The Pocket Guide to TCP/IP Sockets: C Version

Michael J. Donahoo
Kenneth L. Calvert
Computer Chat

- How do we make computers talk?
- How are they interconnected?

Internet Protocol (IP)
Internet Protocol (IP)

- Datagram (packet) protocol
- Best-effort service
  - Loss
  - Reordering
  - Duplication
  - Delay
- Host-to-host delivery
IP Address

- 32-bit identifier
- Dotted-quad: 192.118.56.25
- www.mkp.com -> 167.208.101.28
- Identifies a host interface (not a host)
Transport Protocols

Best-effort not sufficient!

- Add services on top of IP
- User Datagram Protocol (UDP)
  - Data checksum
  - Best-effort
- Transmission Control Protocol (TCP)
  - Data checksum
  - Reliable byte-stream delivery
  - Flow and congestion control
IP addresses identify hosts
- Host has many applications
- Ports (16-bit identifier)

<table>
<thead>
<tr>
<th>Application</th>
<th>WWW</th>
<th>E-mail</th>
<th>Telnet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Port</td>
<td>80</td>
<td>25</td>
<td>23</td>
</tr>
</tbody>
</table>

192.18.22.13
Socket

How does one speak TCP/IP?

- Sockets provides interface to TCP/IP
- Generic interface for many protocols
Sockets

- Identified by protocol and local/remote address/port
- Applications may refer to many sockets
- Sockets accessed by many applications
TCP/IP Sockets

- mySock = socket(family, type, protocol);
- TCP/IP-specific sockets

<table>
<thead>
<tr>
<th>Family</th>
<th>Type</th>
<th>Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP</td>
<td>PF_INET</td>
<td>SOCK_STREAM</td>
</tr>
<tr>
<td>UDP</td>
<td></td>
<td>SOCK_DGRAM</td>
</tr>
</tbody>
</table>

- Socket reference
  - File (socket) descriptor in UNIX
  - Socket handle in WinSock
Specifying Addresses

- **Generic**
  ```c
  struct sockaddr
  {
    unsigned short sa_family; /* Address family (e.g., AF_INET) */
    char sa_data[14];          /* Protocol-specific address information */
  };
  ```

- **IP Specific**
  ```c
  struct sockaddr_in
  {
    unsigned short sin_family; /* Internet protocol (AF_INET) */
    unsigned short sin_port;   /* Port (16-bits) */
    struct in_addr sin_addr;   /* Internet address (32-bits) */
    char sin_zero[8];          /* Not used */
  };
  ```

  ```c
  struct in_addr
  {
    unsigned long s_addr;     /* Internet address (32-bits) */
  };
  ```
Clients and Servers

- **Client:** Initiates the connection

Client: Bob

“Hi. I’m Bob.”

Server: Jane

“Hi, Bob. I’m Jane”

“Nice to meet you, Jane.”

- **Server:** Passively waits to respond
TCP Client/Server Interaction

Server starts by getting ready to receive client connections...

**Client**
1. Create a TCP socket
2. Establish connection
3. Communicate
4. Close the connection

**Server**
1. Create a TCP socket
2. Assign a port to socket
3. Set socket to listen
4. Repeatedly:
   a. Accept new connection
   b. Communicate
   c. Close the connection
TCP Client/Server Interaction

/* Create socket for incoming connections */
if ((servSock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0)
    DieWithError("socket() failed");
TCP Client/Server Interaction

echoServAddr.sin_family = AF_INET;                         /* Internet address family */
echoServAddr.sin_addr.s_addr = htonl(INADDR_ANY);/* Any incoming interface */
echoServAddr.sin_port = htons(echoServPort);           /* Local port */

if (bind(servSock, (struct sockaddr *) &echoServAddr, sizeof(echoServAddr)) < 0)
    DieWithError("bind() failed");

Client
1. Create a TCP socket
2. Establish connection
3. Communicate
4. Close the connection

Server
1. Create a TCP socket
2. Bind socket to a port
3. Set socket to listen
4. Repeatedly:
   a. Accept new connection
   b. Communicate
   c. Close the connection
TCP Client/Server Interaction

Client
1. Create a TCP socket
2. Establish connection
3. Communicate
4. Close the connection

Server
1. Create a TCP socket
2. Bind socket to a port
3. Set socket to listen
4. Repeatedly:
   a. Accept new connection
   b. Communicate
   c. Close the connection

/* Mark the socket so it will listen for incoming connections */
if (listen(servSock, MAXPENDING) < 0)
  DieWithError("listen() failed");
TCP Client/Server Interaction

for (; ;) /* Run forever */
{
    clntLen = sizeof(echoClntAddr);

    if ((clntSock=accept(servSock,(struct sockaddr *)&echoClntAddr,&clntLen)) < 0)
        DieWithError("accept() failed");

Client
1. Create a TCP socket
2. Establish connection
3. Communicate
4. Close the connection

Server
1. Create a TCP socket
2. Bind socket to a port
3. Set socket to listen
4. Repeatedly:
   a. Accept new connection
   b. Communicate
   c. Close the connection
TCP Client/Server Interaction

Server is now blocked waiting for connection from a client

Client
1. Create a TCP socket
2. Establish connection
3. Communicate
4. Close the connection

Server
1. Create a TCP socket
2. Bind socket to a port
3. Set socket to listen
4. Repeatedly:
   a. Accept new connection
   b. Communicate
   c. Close the connection
TCP Client/Server Interaction

Later, a client decides to talk to the server...

Client
1. Create a TCP socket
2. Establish connection
3. Communicate
4. Close the connection

Server
1. Create a TCP socket
2. Bind socket to a port
3. Set socket to listen
4. Repeatedly:
   a. Accept new connection
   b. Communicate
   c. Close the connection
TCP Client/Server Interaction

Client
1. Create a TCP socket
2. Establish connection
3. Communicate
4. Close the connection

Server
1. Create a TCP socket
2. Bind socket to a port
3. Set socket to listen
4. Repeatedly:
   a. Accept new connection
   b. Communicate
   c. Close the connection

/* Create a reliable, stream socket using TCP */
if ((sock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0)
    DieWithError("socket() failed");
TCP Client/Server Interaction

Client
1. Create a TCP socket
2. Establish connection
3. Communicate
4. Close the connection

Server
1. Create a TCP socket
2. Bind socket to a port
3. Set socket to listen
4. Repeatedly:
   a. Accept new connection
   b. Communicate
   c. Close the connection

```c
echoServAddr.sin_family = AF_INET;            /* Internet address family */
echoServAddr.sin_addr.s_addr = inet_addr(servIP); /* Server IP address */
echoServAddr.sin_port = htons(echoServPort); /* Server port */

if (connect(sock, (struct sockaddr *) &echoServAddr, sizeof(echoServAddr)) < 0)
    DieWithError("connect() failed");
```
TCP Client/Server Interaction

Client
1. Create a TCP socket
2. Establish connection
3. Communicate
4. Close the connection

Server
1. Create a TCP socket
2. Bind socket to a port
3. Set socket to listen
4. Repeatedly:
   a. Accept new connection
   b. Communicate
   c. Close the connection

```c
echoStringLen = strlen(echoString); /* Determine input length */

/* Send the string to the server */
if (send(sock, echoString, echoStringLen, 0) != echoStringLen)
    DieWithError("send() sent a different number of bytes than expected");
```
TCP Client/Server Interaction

/* Receive message from client */
if ((recvMsgSize = recv(clntSocket, echoBuffer, RCVBUFSIZE, 0)) < 0)
  DieWithError("recv() failed");

Client
1. Create a TCP socket
2. Establish connection
3. Communicate
4. Close the connection

Server
1. Create a TCP socket
2. Bind socket to a port
3. Set socket to listen
4. Repeatedly:
   a. Accept new connection
   b. Communicate
   c. Close the connection
TCP Client/Server Interaction

**Client**
1. Create a TCP socket
2. Establish connection
3. Communicate
4. Close the connection

**Server**
1. Create a TCP socket
2. Bind socket to a port
3. Set socket to listen
4. Repeatedly:
   a. Accept new connection
   b. Communicate
   c. Close the connection

`close(sock);`  
`close(clntSocket)`
TCP Tidbits

- Client knows server address and port
- No correlation between `send()` and `recv()`

### Client
- `send("Hello Bob")`

### Server
- `recv() -> "Hello "`
- `recv() -> "Bob"
- `send("Hi ")`
- `send("Jane")`
- `recv() -> "Hi Jane"`
Closing a Connection

- close() used to delimit communication
- Analogous to EOF

Client

send(string)
while (not received entire string)
  recv(buffer)
  send(buffer)
close(socket)

Server

recv(buffer)
while (client has not closed connection)
  send(buffer)
  recv(buffer)
close(client socket)