



CS3281 / CS5281
Networking

Will Hedgecock
Sandeep Neema
Bryan Ward



Tel (615) 343-7472 | Fax (615) 343-7440
1025 16th Avenue South Nashville, TN 37212
www.isis.vanderbilt.edu



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 through the networks in “packets” called IP (Internet Protocol) packets
 - Transported in one or more physical packets, 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

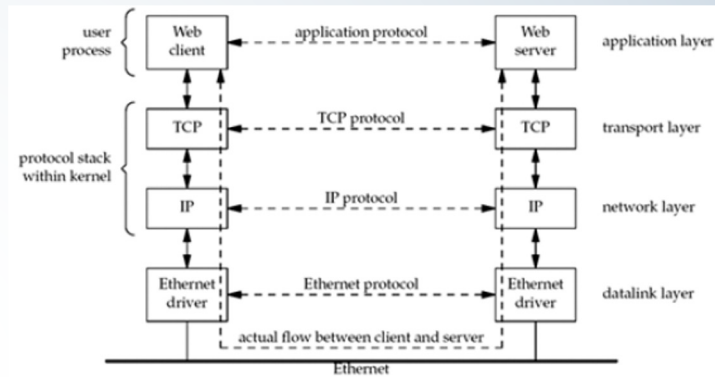


The Internet (cont.)

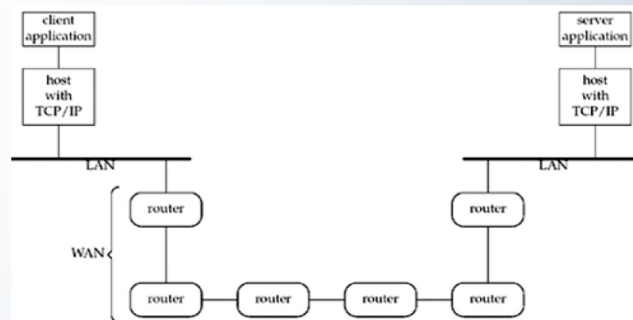
- The core protocol is the Internet Protocol
 - Defines a uniform transport mechanism and a common format for information in transit
 - IP packets are carried by different kinds of hardware using their own protocols
- The Transmission Control Protocol (TCP) sits on top of IP
 - TCP provides a reliable 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



On different LANs

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
- 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)

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

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 is not reliable
 - Data may arrive out of order, be duplicated, or not arrive at all
 - Example of a connectionless socket
 - Doesn't need to be connected to another socket in order to be used
- In the Internet domain:
 - Datagram sockets use UDP
 - Stream sockets use* TCP

**Almost always*



Key Socket Calls

- `socket()` creates a new socket
- `bind()` binds a socket to an address
- `listen()` lets a TCP socket to 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()`

Client-Server Example

```
void server()
{
    int fd;
    struct sockaddr_in in_addr;
    memset(&in_addr, 0, sizeof(struct sockaddr_in));

    if ((fd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        exit_with_error("Server error with socket");
    }

    in_addr.sin_family = AF_INET;
    in_addr.sin_port = 5001;
    inet_pton(AF_INET, "0.0.0.0", &in_addr.sin_addr);

    if (bind(fd, (struct sockaddr *) &in_addr, sizeof(struct sockaddr_in))) {
        exit_with_error("Server error with bind");
    }

    if (listen(fd, 0)) {
        exit_with_error("Server error with listen");
    }

    int port;
    struct sockaddr_in client_info;
    socklen_t client_size = sizeof(client_info);
    memset(&client_info, 0, sizeof(client_info));

    if ((port = accept(fd, (struct sockaddr *) &client_info, &client_size)) == -1) {
        exit_with_error("Server error with accept");
    }

    int ret, total = 0;
    char buf[100];
    while ((ret = read(port, buf + total, sizeof(buf) - 1 - total)) {
        if (ret == -1) {
            exit_with_error("Server error with read");
        }
        total += ret;
    }

    buf[total] = 0;
    printf("Server received: %s, total = %d\n", buf, total);
}
```

```
void client(int port)
{
    int fd;
    struct sockaddr_in in_addr;
    memset(&in_addr, 0, sizeof(struct sockaddr_in));

    if ((fd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        exit_with_error(0);
    }

    in_addr.sin_family = AF_INET;
    in_addr.sin_port = 5001;
    inet_pton(AF_INET, "127.0.0.1", &in_addr.sin_addr);

    if (connect(fd, (struct sockaddr *) &in_addr, sizeof(struct sockaddr_in))) {
        exit_with_error("Client error with connect");
    }

    int ret, sent = 0;
    char *msg = "Hello, server!";
    while (sent != strlen(msg)) {
        ret = write(fd, msg + sent, strlen(msg) - sent);
        if (ret == -1) {
            exit_with_error("Client error with write");
        }
        else {
            sent += ret;
        }
        printf("sent = %d\n", sent);
    }

    void exit_with_error(char *msg)
    {
        perror(msg);
        exit(1);
    }
}
```

Client-Server Example

```
void server()
{
    int fd;
    struct sockaddr_in in_addr;
    memset(&in_addr, 0, sizeof(struct sockaddr_in));

    if ((fd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        exit_with_error("Server error with socket");
    }

    in_addr.sin_family = AF_INET;
    in_addr.sin_port = 5001;
    inet_pton(AF_INET, "0.0.0.0", &in_addr.sin_addr);

    if (bind(fd, (struct sockaddr *) &in_addr, sizeof(struct sockaddr_in))) {
        exit_with_error("Server error with bind");
    }

    if (listen(fd, 0)) {
        exit_with_error("Server error with listen");
    }

    int port;
    struct sockaddr_in client_info;
    socklen_t client_size = sizeof(client_info);
    memset(&client_info, 0, sizeof(client_info));

    if ((port = accept(fd, (struct sockaddr *) &client_info, &client_size)) == -1) {
        exit_with_error("Server error with accept");
    }

    int ret, total = 0;
    char buf[100];
    while ((ret = read(port, buf + total, sizeof(buf) - 1 - total)) {
        if (ret == -1) {
            exit_with_error("Server error with read");
        }
        total += ret;
    }

    buf[total] = 0;
    printf("Server received: %s, total = %d\n", buf, total);
}
```

IPv4 socket address structure

```
struct in_addr {
    in_addr_t s_addr;
};

struct sockaddr_in {
    sa_family_t sin_family;
    in_port_t sin_port;
    struct in_addr sin_addr;
    unsigned char __pad[X];
};
```

#include <sys/socket.h>

int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);

Returns 0 on success, or -1 on error

#include <sys/socket.h>

int listen(int sockfd, int backlog);

Returns 0 on success, or -1 on error

```
#define __ss_aligntype uint32_t
struct sockaddr_storage {
    sa_family_t ss_family;
    __ss_aligntype __ss_align;
    char __ss_padding[SS_PADSIZE];
};
```

#include <sys/socket.h>

int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);

Returns file descriptor on success, or -1 on error

Generic socket address structure

```
struct sockaddr {
    sa_family_t sa_family;
    char sa_data[14];
};
```

Large enough for IPv4 or IPv6

Client-Server Example

```
void client(int port)
{
    int fd;
    struct sockaddr_in in_addr;
    memset(&in_addr, 0, sizeof(struct sockaddr_in));

    if ((fd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        exit_with_error(0);
    }

    in_addr.sin_family = AF_INET;
    in_addr.sin_port = 5001;
    inet_pton(AF_INET, "127.0.0.1", &in_addr.sin_addr);

    if (connect(fd, (struct sockaddr *)&in_addr, sizeof(struct sockaddr_in))) {
        exit_with_error("Client error with connect");
    }

    int ret, sent = 0;
    char *msg = "Hello, server!";
    while (sent != strlen(msg)) {
        ret = write(fd, msg + sent, strlen(msg) - sent);
        if (ret == -1)
            exit_with_error("Client error with write");
        else
            sent += ret;
        printf("sent = %d\n", sent);
    }
}
```

```
void exit_with_error(char *msg)
{
    perror(msg);
    exit(1);
}
```

IPv4 socket address structure

```
struct in_addr {
    in_addr_t s_addr;
};

/* IPv4 4-byte address */
/* Unsigned 32-bit integer */

struct sockaddr_in {
    sa_family_t sin_family;
    in_port_t sin_port;
    struct in_addr sin_addr;
    unsigned char __pad[X];
};

/* IPv4 socket address */
/* Address family (AF_INET) */
/* Port number */
/* IPv4 address */
/* Pad to size of 'sockaddr'
   structure (16 bytes) */
```

#include <arpa/inet.h>

int inet_pton(int domain, const char *src_str, void *addrptr);

Returns 1 on successful conversion, 0 if *src_str* is not in presentation format, or -1 on error

#include <sys/socket.h>

int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);

Returns 0 on success, or -1 on error

Basic write() library call uses a generic file descriptor

Print the last error encountered during a system call or library function

Protocols and Communication

