

CS 5523 Lecture 7: An overview of UDP

- *UDP overview*
- *Common UDP applications*
- *UDP communication models*
- *UDP client-server communication in UNIX*
- *UDP client-server communication in JAVA*
- *UDP failure models*

UDP overview

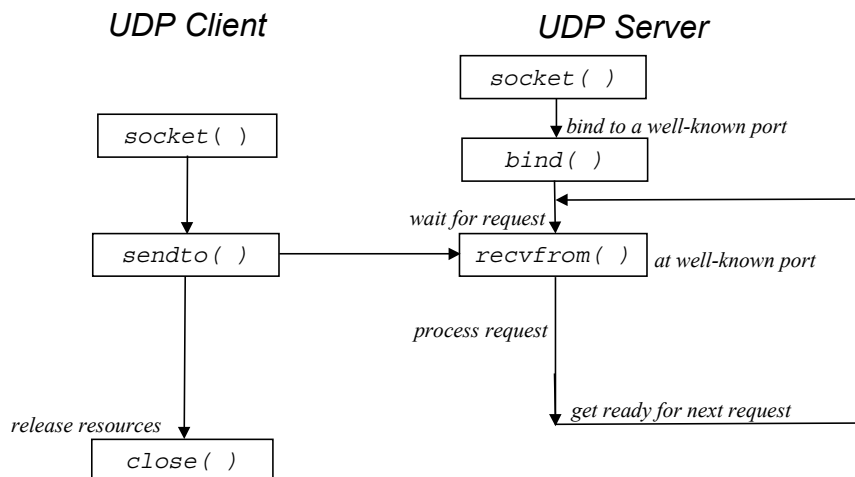
- *abstraction is that of a datagram*
- *sender must first create a socket bound to a local port and local IP address*
- *sender can use the socket to send to any destination*
- *destination (host, port) is explicitly included in each message*
- *the receiver must have a socket bound to the specified port*

What is the datagram abstraction?

Common Internet applications that use UDP:

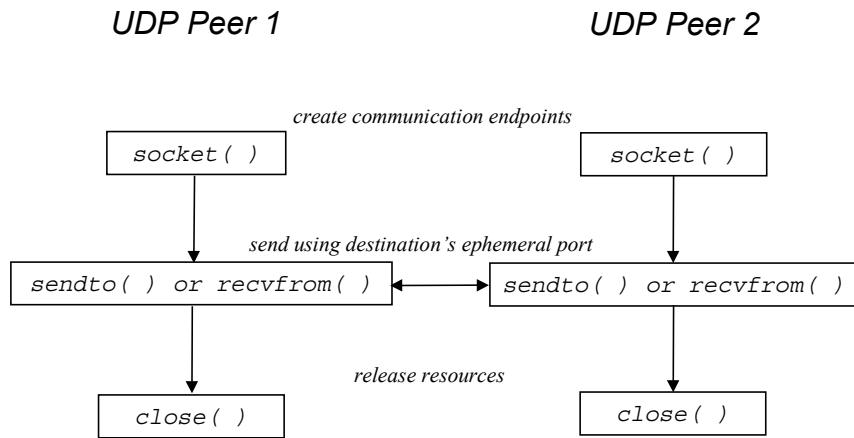
- *Traceroute*
- *RIP (routing)*
- *BOOTP (bootstrap protocol)*
- *DHCP (bootstrap protocol)*
- *NTP (time protocol)*
- *TFTP (trivial FTP)*
- *SNMP (network management)*
- *DNS (name service)*
- *NFS (distributed file system)*
- *Sun RPC (remote procedure call)*
- *DCE RPC (remote procedure call)*

UDP communication model 1 (one-way)



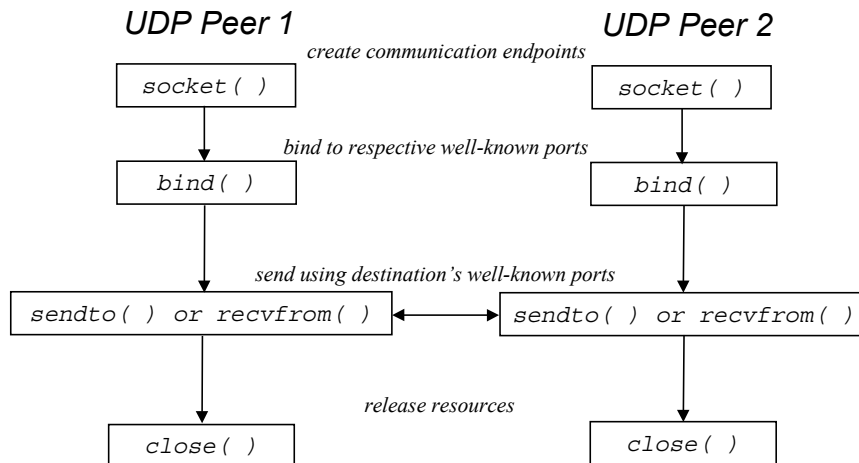
*The client uses the server's well-known port.
The server uses client address from request for response.*

UDP communication model 4 (Peer-peer)



The peers must communicate their ephemeral ports

UDP communication model 5 (Peer-peer)



The peers must register their well-known ports

Problems with UDP models 1-5

- *Well-known ports must be registered.*
- *Peer-to-peer models don't adapt well to well-known ports.*
- *Must have a side mechanism for determining ephemeral port numbers.*
- *Multiprocessor servers could better handle large loads by monitoring multiple ports.*

Alternative mechanisms - where the communication endpoint is determined from the name of a service rather than a port - are being standardized (e.g., `getservbyname()`, `getservbyport()`, and `getaddrinfo()`).

Client in UDP communication model 1 - Unix

```
#include <errno.h>          /* for perror          */
#include <netdb.h>          /* for gethostbyname   */
#include <netinet/in.h>     /* for sockaddr_in, INADDR_ANY */
#include <stdlib.h>         /* for atoi, exit      */
#include <stdio.h>          /* for fprintf, perror */
#include <string.h>         /* for memcpy          */
#include <sys/socket.h>     /* for socket, bind, socklen_t, recvfrom */
                          /* AF_INET, SOCK_DGRAM */
#include <sys/types.h>      /* for socket, bind, uint16_t, ssize_t */
#include <unistd.h>         /* for read, write, close */
#define BLKSIZE 1024

void main(int argc, char *argv[])
{
    uint16_t port;
    int sock;
    struct sockaddr_in server;
    struct hostent *hp;
    ssize_t bytesread;
    ssize_t byteswritten;
    char buf[BLKSIZE];
    if (argc != 3) {
        fprintf(stderr, "Usage: %s host port\n", argv[0]);
        exit(1);
    }
}
```

Client in UDP communication model 1 (continued)

```
/* set up the address to the server to be included in datagram */
server.sin_family = AF_INET;
port = (uint16_t)atoi(argv[2]);
server.sin_port = htons(port);
if (!(hp = gethostbyname(argv[1]))) {
    perror("Unable to get host byname");
    exit(1);
}
memcpy((char *)&server.sin_addr, hp->h_addr_list[0], hp->h_length);

/* create a socket for sending datagrams */
if ((sock = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
    perror("Unable to create socket");
    exit(1);
}
for ( ; ; ) {
    bytesread = read(STDIN_FILENO, buf, BLKSIZE);
    if ( (bytesread == -1) && (errno == EINTR) )
        fprintf(stderr, "Client restarting read\n");
    else if (bytesread <= 0)
        break;
    else {
        byteswritten = sendto(sock, buf, bytesread, 0, (struct sockaddr *)&server, sizeof(server));
        if (byteswritten != bytesread) {
            fprintf(stderr, "Error %ld of %ld byteswritten\n", (long)byteswritten, (long)bytesread);
            break;
        }
    }
}
close(sock);
exit(0);
}
```

Server in UDP communication model 1 - Unix

```
#include <errno.h> /* for perror */
#include <netinet/in.h> /* for sockaddr_in, INADDR_ANY */
#include <stdlib.h> /* for atoi, exit */
#include <stdio.h> /* for fprintf, perror */
#include <sys/socket.h> /* for socket, bind, socklen_t, recvfrom */
/* AF_INET, SOCK_DGRAM */
#include <sys/types.h> /* for socket, bind, uint16_t, ssize_t */
#include <unistd.h> /* for read, write */
#define BLKSIZE 1024
void main(int argc, char *argv[])
{
    uint16_t port;
    int sock;
    struct sockaddr_in server;
    struct sockaddr_in client;
    ssize_t bytesread;
    ssize_t byteswritten;
    socklen_t clientlen;
    char buf[BLKSIZE];

    if (argc != 2) {
        fprintf(stderr, "Usage: %s port\n", argv[0]);
        exit(1);
    }
}
```

Server in UDP communication model 1 (continued)

```

                                /* set up the address of the server */
server.sin_family = AF_INET;
port = (uint16_t)atoi(argv[1]);
server.sin_port = htons(port);
server.sin_addr.s_addr = htonl(INADDR_ANY);
                                /* create the socket */
if ((sock = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
    perror("Unable to create socket");
    exit(1);
}

                                /* bind the socket to a well-known port */
if (bind(sock, (struct sockaddr *)&server, sizeof(server)) < 0) {
    perror("Unable to bind well-known port to socket");
    exit(1);
}

                                /* receive from clients */
while( (bytesread = recvfrom(sock, buf, (size_t)BLKSIZE, 0,
    (struct sockaddr *)&client, &clientlen) ) > 0) {
    byteswritten = write(STDOUT_FILENO, buf, (size_t)bytesread);
    if (bytesread != byteswritten)
        fprintf(stderr, "Error %ld of %ld bytes written\n", (long)byteswritten, (long)bytesread);
}
close(sock);
exit(0);
}

```

Creating a socket and binding it to a port

```

/*
 * u_open
 * Return a file descriptor which is bound to the given port.
 *
 * parameter:
 *   port = number of port to bind to
 * returns: file descriptor if successful and -1 on error
 */
int u_open(u_port_t port){
    int sock;
    struct sockaddr_in server;
    if ( (u_ignore_sigpipe() != 0) || ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) )
        return -1;
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = INADDR_ANY;
    server.sin_port = htons((short)port);
    if ((bind(sock, (struct sockaddr *)&server, sizeof(server)) < 0) || (listen(sock, MAXBACKLOG) < 0))
        return -1;
    return sock;
}

```

Figure 4.3
UDP client sends a message to the server and gets a reply

```
import java.net.*;
import java.io.*;
public class UDPClient{
    public static void main(String args[]){
        // args give message contents and server hostname
        try {
            DatagramSocket aSocket = new DatagramSocket();
            byte [] m = args[0].getBytes();
            InetAddress aHost = InetAddress.getByName(args[1]);
            int serverPort = 6789;
            DatagramPacket request = new DatagramPacket(m, args[0].length(), aHost, serverPort);
            aSocket.send(request);
            byte[] buffer = new byte[1000];
            DatagramPacket reply = new DatagramPacket(buffer, buffer.length);
            aSocket.receive(reply);
            System.out.println("Reply: " + new String(reply.getData()));
            aSocket.close();
        } catch (SocketException e){System.out.println("Socket: " + e.getMessage());}
        } catch (IOException e){System.out.println("IO: " + e.getMessage());}
    }
}
```

Instructor's Guide for Coulouris, Dollimore and Kindberg Distributed Systems: Concepts and Design Edn. 3
© Addison-Wesley Publishers 2000

Figure 4.4
UDP server repeatedly receives a request and sends it back to the client

```
import java.net.*;
import java.io.*;
public class UDPServer{
    public static void main(String args[]){
        try{
            DatagramSocket aSocket = new DatagramSocket(6789);
            byte[] buffer = new byte[1000];
            while(true){
                DatagramPacket request = new DatagramPacket(buffer, buffer.length);
                aSocket.receive(request);
                DatagramPacket reply = new DatagramPacket(request.getData(),
                    request.getLength(), request.getAddress(), request.getPort());
                aSocket.send(reply);
            }
        } catch (SocketException e){System.out.println("Socket: " + e.getMessage());}
        } catch (IOException e) {System.out.println("IO: " + e.getMessage());}
    }
}
```

Instructor's Guide for Coulouris, Dollimore and Kindberg Distributed Systems: Concepts and Design Edn. 3
© Addison-Wesley Publishers 2000

Summary of UDP properties:

- *Message size - determined by application up to 2^{16} bytes*
- *Blocking properties are determined by socket options not UDP*
- *Timeouts - can be set on a socket to prevent indefinite waiting*
- *The recvfrom doesn't specify which senders, anyone can send*

UDP failure model:

- *Omission failures*
- *Messages can be delivered out of order*

Does UDP have integrity?

Does UDP have validity?

For next time:

- *Read Section 9.1 and 9.2*
- *Be prepared to discuss assigned questions from Chapter 3*
- *Print out the man page for nslookup:*

```
man nslookup | mpage -2 | lp
```