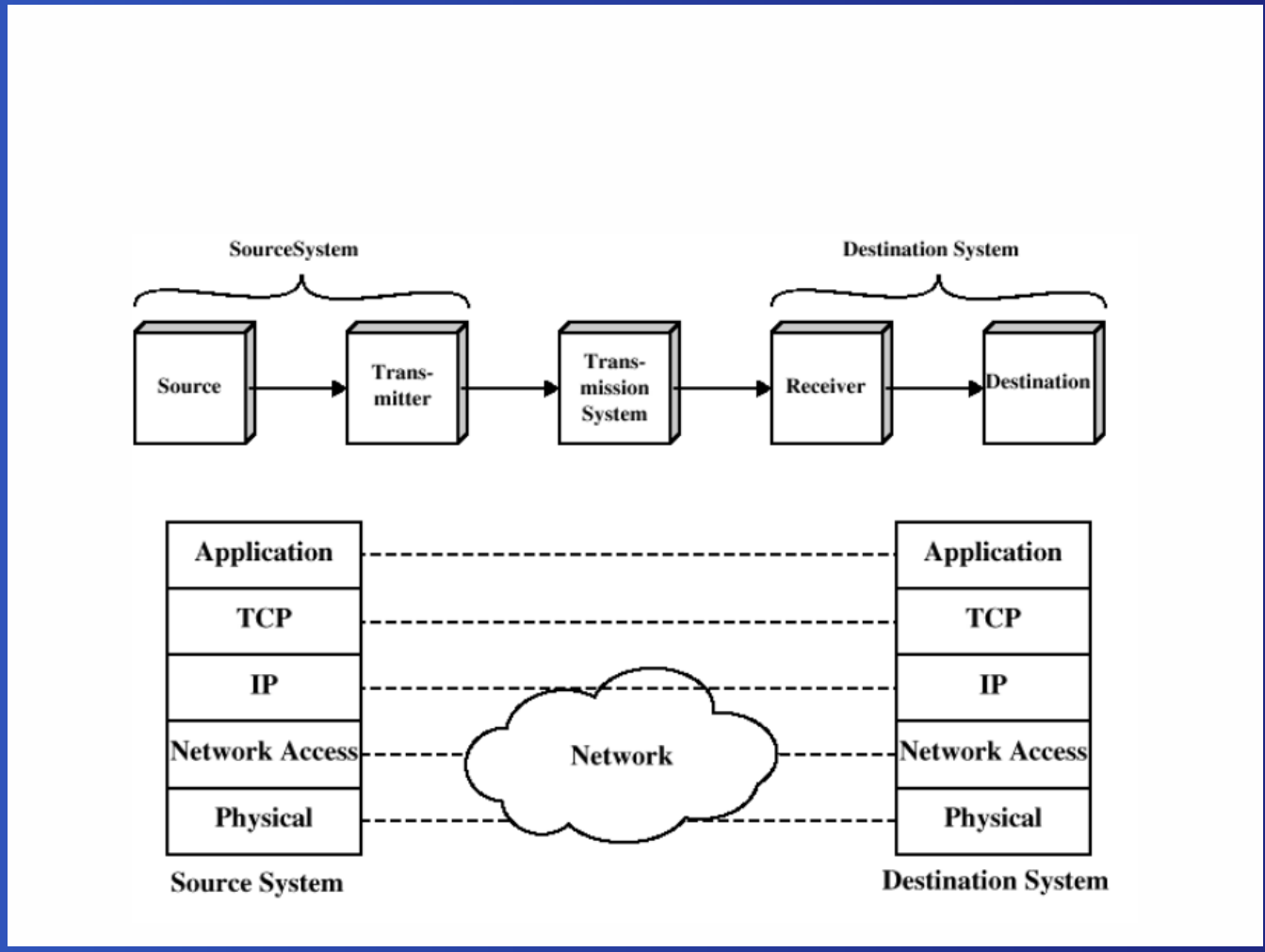
PDUs in the simple control architectur



TCP/IP Protocol Architecture

- 5 layer architecture: Application, Transport, Internet, Network, Physical
- TCP/IP accepts almost all Network and Physical Layers: PPP, GPRS, ATM, SONET, INMARSAT
- Application layer exists in multiple implementations: SYSV streams is one, sockets is another

TCP/IP Protocol Architecture



OSI Protocol Architecture

- Standardized by ISO
- 7 layer architecture: Application, Presentation, Session, Transport, Network, Data Link, Physical
- Connection-oriented
- Hardly implemented, classic example of a “failed” standardisation effort

OSI Protocol Architecture

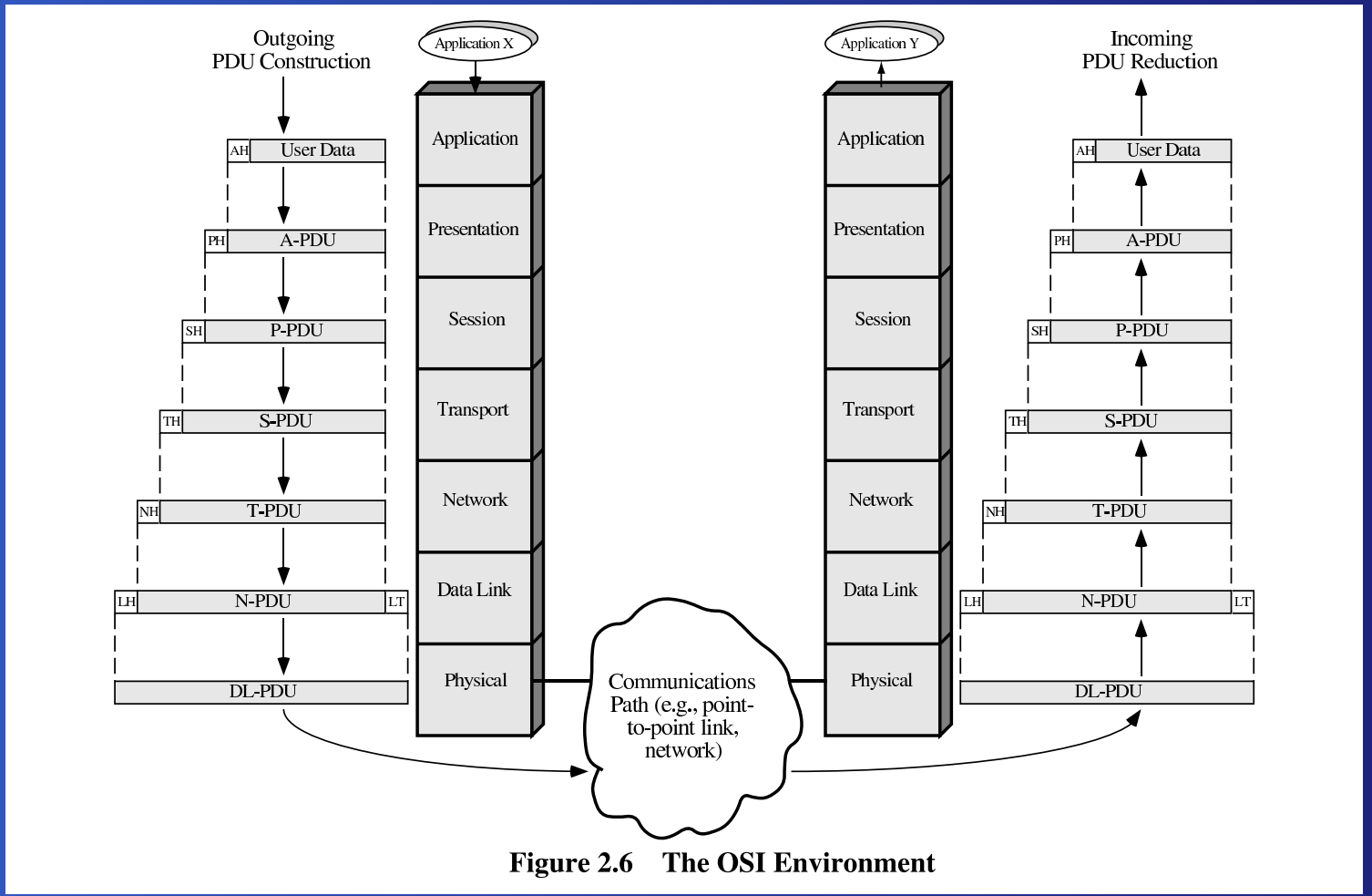
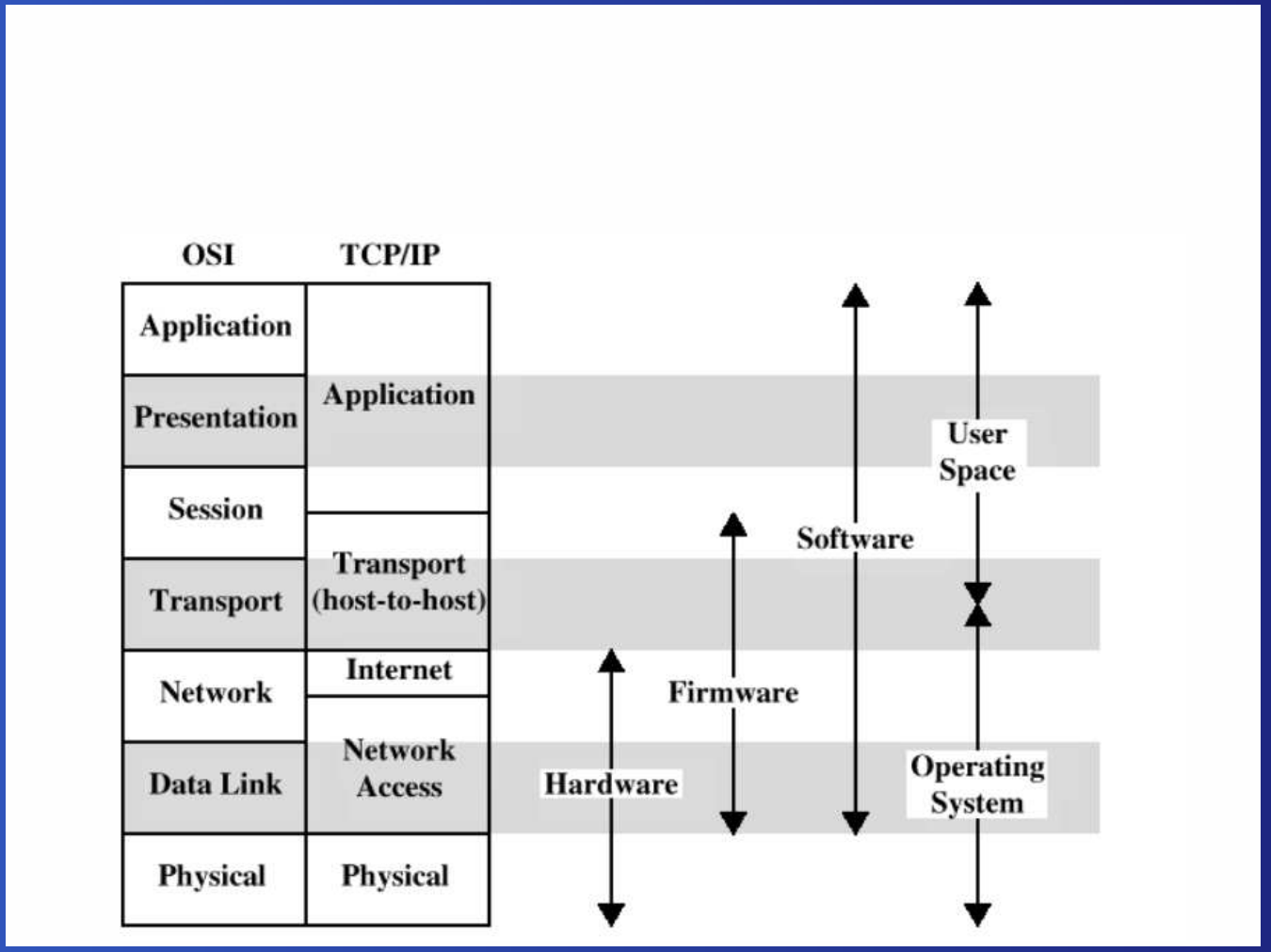


Figure 2.6 The OSI Environment

TCP/IP vs. OSI


OSI
TCP/IP
Application
Presentation
Session
Transport
Network
Data Link
Physical
Application
Transport (host-to-host)
Internet
Network Access
Physical
Hardware
Firmware
Software
User Space
Operating System

Standards

- Standards Organizations: Internet Society, ISO, ITU-T, IEEE , ATM Forum,. . .
- Standardization Process
- Standardization is a form of business war
- Standards always come too late
- Often, De-facto standards are “promoted”. Example: Ethernet $\longrightarrow$ IEEE 802.x

Internet Standards

- Responsibility: Internet Society
  - Internet Architecture Board (IAB): Defines overall architecture
  - Internet Engineering Task Force (IETF): Develops new protocols
  - Internet Engineering Steering Group (IESG): Manages standardization process
  - Internet Corporation for Assigned Names and Numbers (ICANN): Maintains operational information

Internet Standards: Process

- IETF Internet Draft: Review period for 6 months
- Request For Comment: Approved by IESG, Internet standard
- Examples:
  - RFC 821: Simple Mail Transfer Protocol (SMTP)
  - RFC 2616: Hypertext Transfer Protocol (HTTP)
  - RFC 2026: Standardization Process

Internet Standards: Process Flow

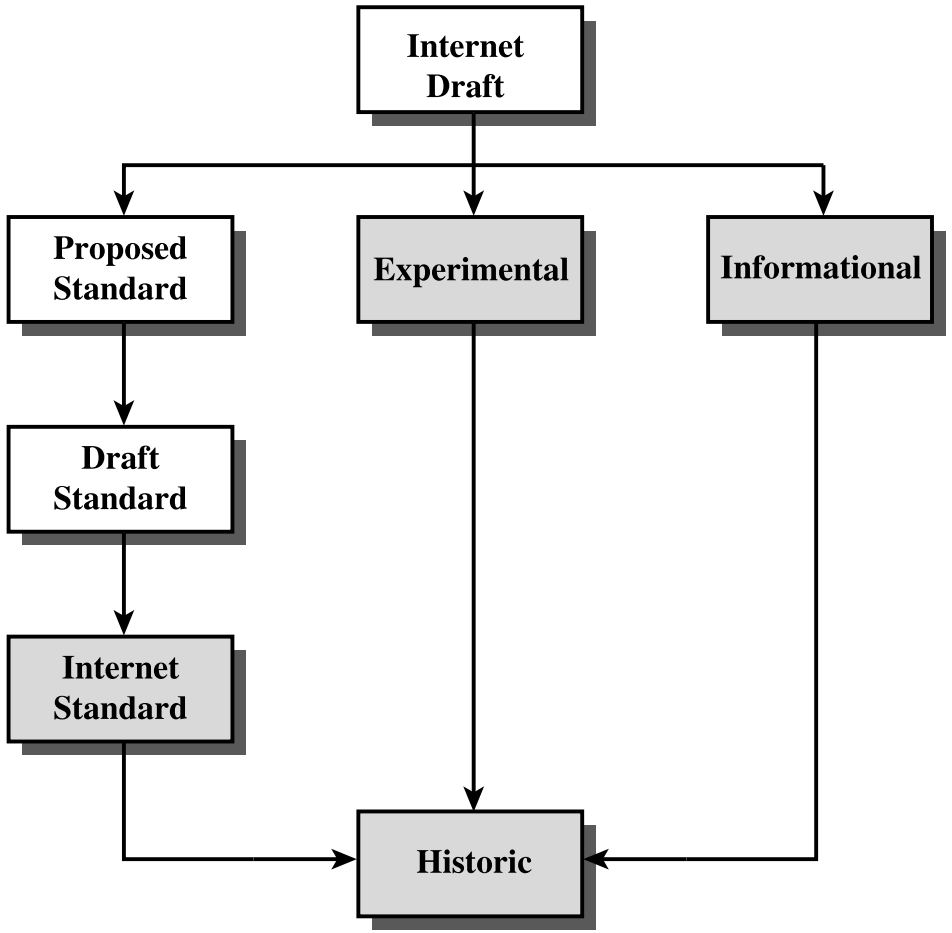


Figure 1.12 Internet RFC Publication Process

Physical Layer: Transmission

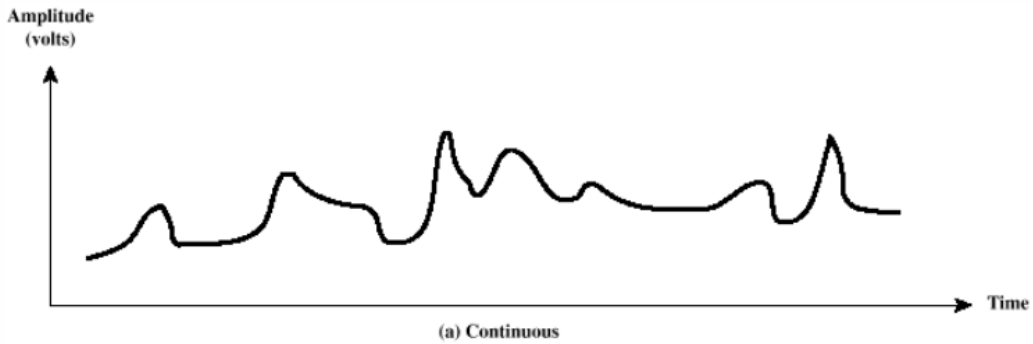
(for computer scientists only. See Adv. EE for the real thing.)

- Terminology
- Data Transmission
- Problems

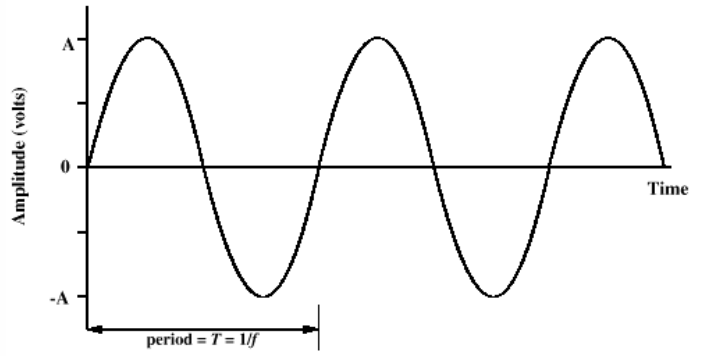
Terminology

- guided media vs. unguided media
- point-to-point vs. multipoint
- simplex, half-duplex, full-duplex

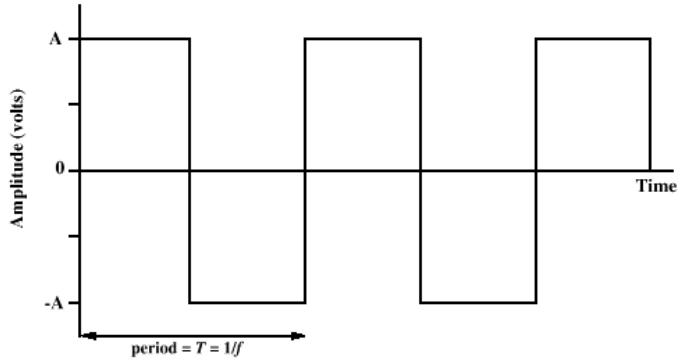
Discrete vs. continuous signals



Periodic signals



(a) Sine wave

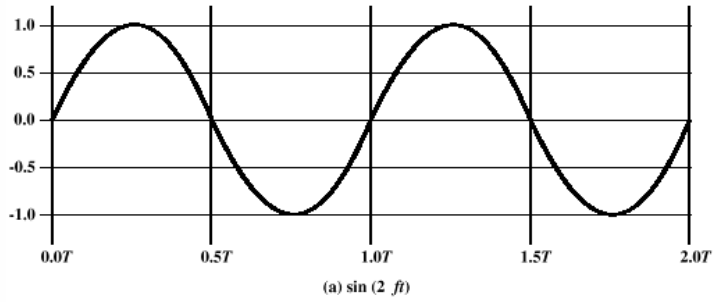


(b) Square wave

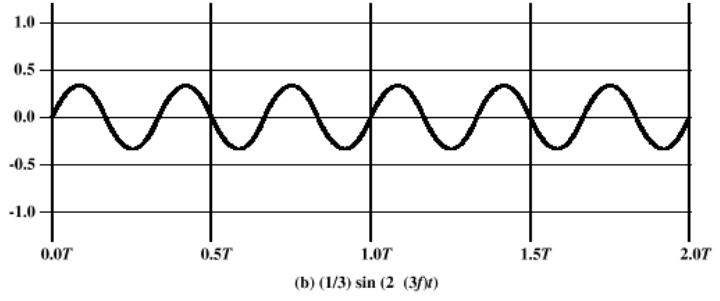
Frequency Domain

- Signals can be constructed by adding signals of multiple frequencies.
- Example:
  $s(t) = \frac{4}{\pi}\left(\sin(2\pi f t) + \frac{1}{3}\sin(2\pi(3f)t)\right)$

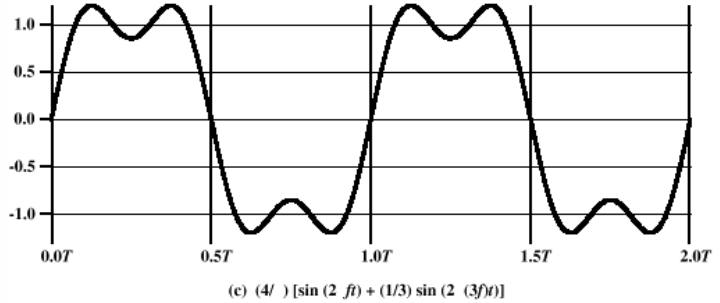
Addition of signals



(a) sin (2 *ft*)

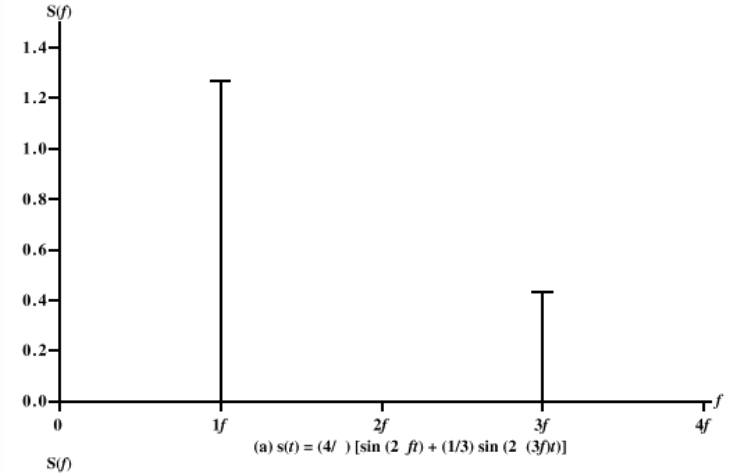


(b) (1/3) sin (2 (3*f*)*t*)



(c) (4/) [sin (2 *ft*) + (1/3) sin (2 (3*f*)*t*)]

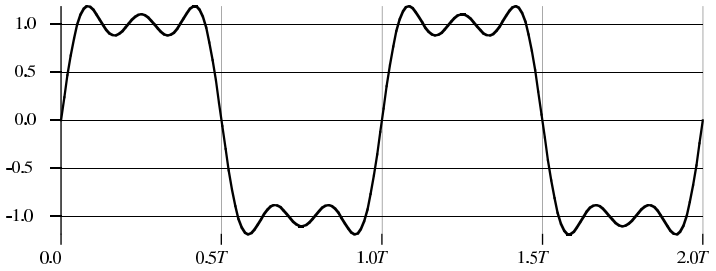
Frequency Domain Representations



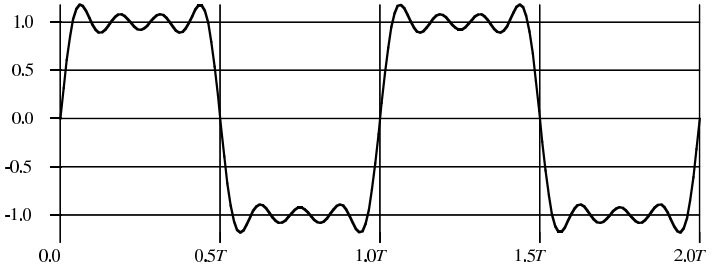
Terminology

- amplitude, frequency, period, phase, wavelength
- fundamental frequency, spectrum
- absolute bandwidth, effective bandwidth
- dc component

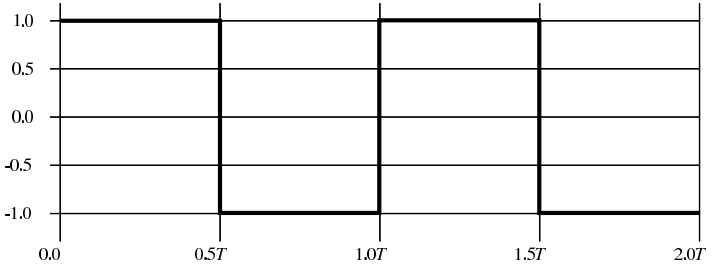
Bandwidth-limited square wave



(a) (4/) [sin (2 ft) + (1/3) sin (2 (3f)t) + (1/5) sin (2 (5f)t)]



(b) (4/) [sin (2 ft) + (1/3) sin (2 (3f)t) + (1/5) sin (2 (5f)t) + (1/7) sin (2 (7f)t)]



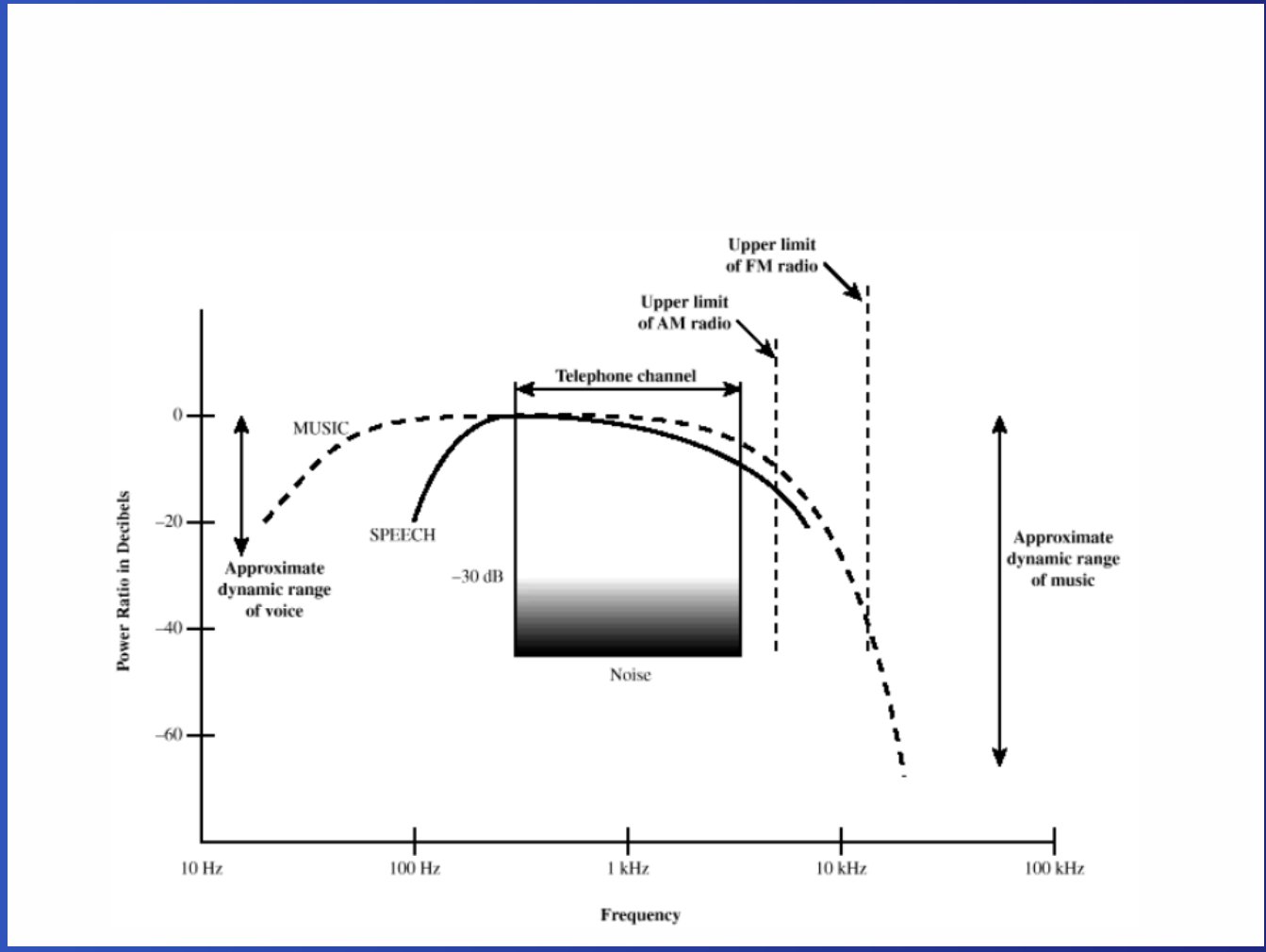
(c) (4/) $\sum$ (1/k) sin (2 (kf)t)

Figure 3.7 Frequency Components of Square Wave ($T = 1/f$)

Data Transmission

- Data: Thing to be transmitted
- Signal: Physical propagation
- Transmission: Sending data by signals
- Analog: Audio, Video
- Digital: text (ASCII/IRA)
- Analog over digital communication system: CODEC
- Digital over analog communication system: Modem

Audio Spectrum



Transmission Problems

- Attenuation

  $N = -10 \log_{10} \frac{P_{out}}{P_{in}}$
- Delay Distortion: frequency-dependent phase shift
- Noise: Thermal, intermodulation, crosstalk, impulse
- Channel capacity:
  - 4 concepts: Data rate, Bandwidth, Noise, Error rate
  - Goal: try to relate these concepts

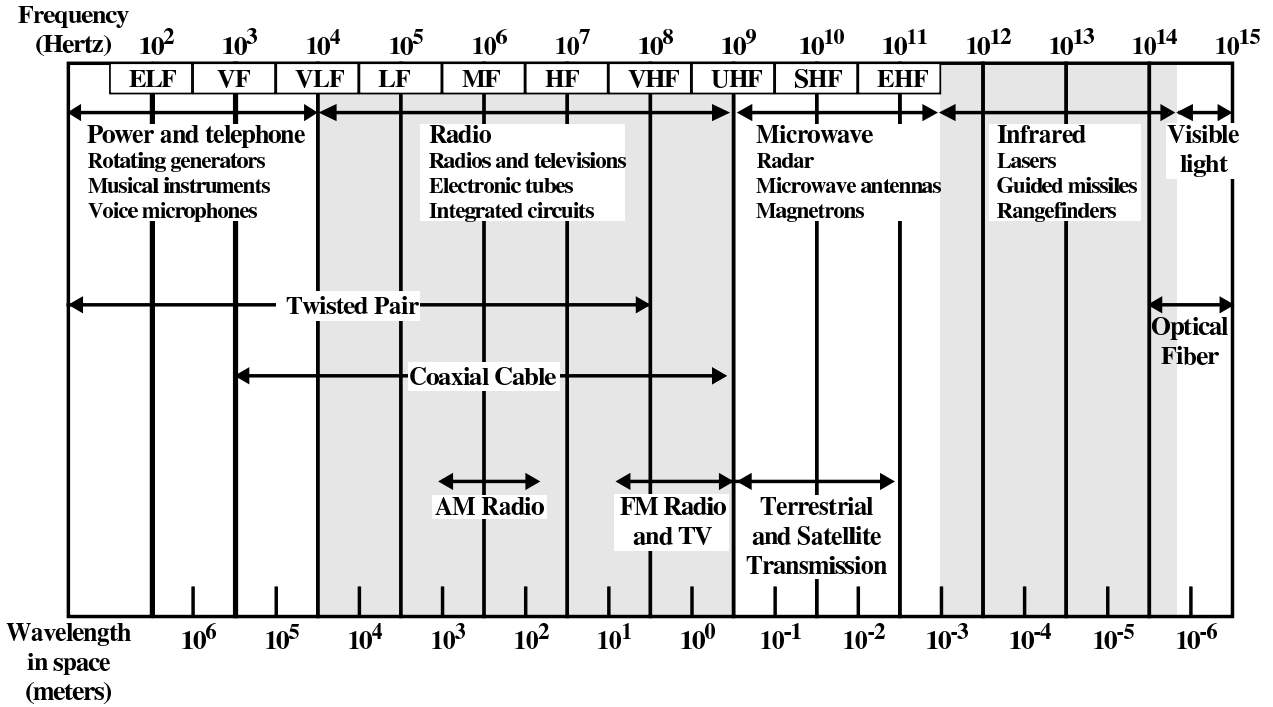
Nyquist Bandwidth

- AKA sampling theorem: With $2f$ sampling frequency, signals up to f can be reconstructed.
- Assumption: Noise-free channel
- Idea: Use the theorem "backwards": Either change signal or not $\leadsto$ two symbols per wavelength $\leadsto$ $2B$ symbols can be transmitted in B bandwidth.
- By using more than two symbols (e.g. voltages), the bandwith can be furter extended: $C = 2B \log_2 M$

Shannon Capacity

- Problem: Nyquist's assumption is a noise-free channel
- With noise present in the channel, "faster" transmission leads to "smaller" bits that can be "damaged" by noise more easily.
- Signal-to-Noise Ratio:
  $$(SNR)_{db} = 10 \log_{10} \frac{\text{signal power}}{\text{noise power}}$$
- Shannon: $C = B \log_2(1 + SNR)$

Electromagnetic Spectrum



ELF = Extremely low frequency
VF = Voice frequency
VLF = Very low frequency
LF = Low frequency
MF = Medium frequency
HF = High frequency
VHF = Very high frequency
UHF = Ultrahigh frequency
SHF = Superhigh frequency
EHF = Extremely high frequency

Figure 4.1 Electromagnetic Spectrum for Telecommunications