CS3281 / CS5281

Networking

CS3281 / CS5281

Fall 2025

Intro to Sockets

- Sockets: method for IPC between applications
    - Can be on the same host
    - Can be on a different host connected by a network
- Typical organization: client-server
    - The client makes requests
        - Example: a web browser
    - The server responds to requests
        - Example: an Apache web server
- Communication involves a network protocol
    - Usually multiple layers of network protocols
- We'll cover TCP/IP
    - Also called the Internet protocol suite

Big Picture: The Internet

- Began in 1960s: as network that could connect computers that were far away
  - Funding came from DARPA, and first ARPANET message was sent from UCLA to Stanford (350 miles) in 1969
- Originally linked research operations and CS departments
  - Spread to the commercial world in the 1990s and become “the Internet”
- Today: the Internet links millions of loosely connected, independent networks
- Data is sent through the networks in “packets” called IP (Internet Protocol) packets
  - Transported in one or more physical mediums, like Ethernet or WiFi
  - Each IP packet passes through multiple gateways
    - Each gateway passes the packet to a gateway closer to the ultimate destination
- An internet (lowercase i) connects different computer networks
  - The Internet (capital I) refers to the TCP/IP internet that connects millions of computers
  - Some modern style guides do not capitalize “Internet.” We do here for conceptual clarity.

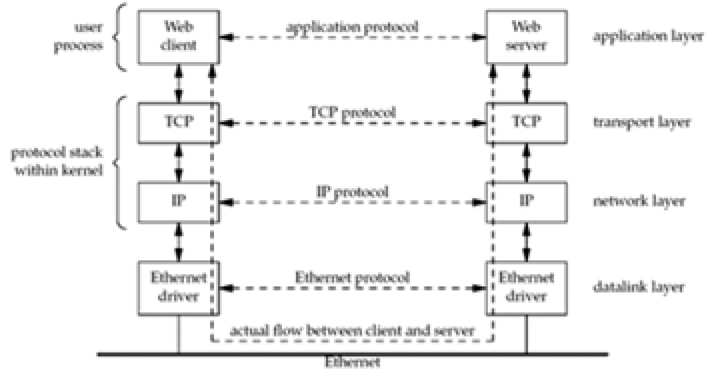
https://en.wikipedia.org/wiki/Capitalization_of_Internet

The Internet (cont.)

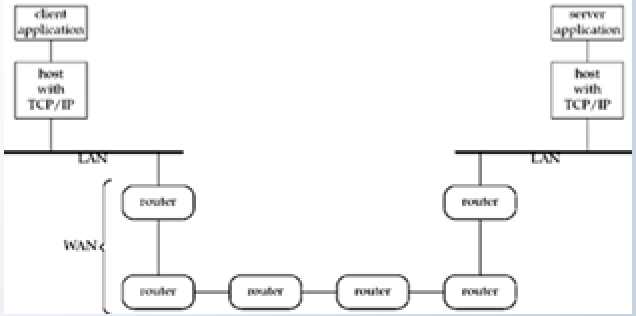
- The core protocol is the Internet Protocol
  - Defines a uniform transport mechanism and a common format for information in transit
  - IP packets are carried by networks with different technologies
- The Transmission Control Protocol (TCP) sits on top of IP
  - TCP provides a reliable mechanism for sending arbitrarily long sequences of bytes
- Above TCP, higher-level protocols use TCP to provide services that we think of as "the Internet"
  - Examples: browsing, e-mail, file sharing
- All of these protocols taken together define the Internet

Protocol Layers

- Example on the right:
  - Web servers and web clients communicate using TCP
  - TCP uses the Internet Protocol (IP)
  - IP uses a data link protocol (like Ethernet)
- The client and server use an application protocol
  - The transport layers use the TCP protocol
- Information flows down the protocol stack on one side, back up on the other
- Client and server are in user space
  - TCP, IP, data link in kernel space (usually)



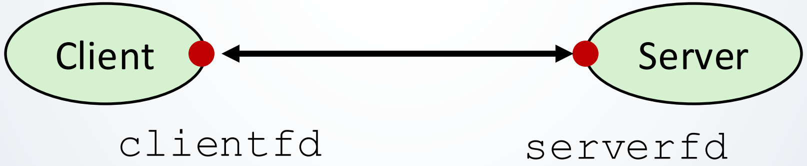
On the same LAN (Local Area Network)



On different LANs (Wide Area Network)

Sockets

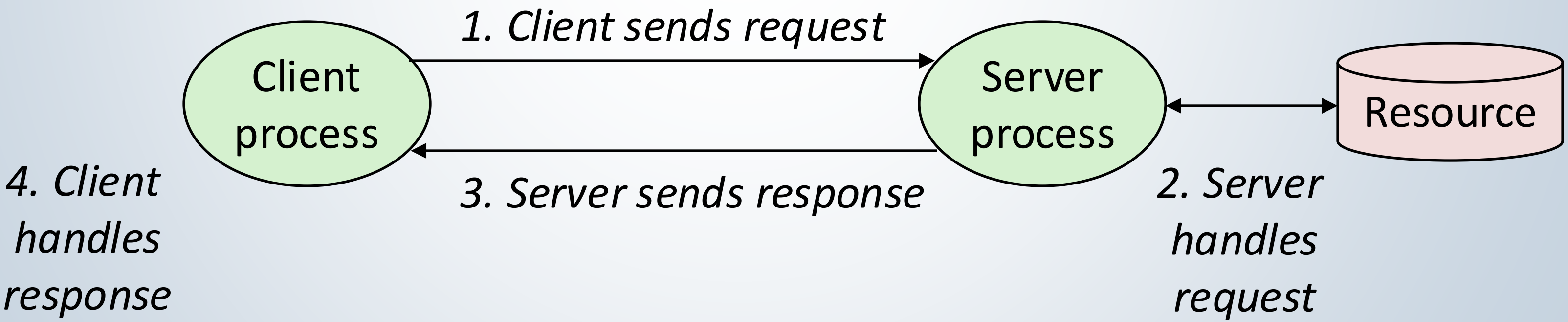
- What is a socket?
  - To the kernel, a socket is an endpoint of communication
  - To an application, a socket is a file descriptor that lets the application read/write from/to the network
    - ***Remember:*** All Unix I/O devices, including networks, are modeled as files
- Clients and servers communicate with each other by reading from and writing to socket descriptors



- The main distinction between regular file I/O and socket I/O is how the application "opens" the socket descriptors

A Client-Server Transaction

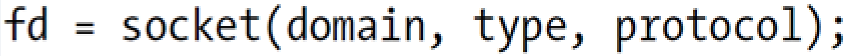
- Most network applications are based on the client-server model:
  - A ***server*** process and one or more ***client*** processes
  - Server manages some ***resource***
  - Server provides ***service*** by manipulating resource for clients
  - Server activated by request from client (telephone analogy)



Note: clients and servers are processes running on hosts (can be the same or different hosts)

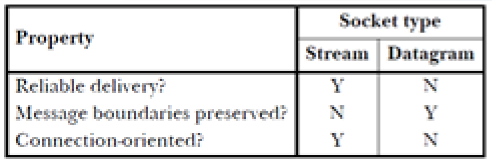
Sockets and Client/Server Communications

- Each application creates a socket
- The server binds its socket to a well-known address so clients can locate it



```
fd = socket(domain, type, protocol);
```

- Domain determines:
  - Format of address, and range of communication (same or different hosts)
    - AF_UNIX, AF_INET, AF_INET6
- Type: stream or datagram
- Protocol: generally 0
  - Nonzero for some types like raw sockets (passes directly from data link to application)



Property	Socket type	
	Stream	Datagram
Reliable delivery?	Y	N
Message boundaries preserved?	N	Y
Connection-oriented?	Y	N

Stream Sockets

- Stream sockets provide reliable, bidirectional, byte-stream communication
    - Reliable: Either the transmitted data arrives intact at the receiving end, or we receive notification of a probable failure in transmission
    - Bidirectional: data may be transmitted in either direction
    - Byte-stream: no message boundaries
        - Example: receiver doesn't know if the sender originally sent two 1-byte messages or one 2-byte message
- Operate in connected pairs (aka connection oriented)
    - *Peer* socket: socket at the other end of a connection
    - *Peer* address: address of that socket
    - *Peer* application: application using the peer socket
        - *Peer* is equivalent to *remote* or *foreign*

Datagram Sockets

- Allow data to be exchanged in the form of messages called *datagrams*
  - Message boundaries are preserved
  - Data transmission
    - not reliable: data may arrive out of order, be duplicated, or not arrive
    - connectionless: datagrams may take different routes between source and destination
- In the Internet domain:
  - Datagram sockets use UDP
  - Stream sockets use TCP

**Almost always*

Protocols and Communication

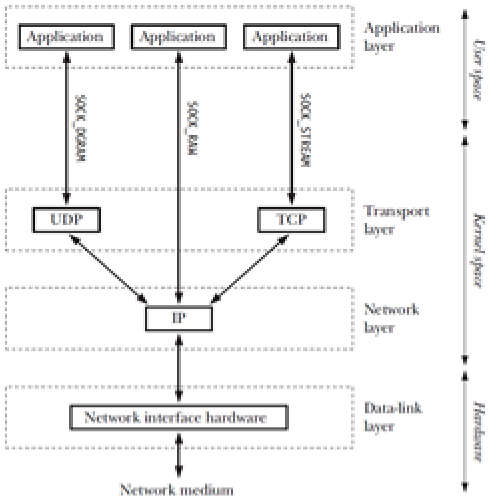


Figure 58-2: Protocols in the TCP/IP suite

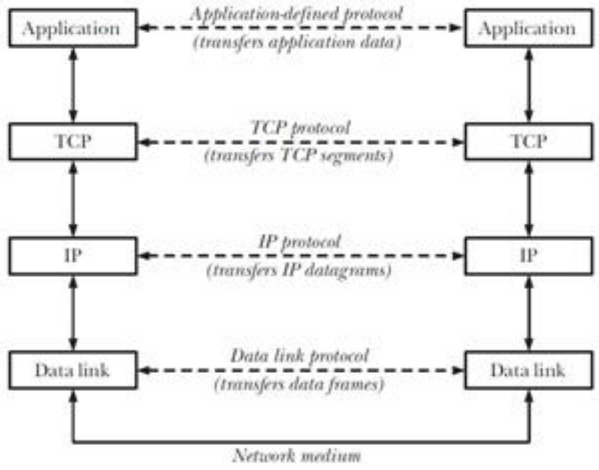


Figure 58-3: Layered communication via the TCP/IP protocols

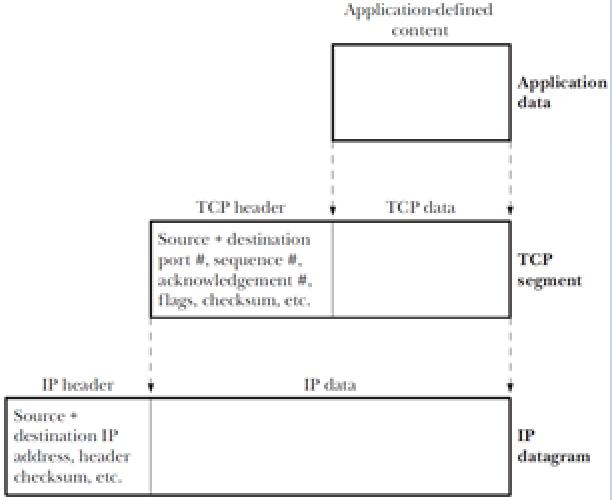

Application-defined content
Application data
TCP header
TCP data
Source + destination port #, sequence #, acknowledgement #, flags, checksum, etc.
TCP segment
IP header
IP data
Source + destination IP address, header checksum, etc.
IP datagram

Figure 58-4: Encapsulation within the TCP/IP protocol layers

Global IP Internet (Upper Case)

- Most famous example of an internet

- Based on the TCP/IP protocol family
  - IP (Internet Protocol) :
    - Provides *basic naming scheme* and unreliable *delivery capability* of packets (datagrams) from *host-to-host*
  - UDP (Unreliable Datagram Protocol)
    - Uses IP to provide *unreliable* datagram delivery from *process-to-process*
  - TCP (Transmission Control Protocol)
    - Uses IP to provide *reliable* byte streams from *process-to-process* over *connections*

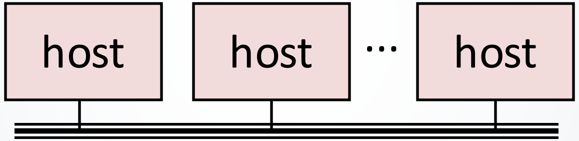
- Accessed via a mix of Unix file I/O and functions from the *sockets interface*

Computer Networks

- A *network* is a hierarchical system of boxes and wires organized by geographical proximity
  - LAN (Local Area Network) spans a building or campus
  - WAN (Wide Area Network) spans country or world
- An *internetwork (internet)* is an interconnected set of networks
  - The Global IP Internet (uppercase "I") is the most famous example of an internet (lowercase "i")
- Let's see how an internet is built from the ground up

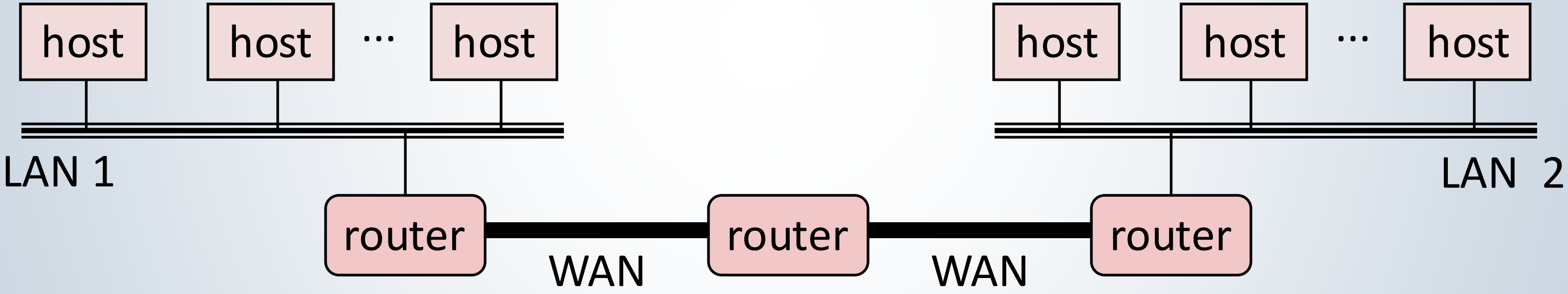
Conceptual View of LANs

- For simplicity, hubs, bridges, and wires are often shown as a collection of hosts attached to a single wire:



Next Level: Internets

- Multiple incompatible LANs can be physically connected by specialized computers called *routers*
- The connected networks are called an *internet* (lower case)



The Notion of an Internet Protocol

- How is it possible to send bits across LANs and WANs?

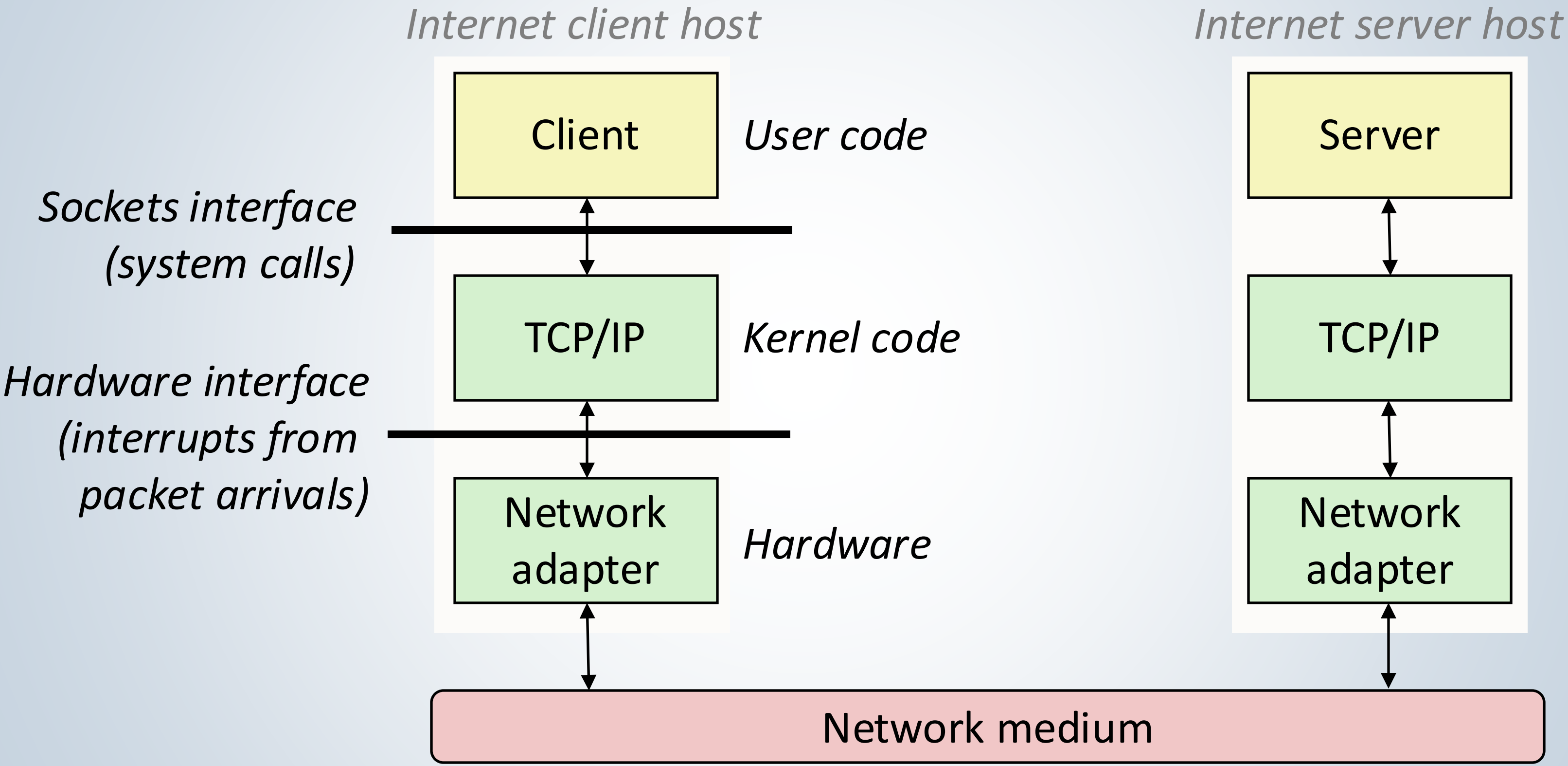
- Solution: *protocol* software running on each host and router
  - Protocol is a set of rules that governs how hosts and routers should cooperate when they transfer data from network to network.
  - Smooths out the differences between the different networks

What Does an Internet Protocol Do?

- Provides a *naming scheme*
  - An internet protocol defines a uniform format for ***host addresses***
  - Each host (and router) is assigned at least one of these internet addresses that uniquely identifies it

- Provides a *delivery mechanism*
  - An internet protocol defines a standard transfer unit (***packet***)
  - Packet consists of ***header*** and ***payload***
    - Header: contains info such as packet size, source and destination addresses
    - Payload: contains data bits sent from source host

Organization of an Internet Application



A Programmer’s View of the Internet

1. Hosts are mapped to a set of 32-bit *IP addresses*
   - 128.2.203.179

2. The set of IP addresses is mapped to a set of identifiers called Internet *domain names*
   - 129.59.107.38 is mapped to www.isis.vanderbilt.edu

3. A process on one Internet host can communicate with a process on another Internet host over a *connection*

Dotted-Decimal Notation

- By convention, each byte in a 32-bit IP address is represented by its decimal value and separated by a period
  - IP address: `0x8002C2F2 = 128.2.194.242`

IP Addresses

- Unsigned 32-bit IP addresses are stored in an *IP address struct*
  - IP addresses are always stored in memory in *network byte order* (big-endian byte order)
    - x86, ARM, risc-v all little endian
  - True in general for any integer transferred in a packet header from one machine to another
    - E.g., the port number used to identify an Internet connection

```
/* Internet address structure */
struct in_addr {
    uint32_t  s_addr; /* network byte order (big-endian) */
};
```

Internet Connections

- Clients and servers communicate by sending streams of bytes over *connections*. Each connection is:
  - *Point-to-point*: connects a pair of processes.
  - *Full-duplex*: data can flow in both directions at the same time,
  - *Reliable*: stream of bytes sent by the source is eventually received by the destination in the same order it was sent.

- *A socket* is an endpoint of a connection
  - *Socket address* is an **`IPaddress:port`** pair

- A *port* is a 16-bit integer that identifies a process:
  - ***Ephemeral port*:** Assigned automatically by client kernel when client makes a connection request.
  - ***Well-known port:*** Associated with some *service* provided by a server (e.g., port 80 is associated with Web servers)

Well-Known Ports and Service Names

- Popular services have permanently assigned *well-known ports and* corresponding *well-known service names*:
  - echo server: 7/echo
  - ssh servers: 22/ssh
  - email server: 25/smtp
  - Web servers: 80/http, 443/https

- Mappings between well-known ports and service names is contained in the file `/etc/services` on each Linux machine.

Anatomy of a Connection

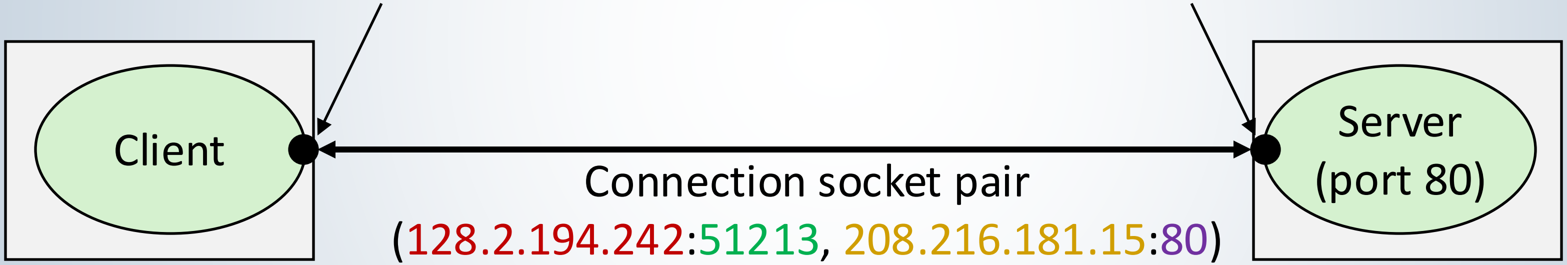
- A connection is uniquely identified by the socket addresses of its endpoints (*socket pair*)
  - `(cliaddr:cliport, servaddr:servport)`

Client socket address

128.2.194.242:51213

Server socket address

208.216.181.15:80



Client

Server

(port 80)

Connection socket pair

(128.2.194.242:51213, 208.216.181.15:80)

Client host address

128.2.194.242

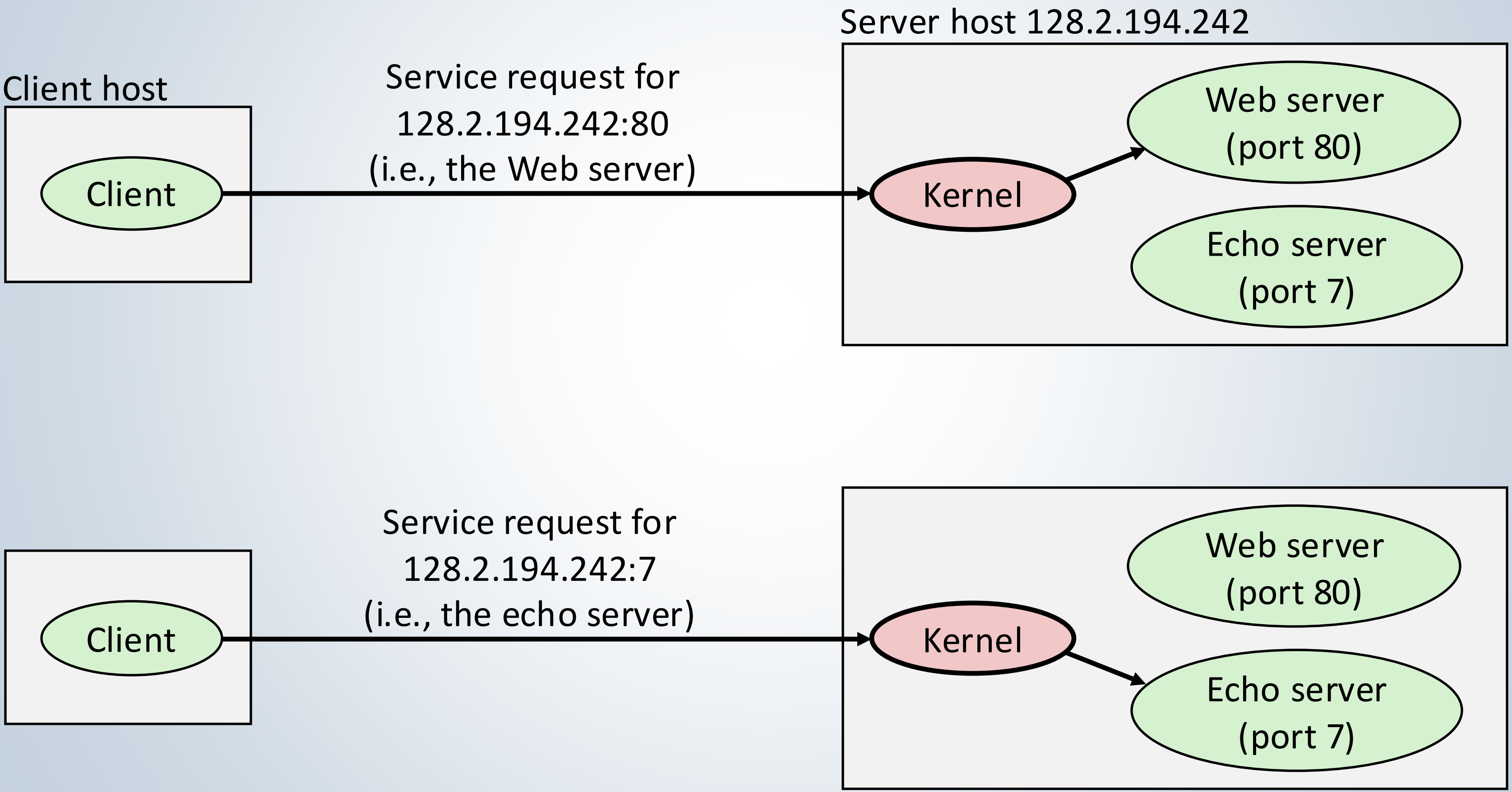
Server host address

208.216.181.15

51213 is an ephemeral port
allocated by the kernel

80 is a well-known port
associated with Web servers

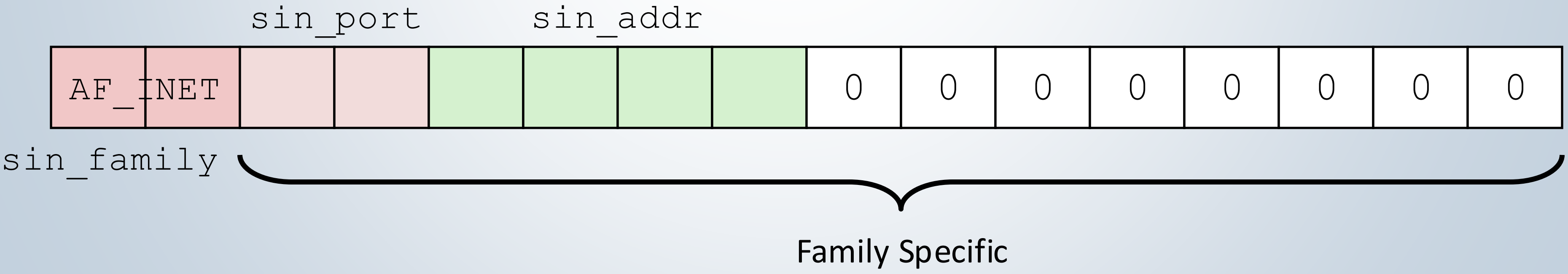
Using Ports to Identify Services



Socket Address Structures

- Internet-specific socket address

```
struct sockaddr_in  {
  uint16_t        sin_family;  /* Protocol family (always AF_INET) */
  uint16_t        sin_port;    /* Port num in network byte order */
  struct in_addr  sin_addr;    /* IP addr in network byte order */
  unsigned char   sin_zero[8]; /* Pad to sizeof(struct sockaddr) */
};
```

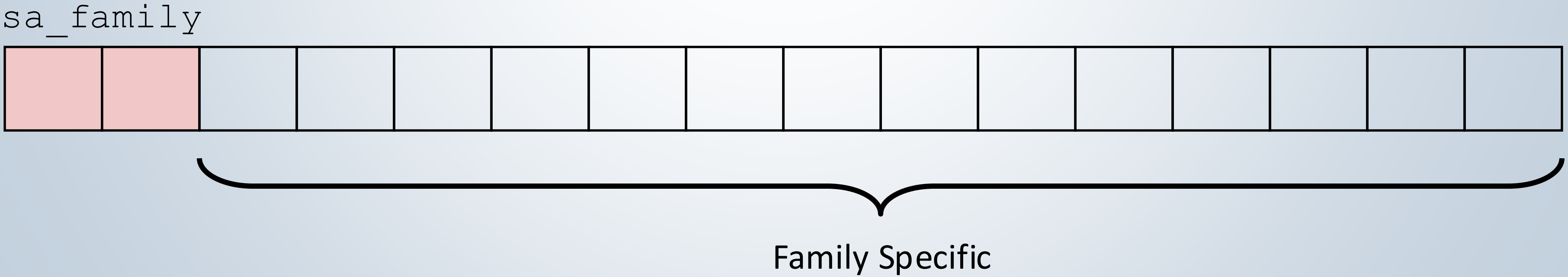


Socket-Address Structures

- Generic socket address:
  - For address arguments to **connect**, **bind**, and **accept**
  - Must cast (struct sockaddr_in *) to (struct sockaddr *) for functions that take socket address arguments.
  - Necessary only because C did not have generic (**void ***) pointers when the sockets interface was designed
  - For casting convenience, we adopt the Stevens convention:

    typedef struct sockaddr SA;

```
struct sockaddr {
  uint16_t  sa_family;    /* Protocol family */
  char      sa_data[14];  /* Address data.  */
};
```



sa_family

Family Specific

Sockets Interface

- Set of system-level functions used in conjunction with Unix I/O to build network applications.
- Created in the early 80's as part of the original Berkeley distribution of Unix that contained an early version of the Internet protocols.
- Available on all modern systems
  - – Unix variants, Windows, OS X, IOS, Android, ARM

Key Socket Calls

- socket() creates a new socket
- bind() binds a socket to an address
- listen() lets a TCP socket accept incoming connections from other sockets
- accept() accepts a connection from a peer application
- connect() establishes a connection with another socket
- Socket I/O can be done using
  - read() and write(), or
  - send(), recv(), sendto(), recvfrom()

Key Socket Calls

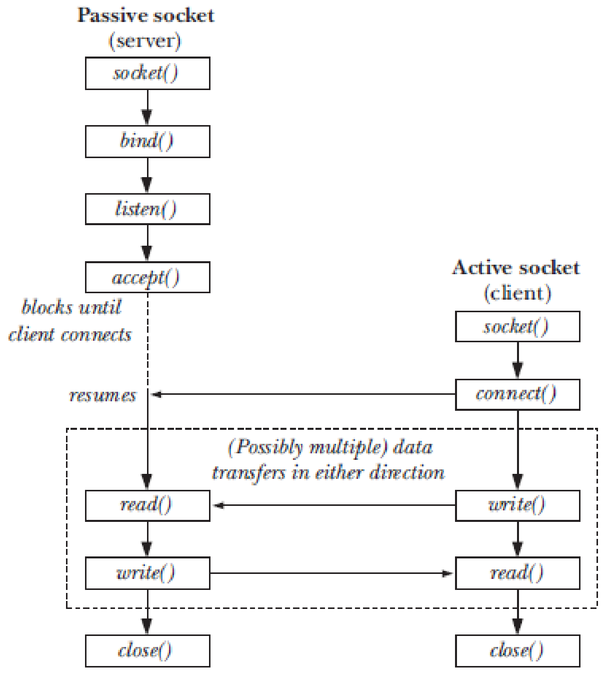


Figure 56-1: Overview of system calls used with stream sockets

Figure from *The Linux Programming Interface* by Michael Kerrisk