

CS 5523 Lecture 5: TCP

- Questions on Laboratory 1
- IP in a Nutshell
- TCP abstractions
- Applications that use TCP
- Ports and sockets
- TCP protocol
- Unix examples
- Java examples
- TCP protocol details

IP layer in a nutshell:

- transmits datagrams from one host to another
- unreliable or best effort delivery
- no data checksum (sound like Protocol A of problem 2.14?)
- puts IP datagrams into network packets
- might break up if IP datagram longer than MTU of network
- packets have fragment identifiers
- physically transmits to next link after resolving network address
- routing algorithms decide what physical link to send on

Figure 3.13
Encapsulation in a message transmitted via TCP over an Ethernet

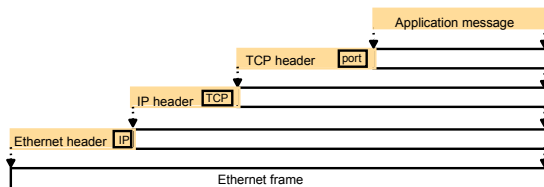


Figure 3.16
Decimal representation of Internet addresses

	octet 1	octet 2	octet 3	Range of addresses
Class A:	Network ID 1 to 127	0 to 255	Host ID 0 to 255	1.0.0.0 to 127.255.255.255
Class B:	Network ID 128 to 191	0 to 255	Host ID 0 to 255	128.0.0.0 to 191.255.255.255
Class C:	Network ID 192 to 223	0 to 255	Host ID 1 to 254	192.0.0.0 to 223.255.255.255
Class D (multicast):	Multicast address			224.0.0.0 to 239.255.255.255
Class E (reserved):	240 to 255	0 to 255	0 to 255	1 to 254 128.0.0.0 to 247.255.255.255

Figure 3.17
IP packet layout

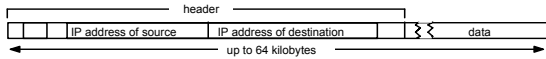


Figure 3.18
IPv6 header layout

Version (4 bits)	Priority (4 bits)	Flow label (24 bits)	
Payload length (16 bits)		Next header (8 bits)	Hop limit (8 bits)
Source address (128 bits)			
Destination address (128 bits)			

Life as an IP packet:

Imagine you were travelling in a car to a destination and each time that you had to make a turn, you would have to start over (from the beginning) if more than two other cars were at the intersection.

TCP abstractions:

- *abstraction of a stream of bytes*
- *a connection is established before messages are sent*
- *assumes one process is the client and one is the server in establishing a connection*
- *messages are sent using handles rather than source-destination addresses*

What characteristics of the network are hidden by the stream abstraction? How?

Common Internet applications that use TCP:

- *BGP (routing)*
- *SMTP (email)*
- *Telnet (remote login)*
- *FTP (file transfer)*
- *HTTP (web)*
- *NNTP (network news)*
- *DNS (name service)*
- *NFS (distributed file system)*
- *Sun RPC (remote procedure call)*
- *DCE RPC (remote procedure call)*

Ports:

- a message destination specified by a small integer (16 bits)
- any process can send a message to it
- internet protocols use the combination (IP address, local port)
- IANA (Internet Assigned Numbers Authority) ports:
 - well-known ports: 1 - 1023
 - registered ports: 1024 - 49151
 - dynamic or private ports: 49152 - 65535
- ICANN is new authority for naming and numbering on the Internet as of 1998

The socket abstraction

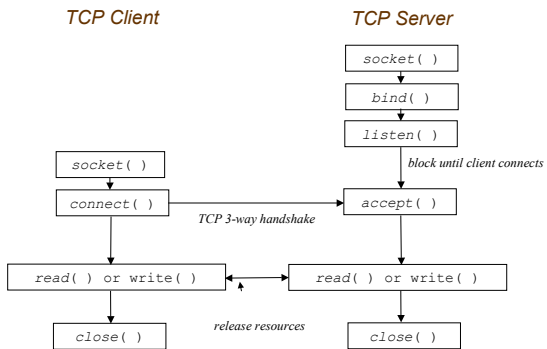
- provide endpoints for communication between processes
- a socket must be bound to a local port
- socket pair - (local IP address, local port, foreign IP address, foreign port) uniquely identifies a communication
- socket pairs uniquely identify communication

Figure 4.5 TCP client in Java (updated)

```
import java.net.*;
import java.io.*;

public class TCPClient {
    public static void main (String args[]) {
        Socket s = null;
        try {
            int serverPort = 10035;
            s = new Socket(args[1], serverPort);
            DataInputStream in = new DataInputStream( s.getInputStream());
            DataOutputStream out =new DataOutputStream( s.getOutputStream());
            out.writeUTF(args[0]);          // UTF is a string encoding see Sn. 4.4
            String data = in.readUTF();     // read a line of data from the stream
            System.out.println("Received: "+ data) ;
        } catch (UnknownHostException e) { System.out.println("Socket:"+e.getMessage()); }
        } catch (EOFException e) { System.out.println("EOF:"+e.getMessage()); }
        } catch (IOException e) { System.out.println("readline:"+e.getMessage()); }
        } finally {
            if (s!=null)
                try {
                    s.close();
                } catch (IOException e) { System.out.println("close:"+e.getMessage()); }
            }
        }
    }
}
```

TCP protocol (Unix)



Client in UNIX

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include "uici.h"
int copyfile(int fromfd, int tofd);

int main(int argc, char *argv[])
{
    int bytes_copied;
    int communfd;
    u_port_t portnumber;

    if (argc != 3) {
        fprintf(stderr, "Usage: %s host port\n", argv[0]);
        return 1;
    }
    portnumber = (u_port_t)atoi(argv[2]);
    if ((communfd = u_connect(portnumber, argv[1])) < 0) {
        fprintf(stderr, "Connection failed: %s\n", u_strerror(communfd));
        return 1;
    }
    fprintf(stderr, "[%ld]: A connection has been made to %s\n",
            (long)getpid(), argv[1]);
    bytes_copied = copyfile(STDIN_FILENO, communfd);
    close(communfd);
    fprintf(stderr, "[%ld]: Bytes sent = %d\n", (long) getpid(), bytes_copied);
    return 0;
}
```

Connection on the client side

```
#include <types.h>
#include <errno.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include "uici.h"

int u_connect(u_port_t port, char *hostn)
{
    struct hostent *hp;
    int retval;
    struct sockaddr_in server;
    int sock;
    if (isdigit((int)(*hostn))) {
        server.sin_addr.s_addr = inet_addr(hostn);
    }
    else {
        hp = gethostbyname(hostn);
        if (hp == NULL) {
            return U_BROST;
        }
        memcpy((char *)&server.sin_addr, hp->h_addr_list[0], hp->h_length);
    }
    server.sin_port = htons((short)port);
    server.sin_family = AF_INET;
```

Connection on the client side (continued)

```
if ( (u_ignore_sigpipe() != 0) ||
      (sock = socket(AF_INET, SOCK_STREAM, 0) < 0) )
    return -1;

while ( ((retval =
         connect(sock, (struct sockaddr *)&server, sizeof(server))) == -1)
        && (errno == EINTR) )
    ;

if (retval == -1) {
    close(sock);
    return U_ECONNECTION;
}
return sock;
}
```

Figure 4.5 TCP server in Java (updated)

```
import java.net.*;
import java.io.*;

public class TCPServer {
    public static void main (String args[]) {
        try {
            int serverPort = 10035; // the server port
            ServerSocket listenSocket = new ServerSocket(serverPort);
            while(true) {
                Socket clientSocket = listenSocket.accept();
                Connection c = new Connection(clientSocket);
            } catch(IOException e) {System.out.println("Listen socket:"+e.getMessage());}
        }
    }
}
```

Figure 4.5 TCP server in Java (continued)

```
class Connection extends Thread {
    DataInputStream in;
    DataOutputStream out;
    Socket clientSocket;

    public Connection (Socket aClientSocket) {
        try {
            clientSocket = aClientSocket;
            in = new DataInputStream( clientSocket.getInputStream());
            out = new DataOutputStream( clientSocket.getOutputStream());
            this.start();
        } catch(IOException e) {System.out.println("Connection:"+e.getMessage());}
    }

    public void run(){
        try { // an echo server
            String data = in.readUTF(); // read a line of data from the stream
            out.writeUTF(data);
            System.out.println("Data from client " + data);
        } catch (EOFException e) { System.out.println("EOF:"+e.getMessage());}
        } catch(IOException e) {System.out.println("readline:"+e.getMessage());}
        } finally {
            try {
                clientSocket.close();
            } catch (IOException e){ /*close failed*/}
        }
    }
}
```

Server in UNIX (establishing the connection)

```
#include <limits.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include "uici.h"
int copyfile(int fromfd, int tofd);

int main(int argc, char *argv[])
{
    int bytes_copied;
    char client[MAX_CANON];
    int communfd;
    int listenfd;
    u_port_t portnumber;

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

    portnumber = (u_port_t) atoi(argv[1]);
    if ((listenfd = u_open(portnumber)) < 0) {
        fprintf(stderr, "Listen endpoint creation failed: %s\n",
            u_strerror(listenfd));
        return 1;
    }
}
```

Server in UNIX (continued)

```
fprintf(stderr, "[%ld]: Waiting for the first connection on port %d\n",
    (long) getpid(), (int) portnumber);

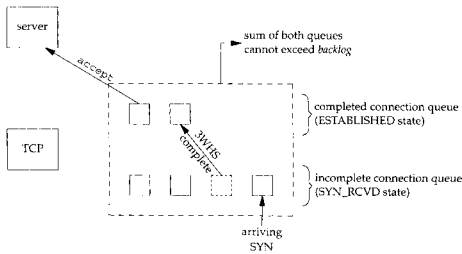
for ( ; ; ) {
    if ((communfd = u_accept(listenfd, client, MAX_CANON)) != -1) {
        fprintf(stderr, "A connection has been received from %s\n", client);
        bytes_copied = copyfile(communfd, STDOUT_FILENO);
        close(communfd);
        fprintf(stderr, "[%ld]: Bytes received = %d\n",
            (long) getpid(), bytes_copied);
    }
    else
        perror("Accept failed");
}
}
```

Creating a socket and binding it to a port

```
int u_open(u_port_t port)
{
    struct sockaddr_in server;
    int sock;
    int true = 1;

    if ( ( u_ignore_sigpipe() != 0 ) || ( sock = socket(AF_INET, SOCK_STREAM, 0) < 0 ) )
        return -1;
    if ( setsockopt(sock, SOL_SOCKET, SO_REUSEADDR, (char *)&true, sizeof(true)) < 0 ) {
        close(sock);
        return U_ESOCKOPT;
    }
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = htonl(INADDR_ANY);
    server.sin_port = htons((short)port);
    if ( bind(sock, (struct sockaddr *)&server, sizeof(server)) < 0 ) {
        close(sock);
        return U_EBIND;
    }
    if ( listen(sock, MAXBACKLOG) < 0 ) {
        close(sock);
        return U_EBACKLOG;
    }
    return sock;
}
```

TCP queues for listening socket (Stevens I Fig. 4.6):



TCP queue sizes (Stevens I Fig. 4.9 and 4.10):

kernel version	Incomplete queue	Complete queue	Maximum actual number of queued connections						
0	512	90,320	hw/bkg	AIX 4.2	DLinux 4.0, Linux 2.0.25	HP-UX 10.30	SunOS 4.1.4	Solaris 2.5.1	Solaris 2.6
1	512	90,320							
2	5085	90							
3	12,960	92							
4	11,000	36							
5	9,856	37							
6	8,750	34							
7	6,145	22							
8	4,825	30							
9	3,687	35							
10	2,672	30							
11	1,895	25							
12	1,434	24							
13	1,035	25							
14	1,035	49							
15	980	7							
16	784	7							
17	696	7							
18	514	7							
19	382	7							
20	294	7							
21	249	7							
22	141	7							
23	152	7							
24	121	7							
25	77	7							
26	44	7							
27	26	7							
28	26	7							
29	26	7							
30	90	7							
31	70	7							
32	26	7							
33	46	7							
34	6	7							
35	90,324	90,324							

For next time:

■ Read Stevens Vol. I Chapter 4, 8
