Computer Networks

EDA387/DIT661

Socket API

Part 2

Outline

- The Daytime server example
- Accept, Connection, Send and Reply
- TCP State Transition
- Port Numbers
- Summary

A Simple Daytime Client

```
int main(int argc, char **argv) {

    int     sockfd, n;  char    recvline[MAXLINE + 1];

      struct sockaddr_in servaddr;

    if (argc != 2)   err_quit("usage: a.out <IPaddress>");

    if ( (sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)   err_sys("socket error");

    bzero(&servaddr, sizeof(servaddr));

    servaddr.sin_family = AF_INET;

    servaddr.sin_port = htons(13);  /* daytime server */

   if (inet_pton(AF_INET, argv[1], &servaddr.sin_addr) <= 0)

     err_quit("inet_pton error for %s", argv[1]);
```

A Simple Daytime Client

```
    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);

}
```

Daytime Client: TCP socket

Self-study

- sockfd = socket(AF_INET, SOCK_STREAM, 0) creates TCP socket
  - Returns a small integer descriptor used for identification
  - The call to socket can fail, why?

- We will encounter many different uses of the term "socket"
  1. The API that we are using is called the sockets API
  2. The function named socket that is part of the sockets API
  3. The TCP socket, which is a TCP endpoint

Reading the Server's Reply

Self-study

- Must be careful when using TCP because it is a byte-stream protocol with no record boundaries

Mon May 26 20 : 58 : 40 2003\r\n

(\r carriage return and \n linefeed)

  - Can be returned in numerous ways:
    - Normally, 1 TCP segment of all 26 bytes,
    - can also be 26 TCP segments each containing 1 byte of data, etc
  - Always need to code the read in a loop and terminate the loop when either read returns 0 (i.e., the other end closed the connection) or a value less than 0 (an error)
  - This technique is also used by version HTTP 1.0

A Simple Daytime Server

```
int main(int argc, char **argv) {

   int listenfd, connfd;

   struct sockaddr_in servaddr;

   char    buff[MAXLINE]; time_t ticks;

   listenfd = Socket(AF_INET, SOCK_STREAM, 0);

   bzeros(&servaddr, sizeof(servaddr));

   servaddr.sin_family = AF_INET;

   servaddr.sin_addr.s_addr = htonl(INADDR_ANY);

   servaddr.sin_port = htons(13); /* daytime server */
```

A Simple 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);

}}
```

Outline

- The Daytime server example
- Accept, Connection, Send and Reply
- TCP State Transition
- Port Numbers
- Summary

Accept Connection & Send Reply

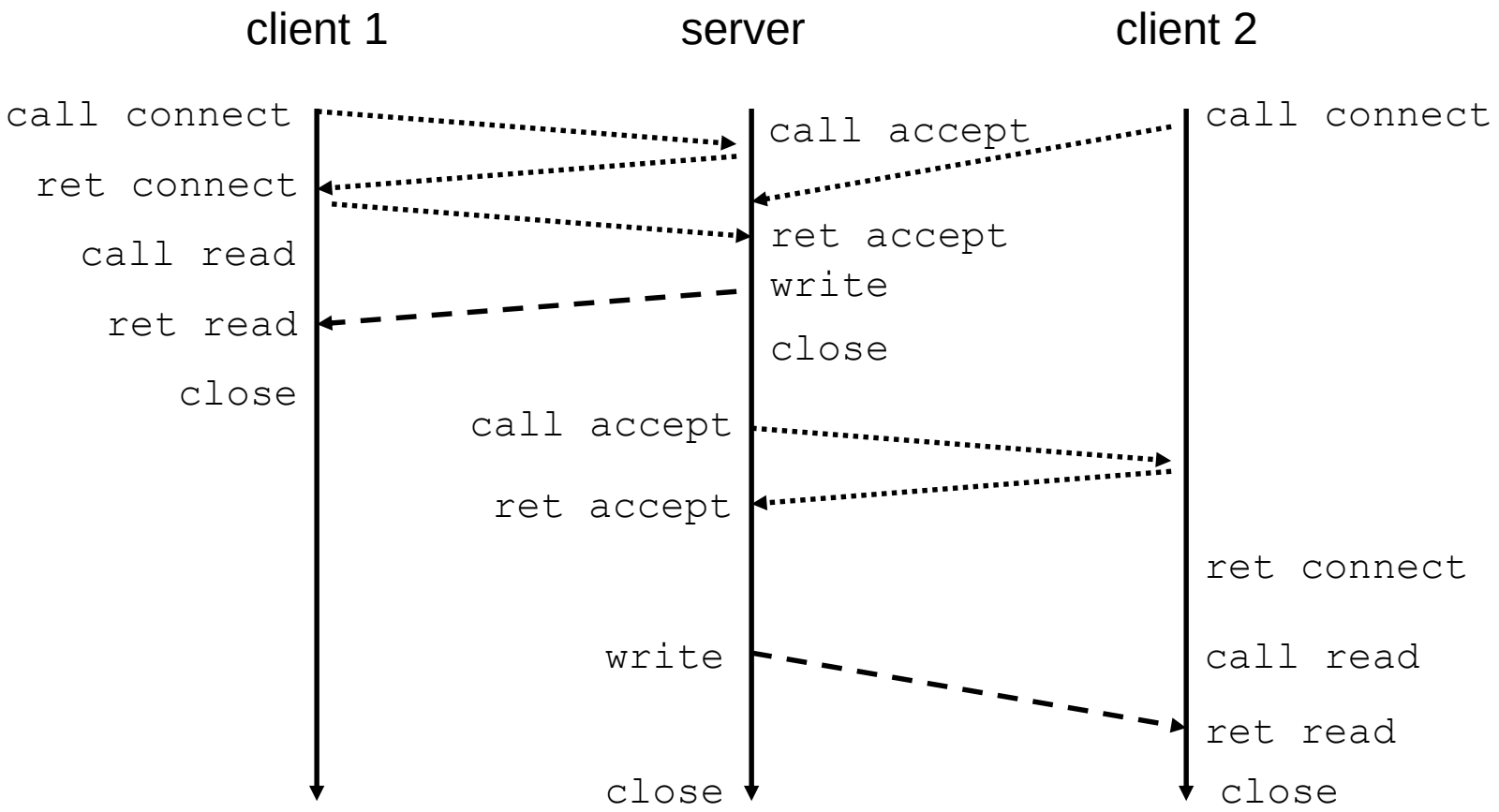
- The server process is put to sleep in the call to accept, waiting for connections to arrive and be accepted
  - TCP connections use a three-way handshake to establish a connection
  - When this handshake completes, accept returns, and the return value from the function is a new descriptor (connfd) that is called the connected descriptor
  - This new descriptor is used for communication with the new client

Accept Connection & Send Reply

- The server handles only one client at a time
  - If multiple client connections arrive at about the same time, the kernel queues them, up to some limit, and returns them to accept one at a time
  - The daytime server is quite fast
  - But if the server took more time to service each client, we would need some way to overlap the service of one client with another client
- This design is called *the iterative server*

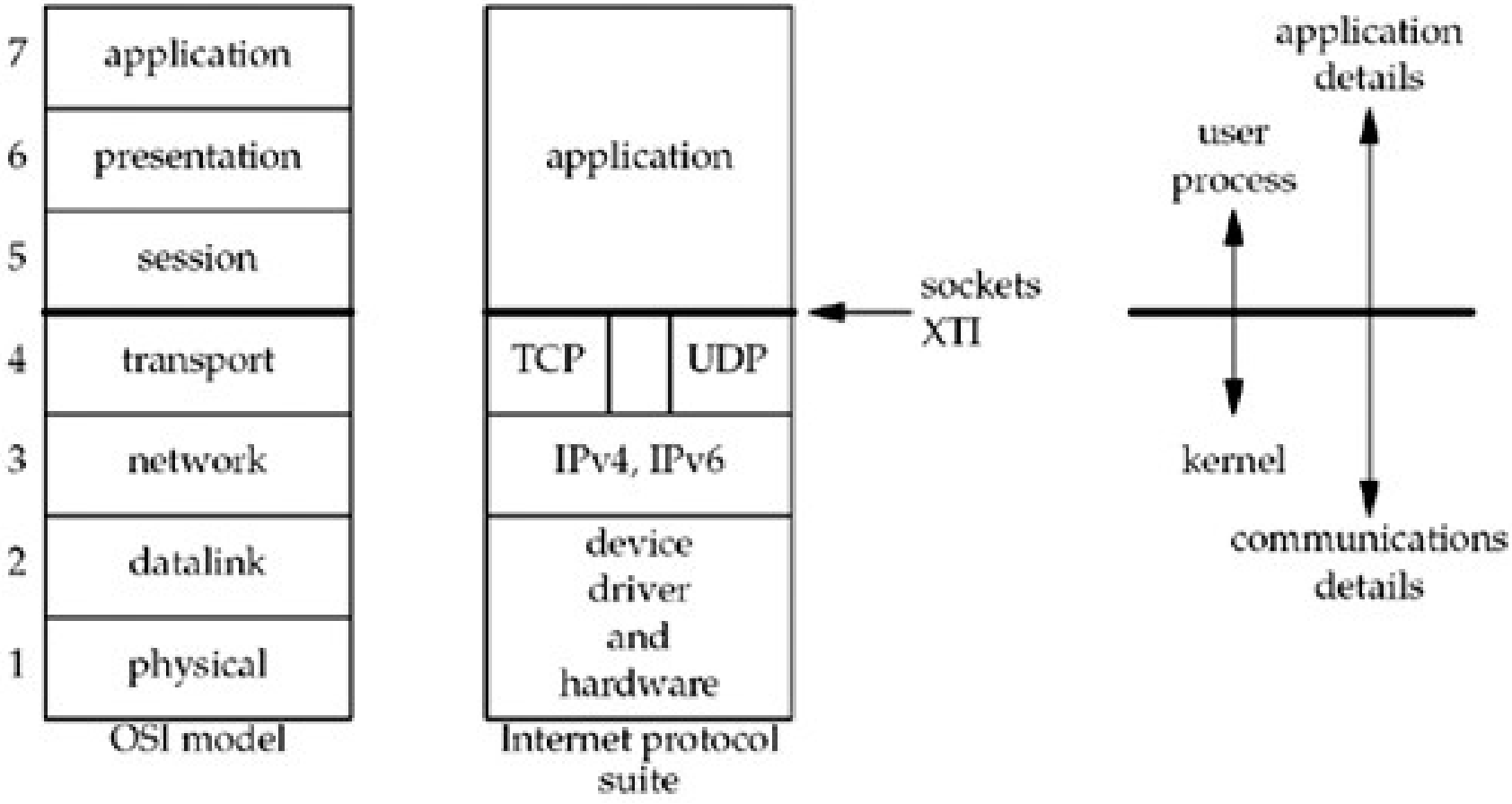
Iterative Servers

- Iterative servers process one request at a time.



OSI Model

Review



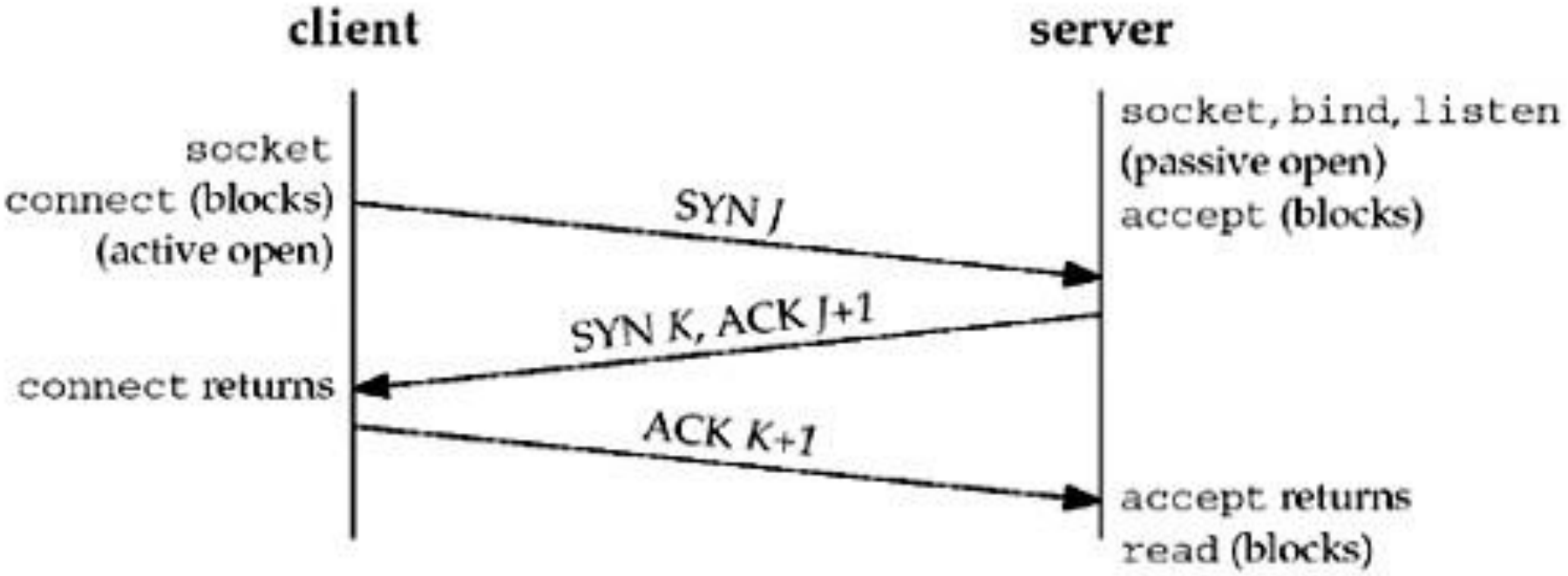
TCP: Establishment & Termination

Review

- To understanding of the connect, accept, and close functions, we must understand how:
  - TCP connections are established and terminated
  - TCP's state transition diagram.

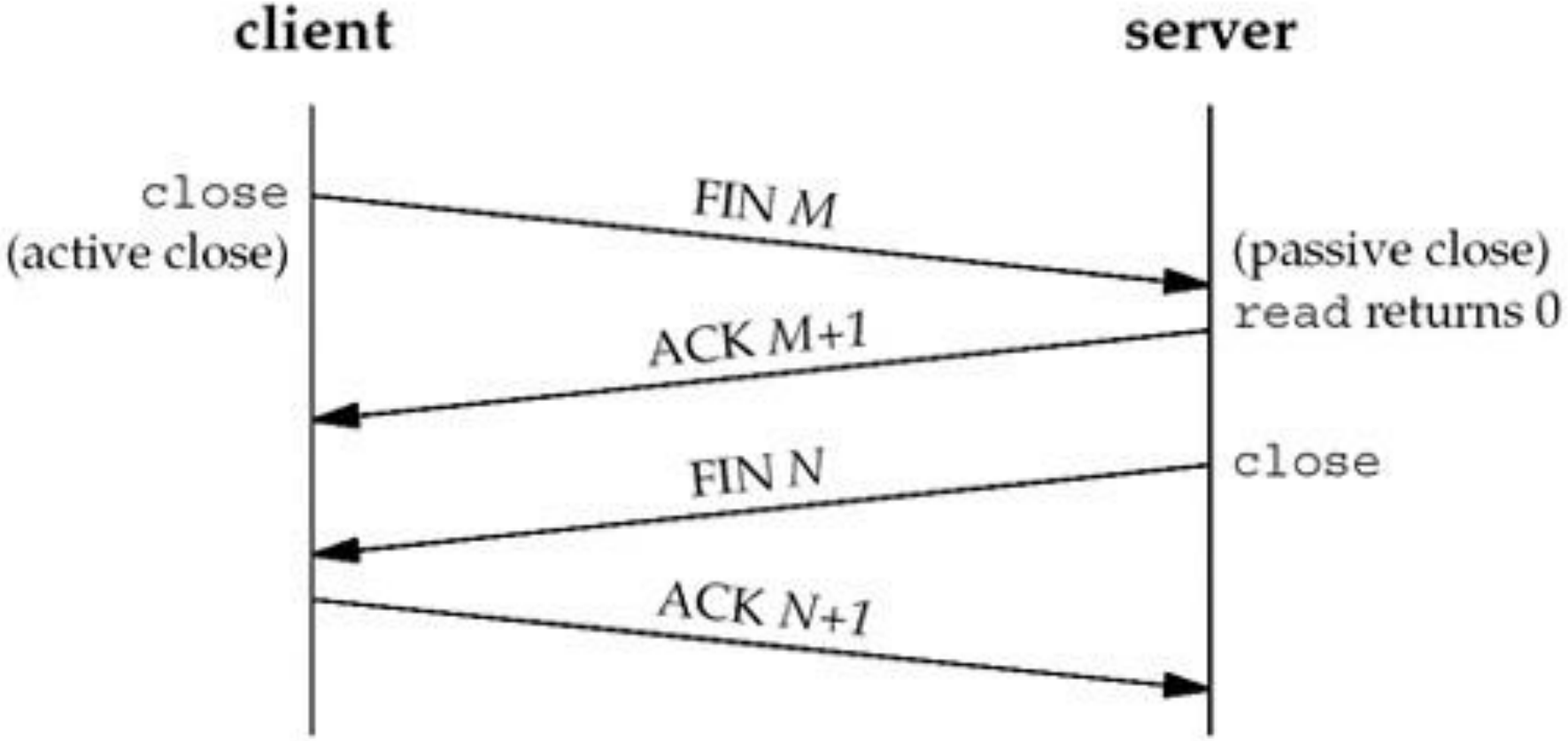
Three-way Handshake

Review



TCP Connection Termination

Review

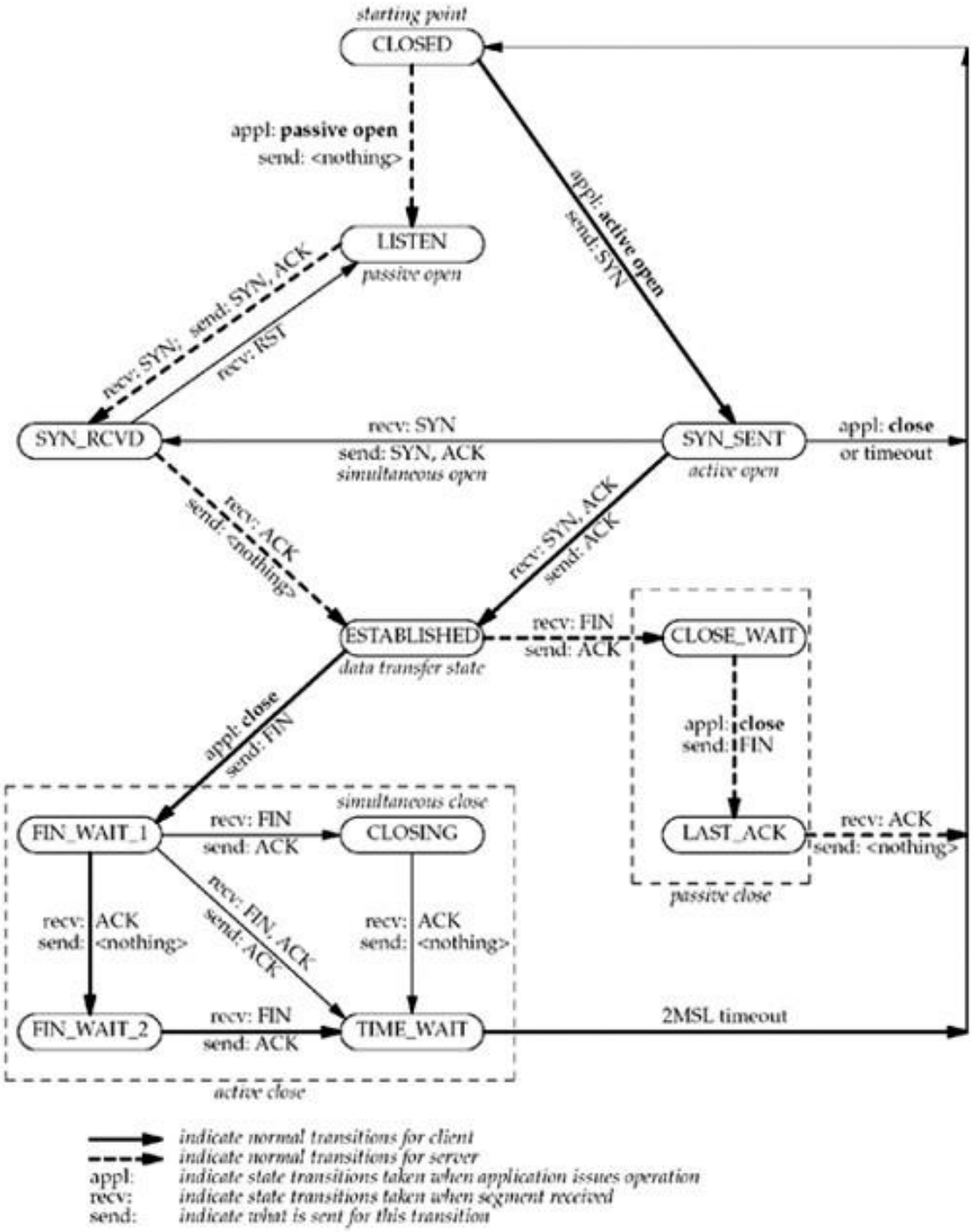


Outline

- The Daytime server example
- Accept, Connection, Send and Reply
- TCP State Transition
- Port Numbers
- Summary

TCP State Transition

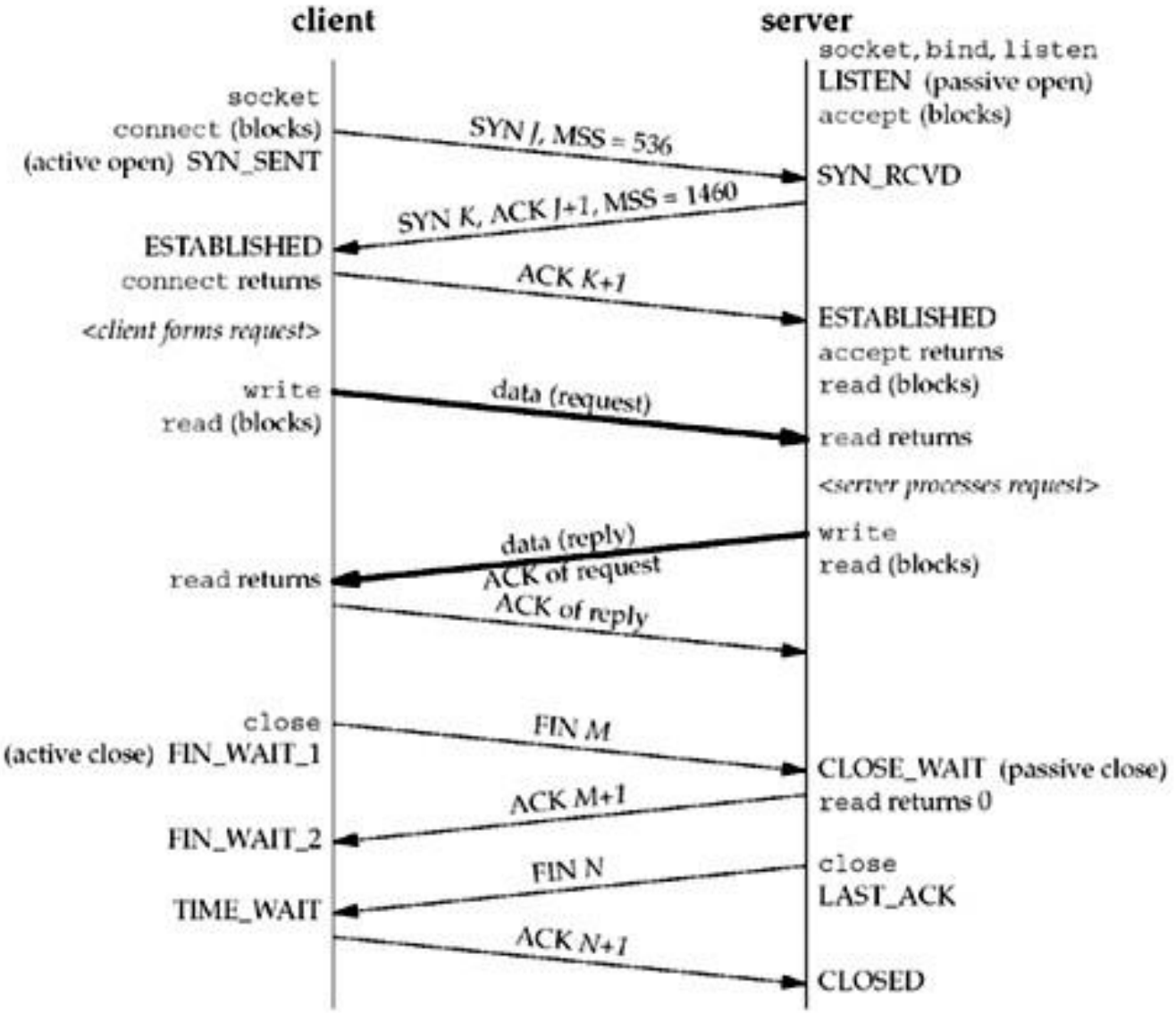
- If an application performs an active open in the CLOSED state, TCP sends a SYN and the new state is SYN_SENT
- If TCP next receives a SYN with an ACK, it sends an ACK and the new state is ESTABLISHED



Watching the Packets

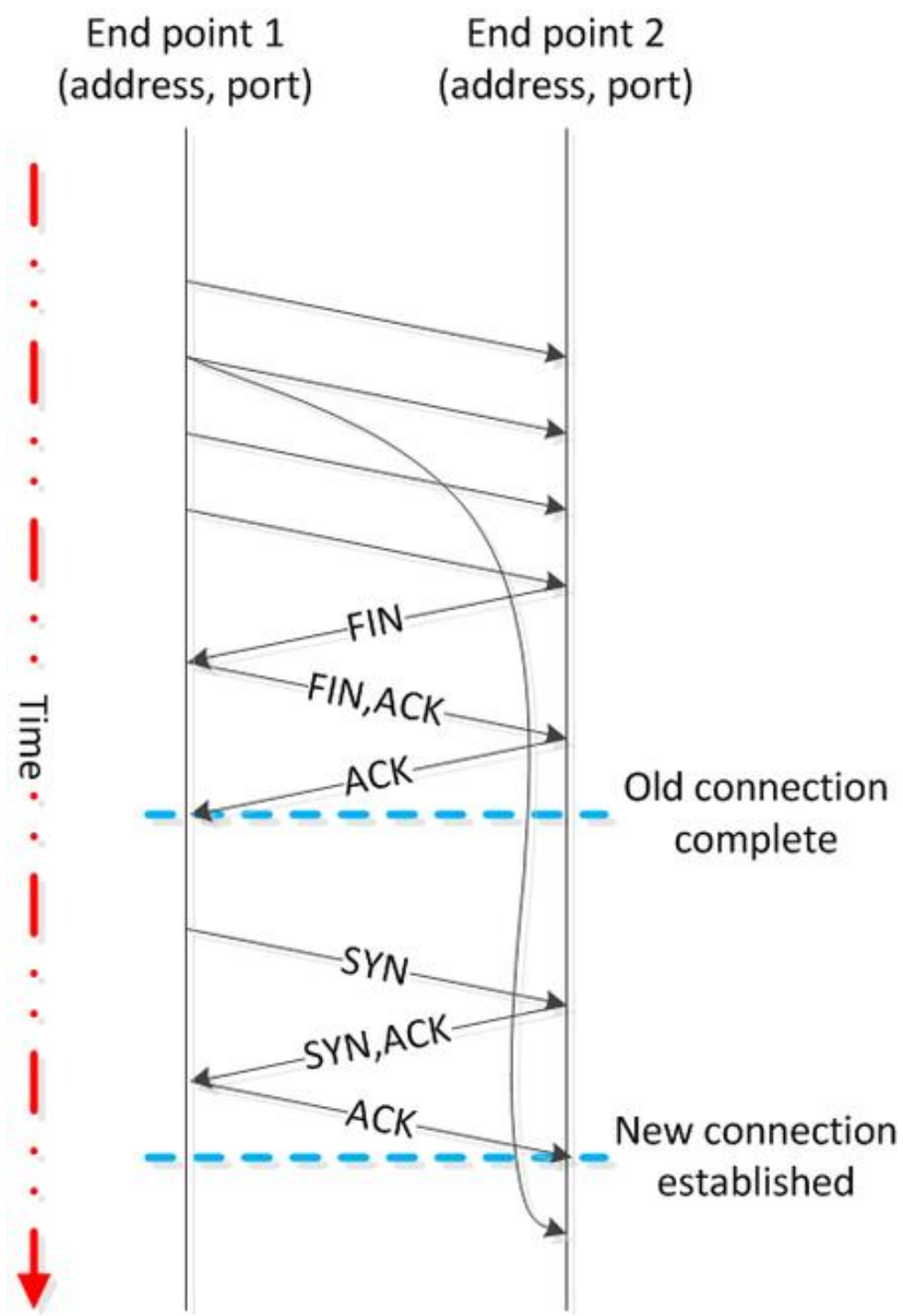
Self-study

- *netstat*: a tool for displaying network connections, routing tables, and a number of network interface statistics
- Try it!



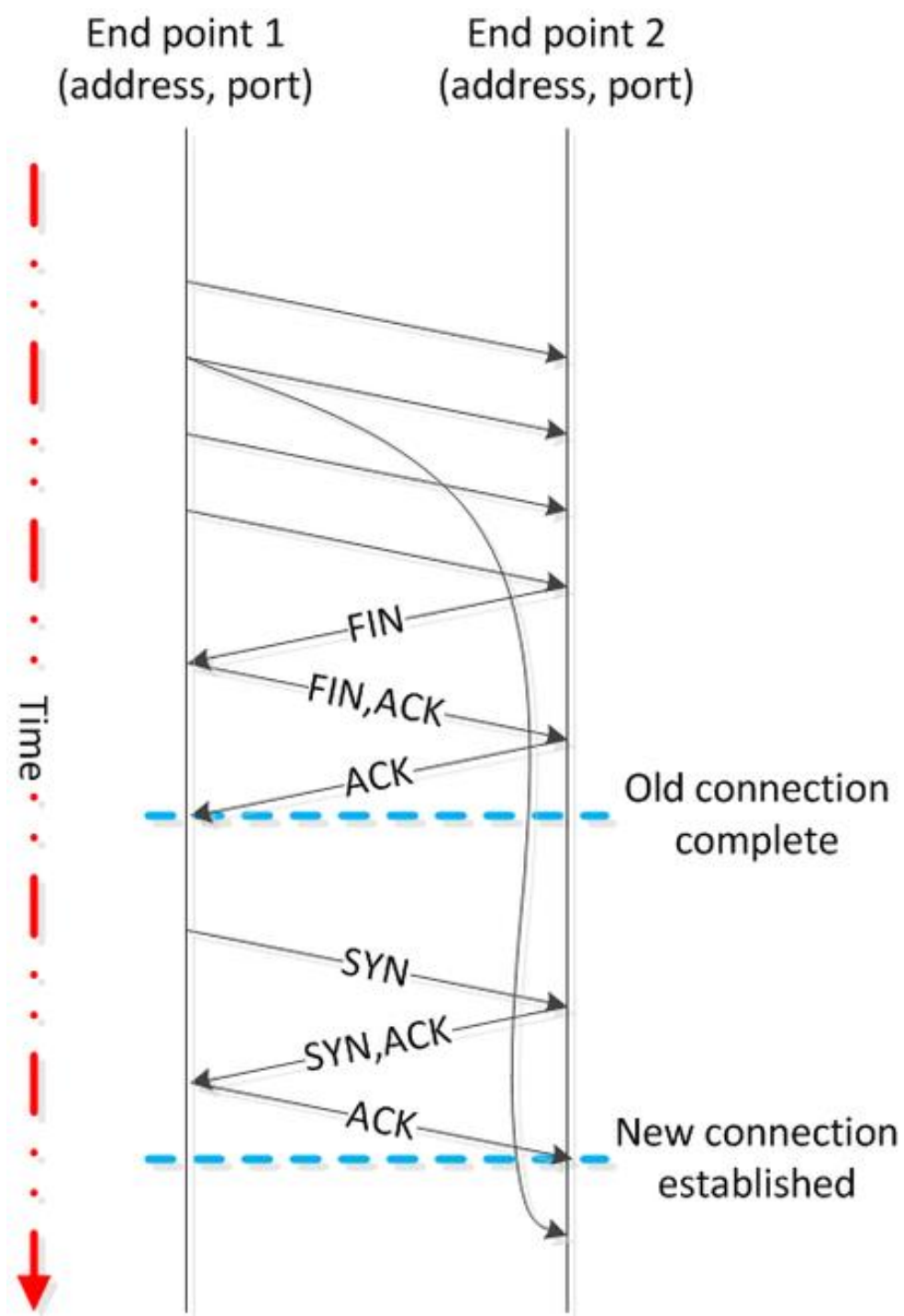
TIME_WAIT State

- An active-close-end goes through this state
  - remains in this state is 2X the maximum segment lifetime
  - MSL= max time that any IP datagram can live in a network
    - It is bounded: every datagram contains an 8-bit hop limit
  - recommended value (RFC1122) is 2 min, BSD uses 30 sec.
  - means TIME_WAIT duration is between 1 and 4 min.
- Packet lost is usually the result of routing problems
  - Once routing problem is corrected and the packet that was lost in the loop is sent to the final destination
  - Happens within MSL time



TIME_WAIT

- Suppose that end point 2 wasn't in TIME_WAIT state when a delayed segment arrives.
- That segment is mistaken for part of the second connection (when it has an appropriate sequence numbers) .



TIME_WAIT

- When the final ACK (end point 2) omitted, end point 1 will resend the final FIN.
- If end point 2 state is CLOSED, it sends RST (unexpected FIN).
- End point 1 receive an error even though all data was received correctly.

Expiring TCP duplicates

- Initiate a new incarnation of a connection that is currently in the TIME_WAIT state
- Since the duration of the TIME_WAIT state is twice the MSL, this allows MSL sec. for a packet in one direction to be lost, and another MSL sec. for the reply to be lost
- Guaranteed that when establishing a connection, all old duplicates from previous incarnations of the connection have expired in the network

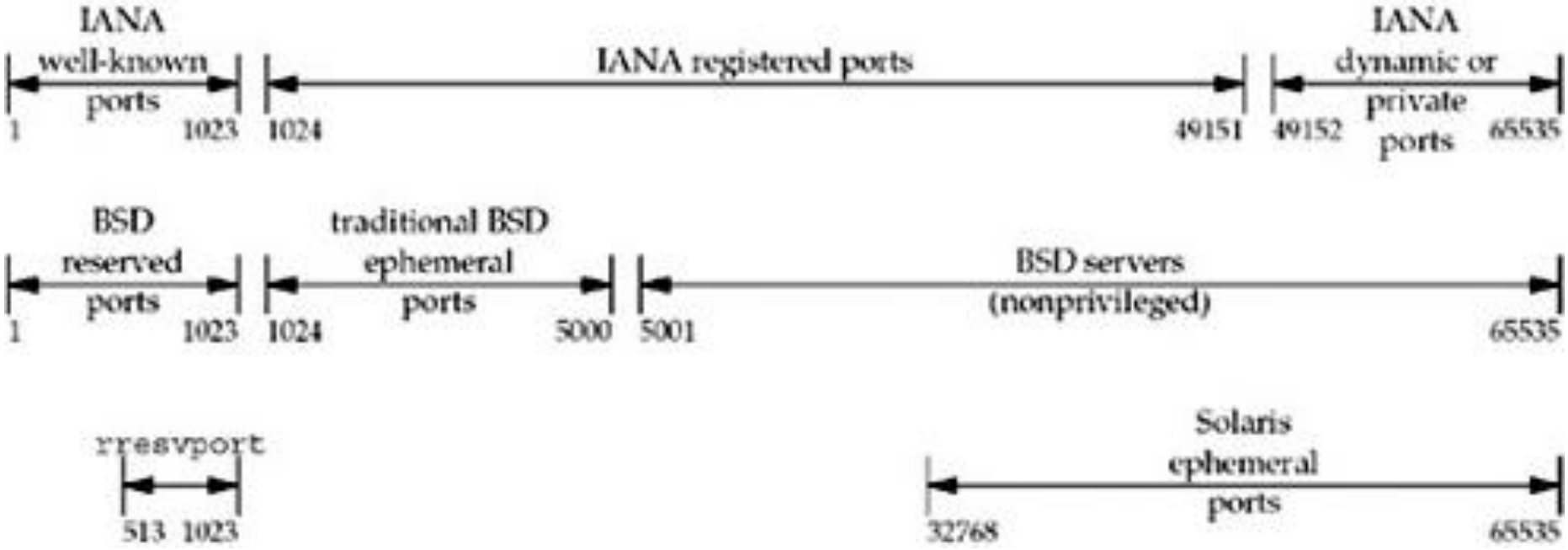
Outline

- The Daytime server example
- Accept, Connection, Send and Reply
- TCP State Transition
- Port Numbers
- Summary

Port Numbers

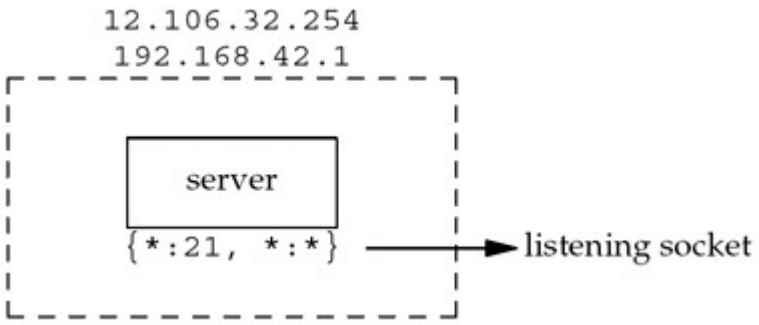
Review

1. Well-known: Internet Assigned Numbers Authority
2. Registered: IANA only lists the uses of these ports
3. Dynamic or Private: not control (ephemeral ports)



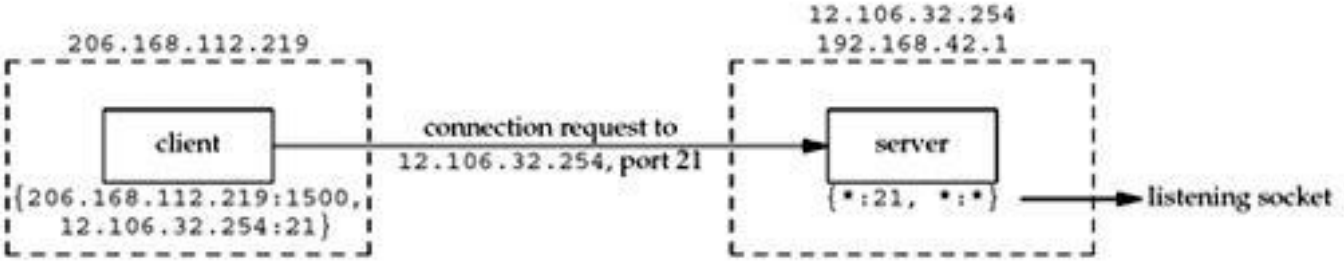
TCP Port Numbers and Concurrent Servers

- Lets consider a multihomed host with 2 IP addresses 12.106.32.254 and 192.168.42.1, and the server does a passive open using its well-known port, e.g., 21

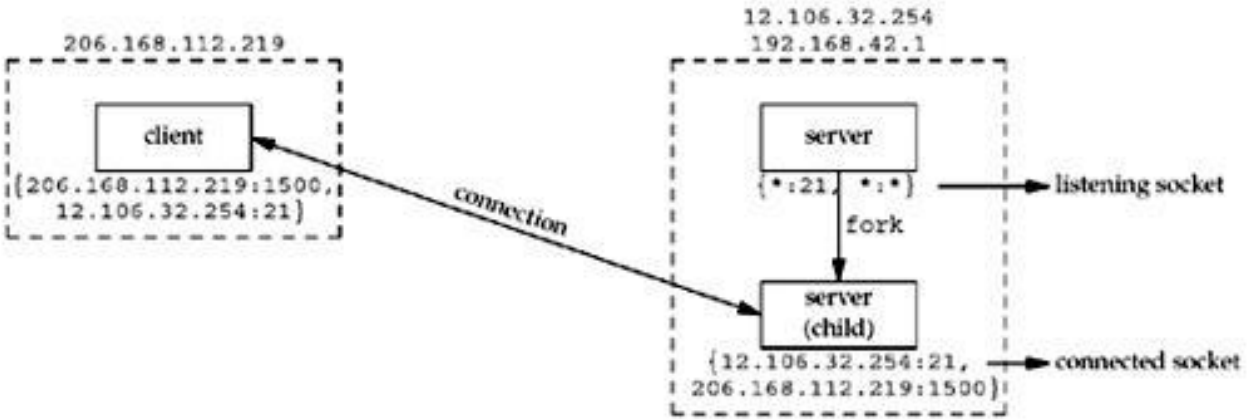


- {*:21, *:*} indicates the server's socket pair
- Server waits for connection requests on local interfaces (1st *) and port 21

TCP Port Numbers and Concurrent Servers



- When the server receives and accepts the client's connection, it forks a copy of itself, letting the child handle the client

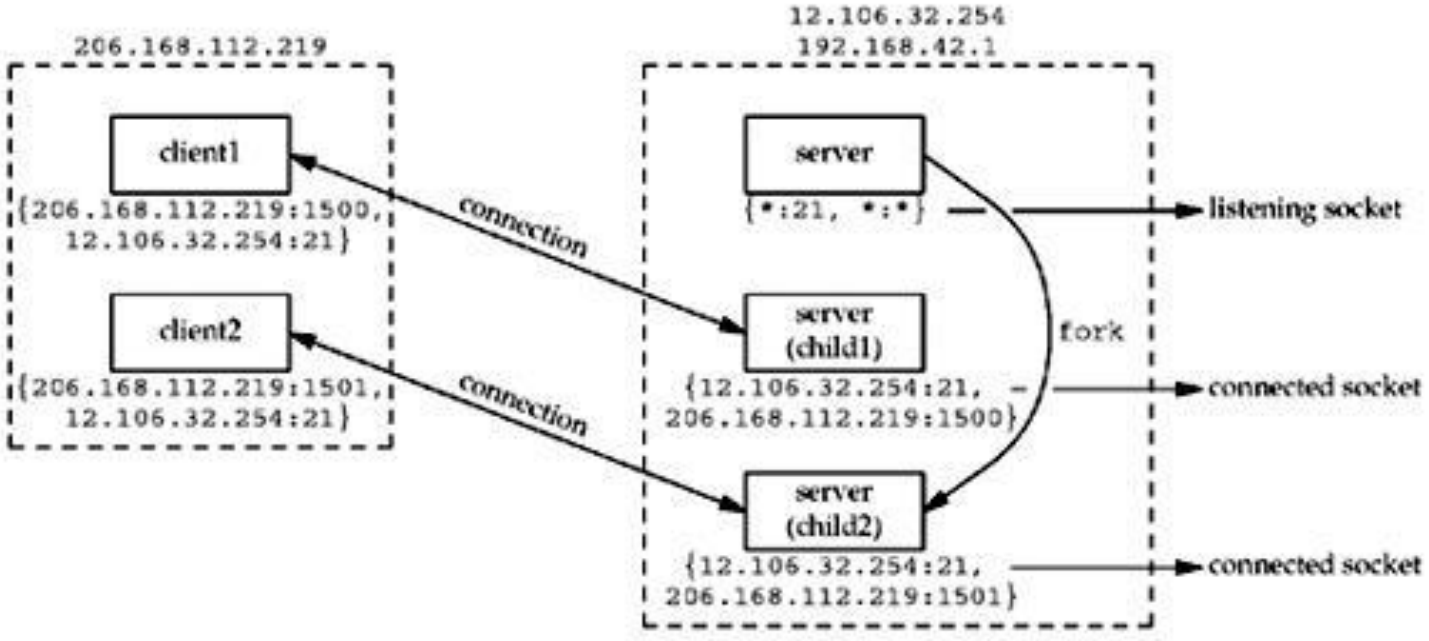


TCP Port Numbers and Concurrent Servers

- Server host must distinguish between the listening socket and the connected socket
- Connected socket uses the same local port (21) as the listening socket
- The multihomed server, the local address is filled in for the connected socket (12.106.32.254) once the connection is established
- Assume that another client process on the client host requests a connection with the same server

TCP Port Numbers and Concurrent Servers

- TCP code on the client host assigns the new client socket an unused ephemeral port number, e.g.,1501
- On the server, the two connections are distinct



TCP Port Numbers and Concurrent Servers

- TCP cannot demultiplex incoming segments by looking at just the destination port number

- TCP must look at all four elements in the socket pair to determine which endpoint receives an arriving segment

Outline

- The Daytime server example
- Accept, Connection, Send and Reply
- TCP State Transition
- Port Numbers
- Summary

Summary

- We have seen our first client-server program
- We reviewed the TCP protocol, its state machine and the use of the socket pair

Next Lecture

We are going to look into the way that IP addresses are encoded and review our knowledge about byte ordering issues.

Then we will look at the socket box, before learning more of what is inside the box and how the socket API interacts with the kernel.

Moreover, we are going to discover more about the limitations of iterative servers.

Review Questions

- Someone has suggested shortening the TIME_WAIT state duration. What could be the outcome of this suggestion?

1. Shorter process block time when calling the close system call, but only for actively closing communication end, which is always the server
2. Process state corruption due to unexpected communication received from the other end
3. Shorter process block time when calling the close system call, but only for passively closing communication end
4. The processes can block forever on the listen system call
5. None of the above

Review Questions

How many simultaneous socket connections possible?

1. A constant number that is defined by the port number
2. At most two, one for the listening socket and one for the connection socket
3. A variable number that depends merely on the amount available memory
4. One in the case of a client process and two in the case of a server process
5. None of the above

Review Questions

- Which process starts TCP's three way handshake

1. Either the server or the client after calling the connect system call
2. The client after calling the socket system call
3. The server after calling the close system call
4. The client after calling the close system call
5. The server after calling the accept system call
6. Either the server or the client after calling the close system call
7. None of the above

Review Questions

- Which process starts TCP's four way handshake?

1. The server after calling the close system call
2. Either the server or the client after calling the close system call
3. Either the server or the client after calling the connect system call
4. The client after calling the socket system call
5. The client after calling the close system call
6. The server after calling the accept system call
7. None of the above