Computer Networks - 2021

Mohammed El-Hajj

Jacobs University Bremen

Sept 3, 2021

Computer Networks and Distributed Systems

• Introduction to Computer Science	1st Semester
• Programming in C I	1st Semester
• Algorithms and Data Structures	2nd Semester
• Programming in C II	2nd Semester
• Computer Architecture and Programming Languages	3rd Semester
• Operating Systems	3rd Semester
• Computer Networks	4th Semester
• Distributed Algorithms	6th Semester
• Project and Bachelor Thesis	5th/6th Semester

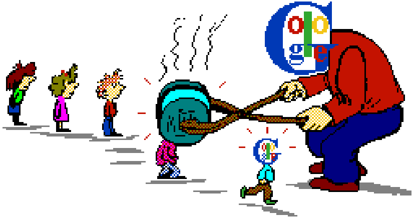
Credits

The following slides are the adaptation of:

- Lecture Slides of Prof [Jürgen Schönwälder] – Computer Networks 2019 – Jacobs University

Course Objectives

- Introduce fundamental data networking concepts
- Focus on widely deployed Internet protocols
- Prepare students for further studies in networking
- Combine theory with practical experiences
- Raise awareness of weaknesses of the Internet



Course Content

1. Introduction
2. Fundamental Networking Concepts
3. Local Area Networks (IEEE 802)
4. Internet Network Layer (IPv4, IPv6)
5. Internet Routing (RIP, OSPF, BGP)
6. Internet Transport Layer (UDP, TCP)
7. Firewalls and Network Address Translators
8. Domain Name System (DNS)
9. Abstract Syntax Notation 1 (ASN.1)
10. External Data Representation (XDR)
11. Augmented Backus Naur Form (ABNF)
12. Electronic Mail (SMTP, IMAP)
13. Document Access and Transfer (HTTP, FTP)

Reading Material

- A.S. Tanenbaum, ”Computer Networks”, 4th Edition, Prentice Hall, 2002
- W. Stallings, ”Data and Computer Communications”, 6th Edition, Prentice Hall, 2000
- C. Huitema, ”Routing in the Internet”, 2nd Edition, Prentice Hall, 1999
- D. Comer, ”Internetworking with TCP/IP Volume 1: Principles Protocols, and Architecture”, 4th Edition, Prentice Hall, 2000
- J.F. Kurose, K.W. Ross, ”Computer Networking: A Top-Down Approach Featuring the Internet”, 3rd Edition, Addison-Wesley 2004.

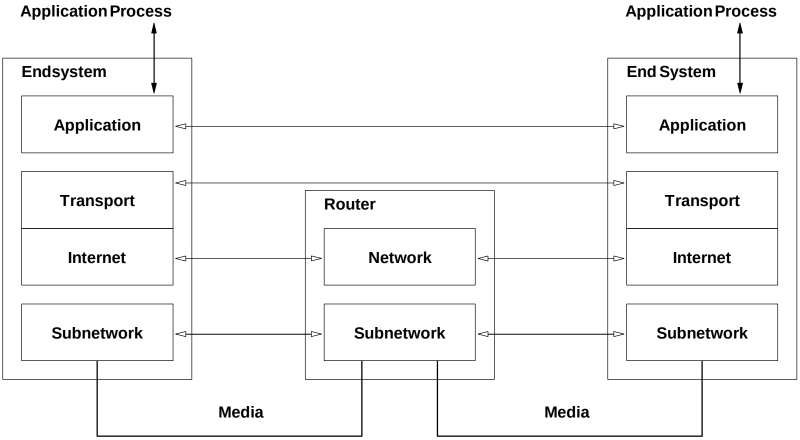
Grading

- Final examination (100 %)
  - Covers the whole lecture
  - Closed book (and closed computers / networks)
- Quizzes (bonus) (5%)
  - Control your continued learning success
  - 3 quizzes with 10 pts each
  - 50 pts and above equals 30% of the overall grade
- Assignments(bonus) (5%)
  - Learning by solving assignments
  - Implement network protocols
  - Gain some practical experience in a lab session
  - 2 assignments with 10 pts each
  - 50 pts and above equals 30% of the overall grade

Part 1: Introduction

Section 1: Internet Concepts and Design Principles

Internet Model



Internet Design Principles

- Connectivity is its own reward
- All functions which require knowledge of the state of end-to-end communication should be realized at the endpoints (end-to-end argument)
- There is no central instance which controls the Internet and which is able to turn it off
- Addresses should uniquely identify endpoints
- Intermediate systems should be stateless wherever possible
- Implementations should be liberal in what they accept and stringent in what they generate
- Keep it simple (when in doubt during design, choose the simplest solution)

Internet Design Principles

- Stateful services keep track of sessions or transactions and react differently to the same inputs based on that history.
- Stateless services rely on clients to maintain sessions and center around operations that manipulate resources, rather than the state.
- The internet moves from stateful to stateless architectures

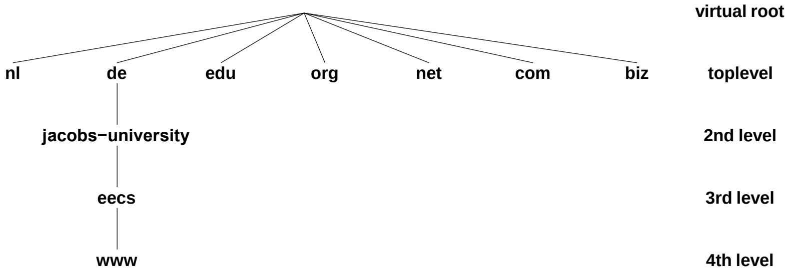
Internet Design Principles

- A communications network is a collection of systems (nodes) which transfer information between users attached to the networks.
- Two types of systems are defined within a communications network:
    - End Systems
    - Intermediate Systems

Internet Addresses

- Four byte IPv4 addresses are typically written as four decimal numbers separated by dots where every decimal number represents one byte (dotted quad notation). A typical example is the IPv4 address 192.0.2.1
- Sixteen byte IPv6 addresses are typically written as a sequence of hexadecimal numbers separated by colons (:) where every hexadecimal number represents two bytes
- Leading nulls in IPv6 addresses can be omitted and two consecutive colons can represent a sequence of nulls
- The IPv6 address 2001:00db8:0000:0000:0000:0000:0000:0001 can be written as 2001:db8::1
- See RFC 5952 for the recommended representation of IPv6 addresses

Internet Domain Names

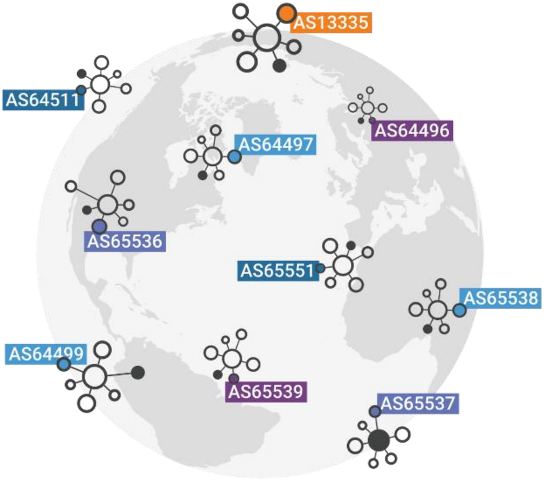


- The Domain Name System (DNS) provides a distributed hierarchical name space which supports the delegation of name assignments
- DNS name resolution translates DNS names into one or more Internet addresses

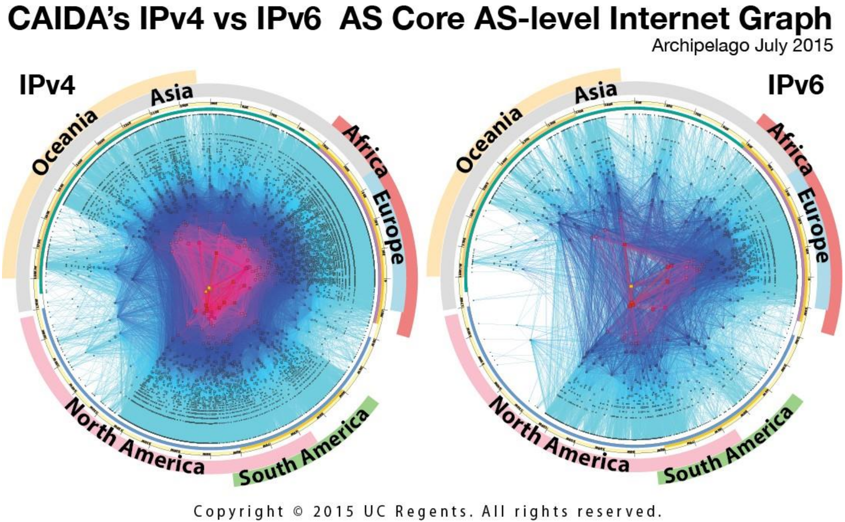
Autonomous Systems

- The global Internet consists of a set of inter-connected autonomous systems
- An *autonomous system* (AS) is a set of routers and networks under the same administration
- Autonomous systems are identified by 32-bit numbers, called AS numbers (ASNs) (originally the number space was limited to 16-bit but this has been increased to 32-bit)
- IP packets are forwarded between autonomous systems over paths that are established by an *Exterior Gateway Protocol* (EGP)
- Within an autonomous system, IP packets are forwarded over paths that are established by an *Interior Gateway Protocol* (IGP)

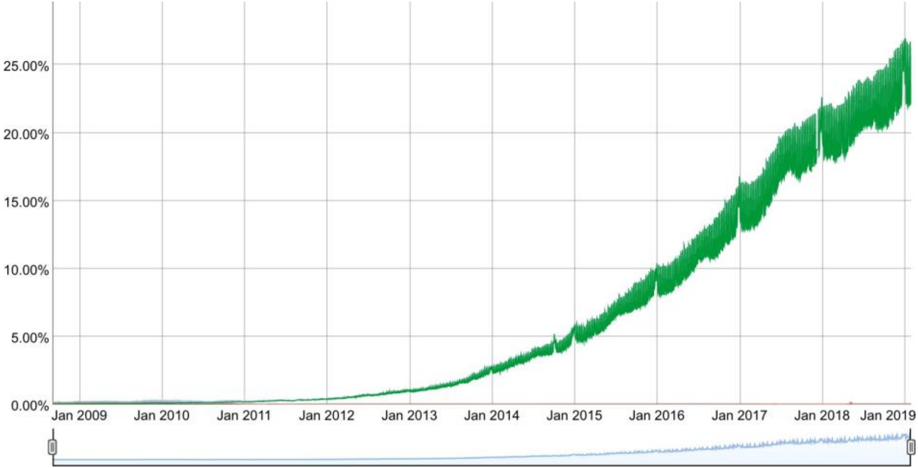
Autonomous Systems



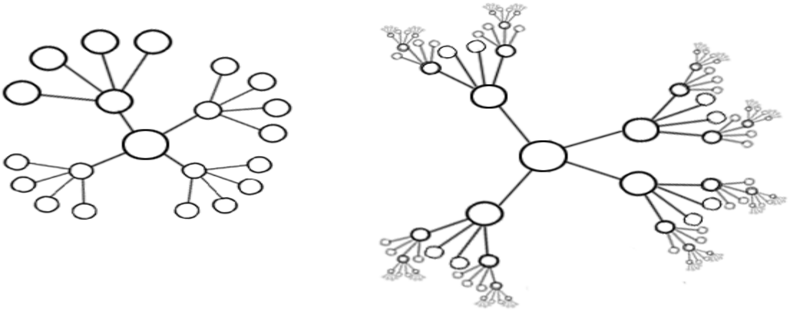
Autonomous Systems



Google IPv6 Adoption Statistics



Internet – A Scale-free Network?



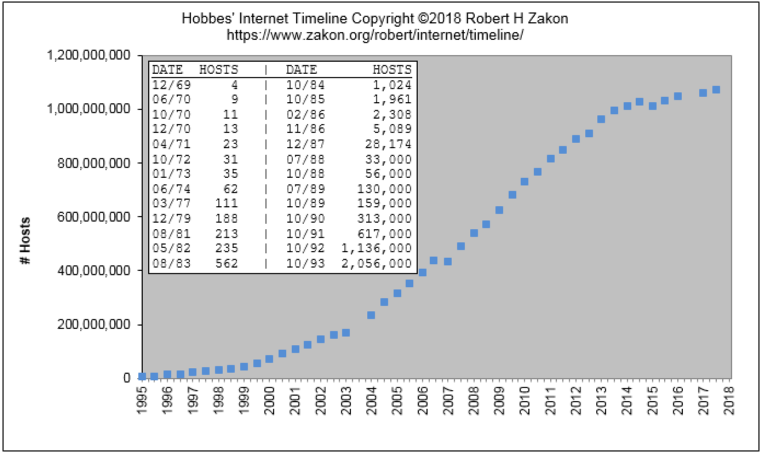
- Scale-free: The probability $P(k)$ that a node in the network connects with k other nodes is roughly proportional to $k^{-\gamma}$ for some parameter γ.

Section 2: Structure and Growth of the Internet

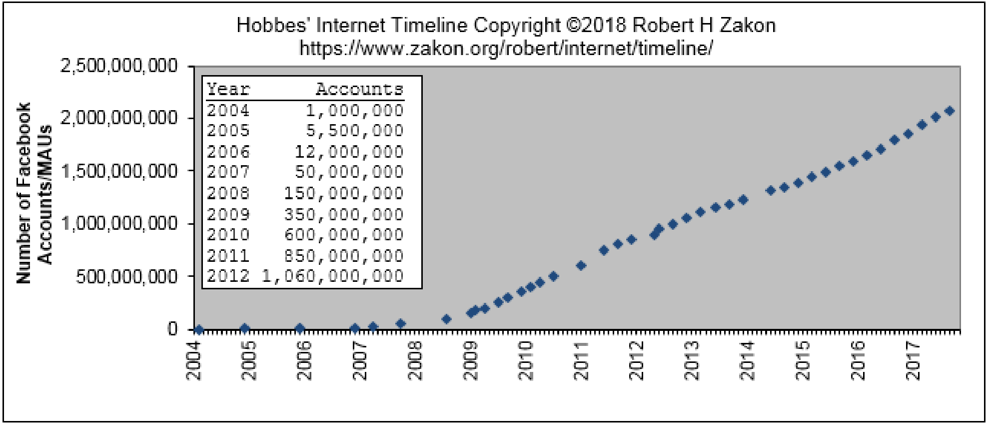
Evolution of Network Services

- 1970: private communication (email)
- 1980: discussion forums (usenet)
- 1985: software development and standardization (GNU)
- 1995: blogs, art, games, trading, searching (ebay, amazon)
- 1998: multimedia communication (rtp, sip, netflix)
- 2000: books and encyclopedia (wikipedia)
- 2005: social networks, video sharing (facebook, youtube)
- 2010: cloud computing, content delivery networks (akamai, amazon)
- 2015: voice-controlled personal assistants (amazon alexa, google home)
- 2020: Internet of Every Thing (IOE)

Growth of the Internet



Growth of Facebook

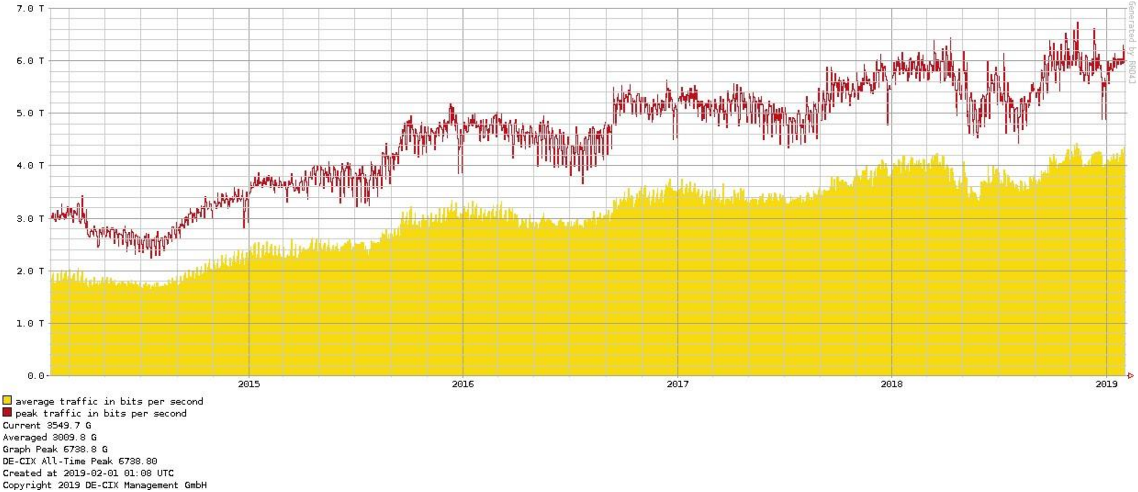


Year	Accounts
2004	1,000,000
2005	5,500,000
2006	12,000,000
2007	50,000,000
2008	150,000,000
2009	350,000,000
2010	600,000,000
2011	850,000,000
2012	1,060,000,000

Internet Exchange Points (2021)

- Internet Exchange Frankfurt/Germany (DE-CIX)
  - Connected networks: ≈800
  - Average throughput (1 year): ≈ $4.0 Tbps$
  - Maximum throughput (1 year): ≈ $6.0 Tbps$
  - Maximum transport capacity: ≈ $8 Tbps$
  - Total optical backbone capacity: ≈$48 Tbps$
  - 3 × 160 100 Gigabit-Ethernetports
  - http://www.decix.de/
- Amsterdam Internet Exchange (AMS-IX)
  - http://www.ams-ix.net/
- London Internet Exchange (LINX)
  - https://www.linx.net/

DE-CIX Traffic (5 Years)



Networking Challenges

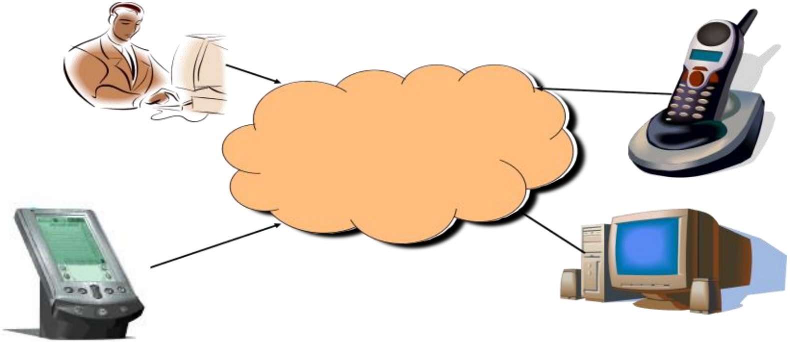
- Switching efficiency and energy efficiency
- Routing and fast convergence
- Security, trust, and key management
- Network measurements and automated network operations
- Ad-hoc networks and self-organizing networks
- Wireless sensor networks and the Internet of Things
- Delay and disruption tolerant networks
- High bandwidth and long delay networks
- Home networks, data center networks, access networks
- ...

Section 3: Internet Programming with Sockets

Internet Programming with Sockets

- Sockets are abstract communication endpoints with a rather small number of associated function calls
- The socket API consists of
  - address formats for various network protocol families
  - functions to create, name, connect, destroy sockets
  - functions to send and receive data
  - functions to convert human readable names to addresses and vice versa
  - functions to multiplex I/O on several sockets
- Sockets are the de-facto standard communication API provided by operating systems

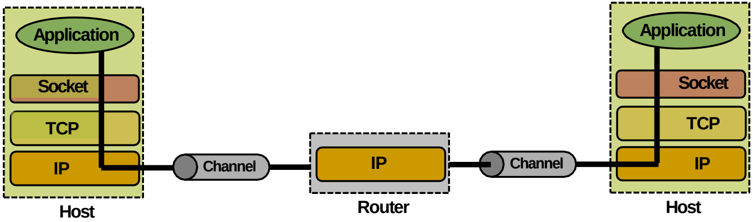
Internet Programming with Sockets



Also known as a "host"...

Berkley Sockets

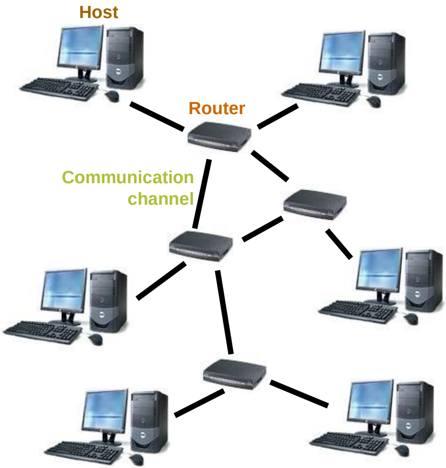
- Universally known as **Sockets**
- It is an abstraction through which an application may send and receive data
- Provide **generic access** to interprocess communication services
  - e.g. IPX/SPX, Appletalk, TCP/IP
- Standard API for networking



Internet Programming with Sockets

- Computer Network
  - hosts, routers,
    communication channels
- **Hosts** run applications
- **Routers** forward information
- **Packets**: sequence of bytes
  - contain control information
  - e.g. destination host
- **Protocol** is an agreement
  - meaning of packets
  - structure and size of packets

  e.g. Hypertext Transfer Protocol (HTTP)



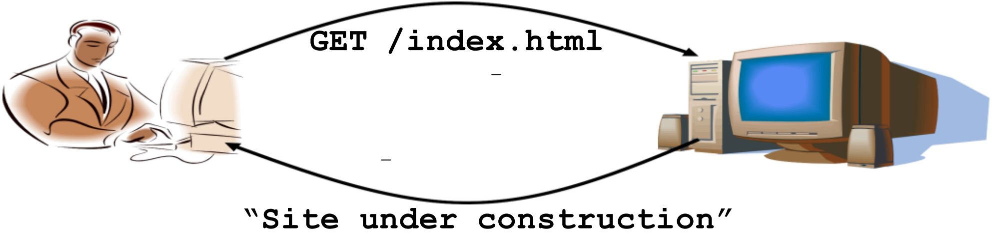
Client and Server

Client program

- Running on end host
- Requests service
- E.g., Web browser

Server program

- Running on end host
- Provides service
- E.g., Web server



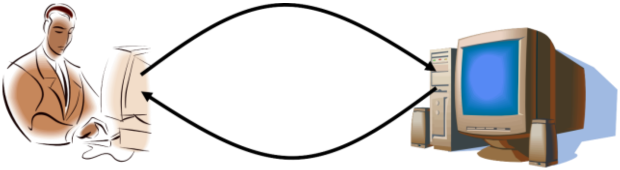
Client-Server Communication

Client

- Sometimes on
- Initiates a request to the server when interested
- E.g., web browser
- Needs to know the server's address

Server

- Always on
- Serve services to many clients
- E.g., https://www.jacobs-university.de/
- Not initiate contact with the clients
- Needs a fixed address



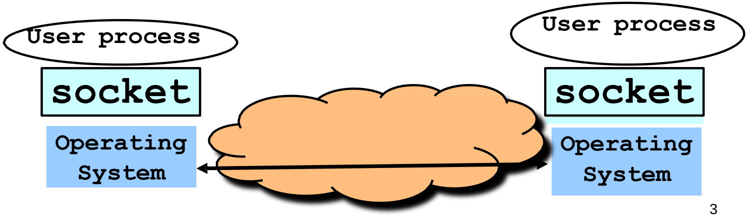
Socket: End Point of Communication

Processes send messages to one another

- Message traverse the underlying network

A Process sends and receives through a "socket"

- Analogy: the doorway of the house.
- Socket, as an API, supports the creation of network applications



UNIX Socket API

Socket interface

- A collection of system calls to write a networking program at user-level.
- Originally provided in Berkeley UNIX
- Later adopted by all popular operating systems

In UNIX, everything is like a file

- All input is like reading a file
- All output is like writing a file
- File is represented by an integer file descriptor
- Data written into socket on one host can be read out of socket on other host

System calls for sockets

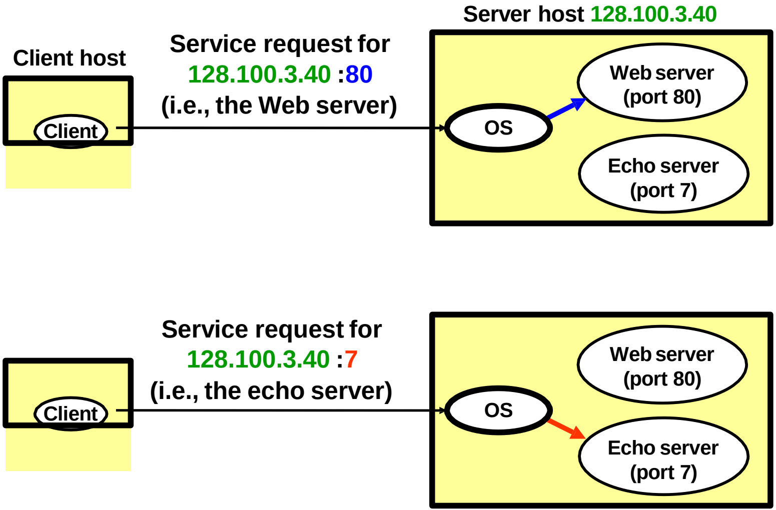
- Client: create, connect, write, read, close
- Server: create, bind, listen, accept, read, write, close

Typical Client Program

Prepare to communicate

- Create a socket
- Determine server address and port number
- Why do we need to have port number?

Using Ports to Identify Services



Socket Parameters

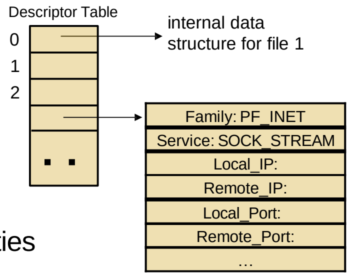
A socket connection has 5 general parameters:

- The protocol
  - Example: TCP and UDP.
- The local and remote address
  - Example: 128.100.3.40
- The local and remote port number
  - Some ports are reserved (e.g., 80 for HTTP)
  - Root access require to listen on port numbers below 1024

DNSlookup	UDP	53
FTP	TCP	21
HTTP	TCP	80
POP3	TCP	110
Telnet	TCP	23

Sockets

- Uniquely identified by
  - an internet address
  - an end-to-end protocol (e.g. TCP or UDP)
  - aport number
- Two types of (TCP/IP) sockets
  - **Stream** sockets (e.g. uses TCP)
    - provide reliable byte-stream service
  - **Datagram** sockets (e.g. uses UDP)
    - provide best-effort datagram service
    - messagesup to 65.500bytes
    - Socket extend the convectional UNIX I/O facilities
      - file descriptors for network communication
      - extended the read and write system calls



Typical Client Program

Prepare to communicate

- Create a socket
- Determine server address and port number
- Initiate the connection to the server

Exchange data with the server

- Write data to the socket
- Read data from the socket
- Do stuff with the data (e.g., render a Web page)

Close the socket

Important Functions for Client Program

- socket()

  create the socket descriptor
- connect()

  connect to the remote server
- read(),write()

  communicate with the server
- close()

  end communication by closing socket descriptor

Creating a Socket

int socket(int domain, int type, int protocol)

- Returns a descriptor (or handle) for the socket
- Domain: protocol family
  - PF_INET for the Internet
- Type: semantics of the communication
  - SOCK_STREAM: Connection oriented
  - SOCK_DGRAM: Connectionless
- Protocol: specific protocol
  - UNSPEC: unspecified
  - (PF_INET and SOCK_STREAM already implies TCP)
- **E.g., TCP: sd = socket(PF_INET, SOCK_STREAM, 0);**
- **E.g., UDP: sd = socket(PF_INET, SOCK_DGRAM, 0);**

Connecting to the Server

- *int connect(int sockfd, struct sockaddr *server_address, socketlen_t addrlen)*
  - Arguments: socket descriptor, server address, and address size
  - Remote address and port are in struct sockaddr
  - Returns 0 on success, and -1 if an error occurs

Sending and Receiving Data

Sending data

- *write(int sockfd, void *buf, size_t len)*
  - Arguments: socket descriptor, pointer to buffer of data, and length of the buffer
  - Returns the number of characters written, and -1 on error

Receiving data

- *read(int sockfd, void *buf, size_t len)*
  - Arguments: socket descriptor, pointer to buffer to place the data, size of the buffer
  - Returns the number of characters read (where 0 implies "end of file"), and -1 on error

Closing the socket

- *int close(int sockfd)*

Servers Differ From Clients

Passive open

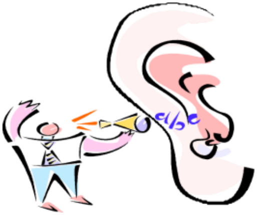
- Prepare to accept connections
- … but don't actually establish one
- … until hearing from a client

Hearing from multiple clients

- Allow a backlog of waiting clients
- ... in case several try to start a connection at once

Create a socket for each client

- Upon accepting a new client
- … create a *new* socket for the communication



Typical Server Program

Prepare to communicate

- Create a socket
- Associate local address and port with the socket

Wait to hear from a client (passive open)

- Indicate how many clients-in-waiting to permit
- Accept an incoming connection from a client

Exchange data with the client over new socket

- Receive data from the socket
- Send data to the socket
- Close the socket

Repeat with the next connection request

Important Functions for Server Program

- socket()
  create the socket descriptor
- bind()
  associate the local address
- listen()
  wait for incoming connections from clients
- accept()
  accept incoming connection
- read(),write()
  communicate with client
- close()
  close the socket descriptor

Socket Preparation for Server

Bind socket to the local address and port

- *int bind (int sockfd, struct sockaddr *my_addr, socklen_t addrlen)*
- Arguments: socket descriptor, server address, address length
- Returns 0 on success, and -1 if an error occurs

Define the number of pending connections

- *int listen(int sockfd, int backlog)*
- Arguments: socket descriptor and acceptable backlog
- Returns 0 on success, and -1 on error

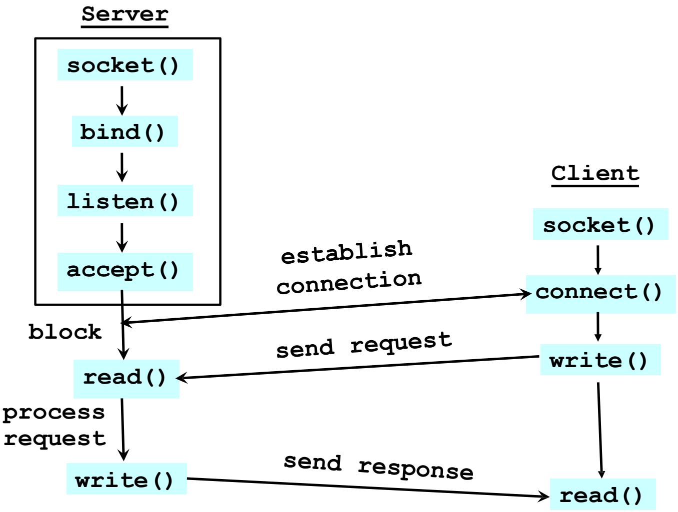
Server Operation

- **accept()** returns a new socket descriptor as output

•New socket should be closed when done with communication

•Initial socket remains open, can still accept more connections

Putting it All Together



Supporting Function Calls

gethostbyname() get address for given host name (e.g. 128.100.3.40 for name “cs.toronto.edu”);

getservbyname() get port and protocol for a given service e.g. ftp, http (e.g. “http” is port 80, TCP)

getsockname() get local address and local port of a socket

getpeername() get remote address and remote port of a socket

IPv4 Socket Addresses

```
#include <sys/socket.h>
#include <netinet/in.h>

typedef ... sa_family_t;
typedef ... in_port_t;

struct in_addr {  uint8_t
       s_addr[4];                          /* IPv4 address */
};

struct sockaddr_in {
       uint8_t          sin_len;           /* address length (BSD) */
       sa_family_t sin_family;             /* address family */
       in_port_t        sin_port;          /* transport layer port */
       struct in_addr sin_addr;            /* IPv4 address */
};
```

IPv6 Socket Addresses

```
#include <sys/socket.h>
#include <netinet/in.h>

typedef ... sa_family_t;
typedef ... in_port_t;

struct in6_addr {
    uint8_t   s6_addr[16];             /* IPv6 address */
};

struct sockaddr_in6 {
    uint8_t        sin6_len;           /* address length (BSD) */
    sa_family_t    sin6_family;        /* address family */
    in_port_t      sin6_port;          /* transport layer port */
    uint32_t       sin6_flowinfo;      /* flow information */
    struct in6_addr sin6_addr;         /* IPv6 address */
    uint32_t       sin6_scope_id;      /* scope identifier */
};
```

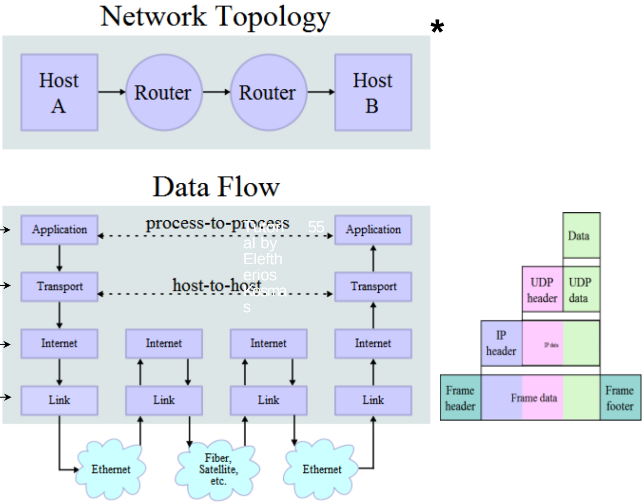
TCP/IP

FTP, SMTP, …

Transport Layer
TCP or UDP

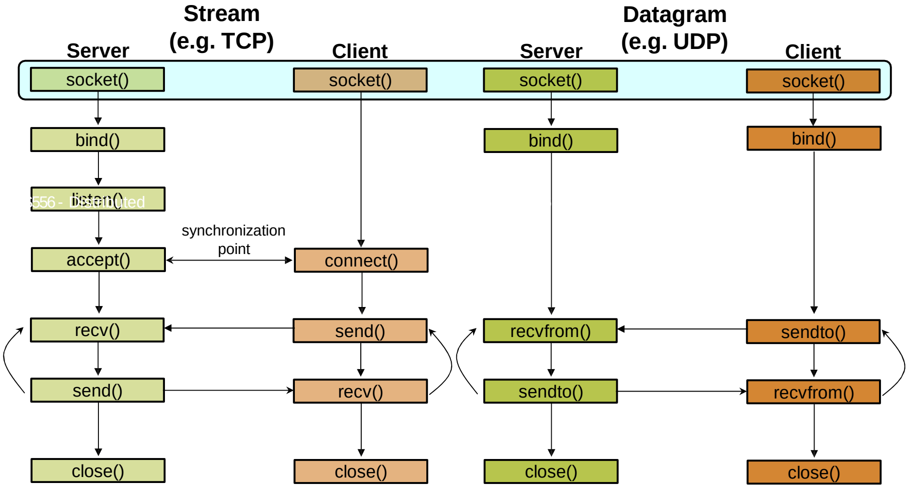
Network Layer
IP

Communication
Channels



* image is taken from “http://en.wikipedia.org/wiki/TCP/IP_model”

Client - Server Communication - Unix



Socket API Summary

```
#include <sys/types.h>
#include <sys/socket.h>
#include <unistd.h>

#define                    ...
SOCK_STREAM                ...
#define                    ...
SOCK_DGRAM                 ...
#define SCOK_RAW
#define SOCK_RDM
#define SOCK_SEQPACKET
...

#define   AF_LOCAL   ...
#define    AF_INET    ...
#define AF_INET6 ...

#define   PF_LOCAL   ...
#define    PF_INET    ...
#define PF_INET6 ...
```

Socket API Summary

```
int socket(int domain, int type, int protocol);  int bind(int
socket, struct sockaddr *addr,
                socklen_t addrlen);
int connect(int socket, struct sockaddr *addr,  socklen_t
                     addrlen);
int listen(int socket, int backlog);
int accept(int socket, struct sockaddr *addr,  socklen_t
                  *addrlen);

ssize_t write(int socket, void *buf, size_t count);  int send(int
socket, void *msg, size_t len, int  flags);
int sendto(int socket, void *msg, size_t len, int  flags,
                  struct sockaddr *addr, socklen_t addrlen);

ssize_t read(int socket, void *buf, size_t count);
int recv(int socket, void *buf, size_t len, int flags);  int recvfrom(int
socket, void *buf, size_t len, int  flags,
                      struct sockaddr *addr, socklen_t *addrlen);
```

Socket API Summary

```
int shutdown(int socket, int how);  int
close(int socket);

int getsockopt(int socket, int level, int optname,  void *optval,
                              socklen_t *optlen);
int setsockopt(int socket, int level, int optname,
                              void *optval, socklen_t optlen);  int
getsockname(int socket, struct sockaddr *addr,
                                socklen_t *addrlen);
int getpeername(int socket, struct sockaddr *addr,
                                socklen_t *addrlen);
```

- All API functions operate on abstract socket addresses
- Not all functions make equally sense for all socket types

Mapping Names to Addresses

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>

#define      AI_PASSIVE      ...
#define   AI_CANONNAME    ...
#define AI_NUMERICHOST ...

struct addrinfo {  int
        int                    ai_flags;
        int  int               ai_family;
        size_t                 ai_socktype;
                               ai_protocol;
                               ai_addrlen;
        struct sockaddr *ai_addr;  char
                               *ai_canonname;
        struct addrinfo *ai_next;
};
```

Mapping Names to Addresses

```
int getaddrinfo(const char *node,
                const char *service,
                const struct addrinfo *hints,  struct
                addrinfo **res);
void freeaddrinfo(struct addrinfo *res);
const char *gai_strerror(int errcode);
```

- Many books still document the old name and address mapping functions
  - gethostbyname()
  - gethostbyaddr()
  - getservbyname()
  - getservbyaddr()

  which are IPv4 specific and should not be used anymore

Mapping Addresses to Names

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>

#define NI_NOFQDN          ...
#define NI_NUMERICHOST   ...
#define NI_NAMEREQD        ...
#define NI_NUMERICSERV ...
#define NI_NUMERICSCOPE ...
#define NI_DGRAM            ...

int getnameinfo(const struct sockaddr *sa,
                socklen_t salen,
                char *host, size_t hostlen,   char
                *serv, size_t servlen,  int flags);
const char *gai_strerror(int errcode);
```

Multiplexing

```
#include <sys/select.h>

typedef ... fd_set;

FD_ZERO(fd_set *set);
FD_SET(int fd, fd_set *set);
FD_CLR(int fd, fd_set *set);
FD_ISSET(int fd, fd_set *set);

int select(int n, fd_set *readfds, fd_set *writefds,  fd_set
           *exceptfds, struct timeval *timeout);
int pselect(int n, fd_set *readfds, fd_set *writefds,
            fd_set *exceptfds, struct timespec *timeout,  sigset_t
            sigmask);
```

- select() works with arbitrary file descriptors
- select() frequently used to implement the main loop of event-driven programs

References

B. Carpenter.
Architectural Principles of the Internet.
RFC 1958, IAB, June 1996.

B. Carpenter.
Internet Transparency.
RFC 2775, IBM, February 2000.

R. Gilligan, S. Thomson, J. Bound, J. McCann, and W. Stevens.
Basic Socket Interface Extensions for IPv6.
RFC 3493, Intransa Inc., Cisco, Hewlett-Packard, February 2003.

R. Atkinson and S. Floyd.
IAB Concerns and Recommendations Regarding Internet Research and Evolution.
RFC 3869, Internet Architecture Board, August 2004.

S. Kawamura and M. Kawashima.
A Recommendation for IPv6 Address Text Representation.
RFC 5952, NEC BIGLOBE, Ltd., NEC AccessTechnica, Ltd., August 2010.