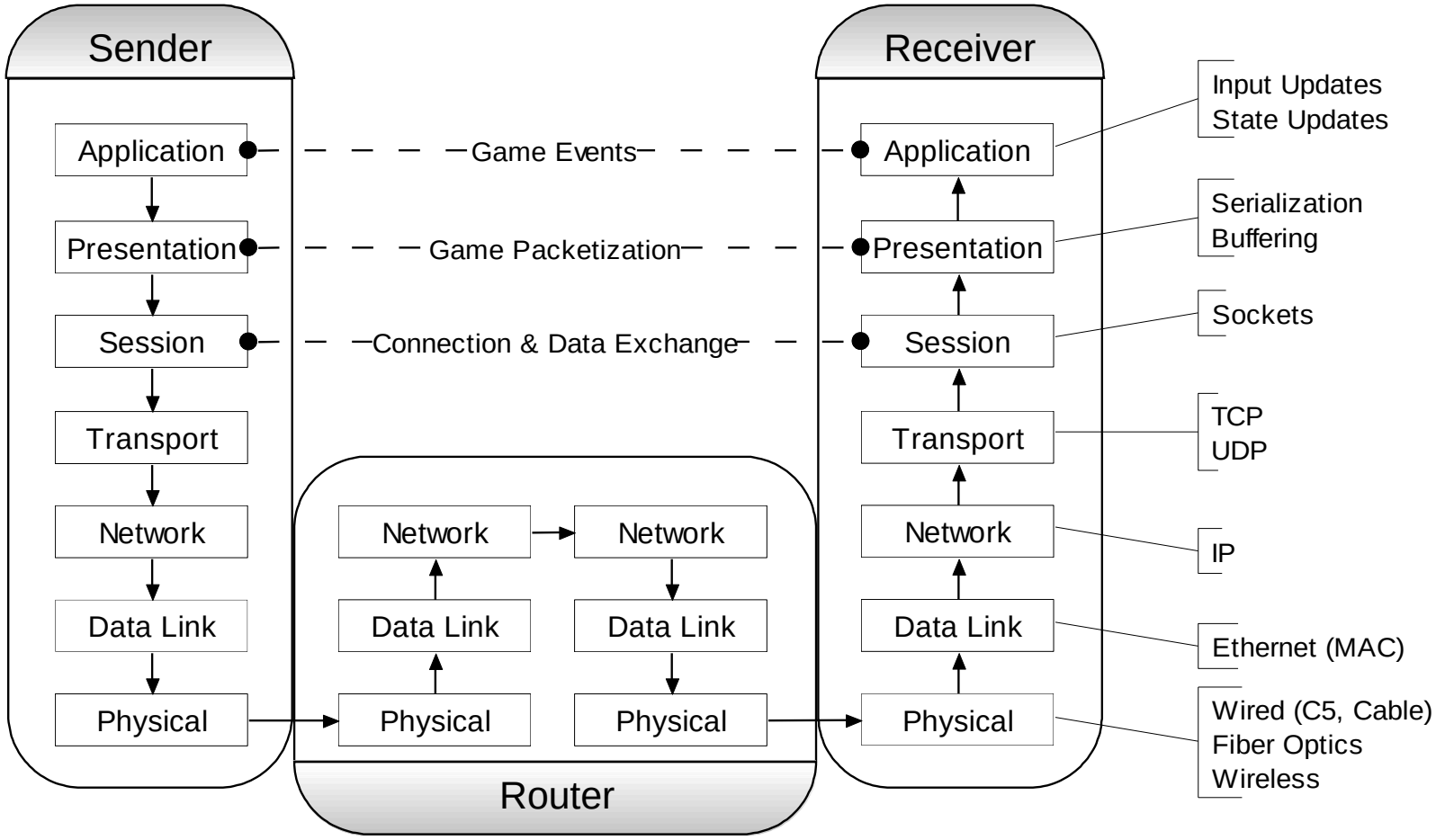
C++
Sockets

CO650 Advanced Programming

Winsock

- Sockets were initially developed for Unix
- Winsock is Windows implementation of sockets.
- Windows Sockets API (WAS)
- ws2_32.dll (C:\WINDOWS\System32)
- Winsock 2 extends Winsock 1.1 support for several protocols

OSI 7 Layers


Sender
Receiver
Application
Presentation
Session
Transport
Network
Data Link
Physical
Game Events
Game Packetization
Connection & Data Exchange
Network
Network
Data Link
Data Link
Physical
Physical
Router
Application
Presentation
Session
Transport
Network
Data Link
Physical
Input Updates
State Updates
Serialization
Buffering
Sockets
TCP
UDP
IP
Ethernet (MAC)
Wired (C5, Cable)
Fiber Optics
Wireless

OSI Layers

- Physical Layer : Bandwidth, Latency & medium

	Serial	USB 1&2	ISDN	DSL	Cable	LAN 10/100/1G BaseT	Wireless 802.11 a/b/g	Power Line	T1
Speed (bps)	20K	12M 480M	128k	1.5M down 896K up	3M down 256K up	10M 100M 1G	b=11M a,g=54M	14M	1.5 M

- Data link Layer : Network Interface Card, MAC address, Ethernet
- Network Layer : IP, Routing, Domain Name Service
- Transport Layer: TCP & UDP, Ports
- Session Layer : File I/O, WinSock, Sockets
- Presentation Layer : Compression, Encryption, Buffering
- Application Layer : Game logic

Architecture

- The connection is always between two devices, and each side uses its own IP and port number. Usually, one side is called the **client**, the other side the **server**.
- The server is continually waiting for incoming connections. This is called **listening** which is always done on a certain IP and port number.

IP Addresses

- Both the server and client use an IP and port number.
- The IP address of both server and client is configured during Network setup unless it is allocated dynamically.
- A machine may have more than one network interface card (NIC), in which case it may have more than one IP address.
- When developing Network Programs the port number of the server is usually specified within the code, whereas the client port number is allocated by the O/S.
- The Loopback address 127.0.0.1 refers to the current machine. This can be used during development to test both client and server on a single machine.

Ports

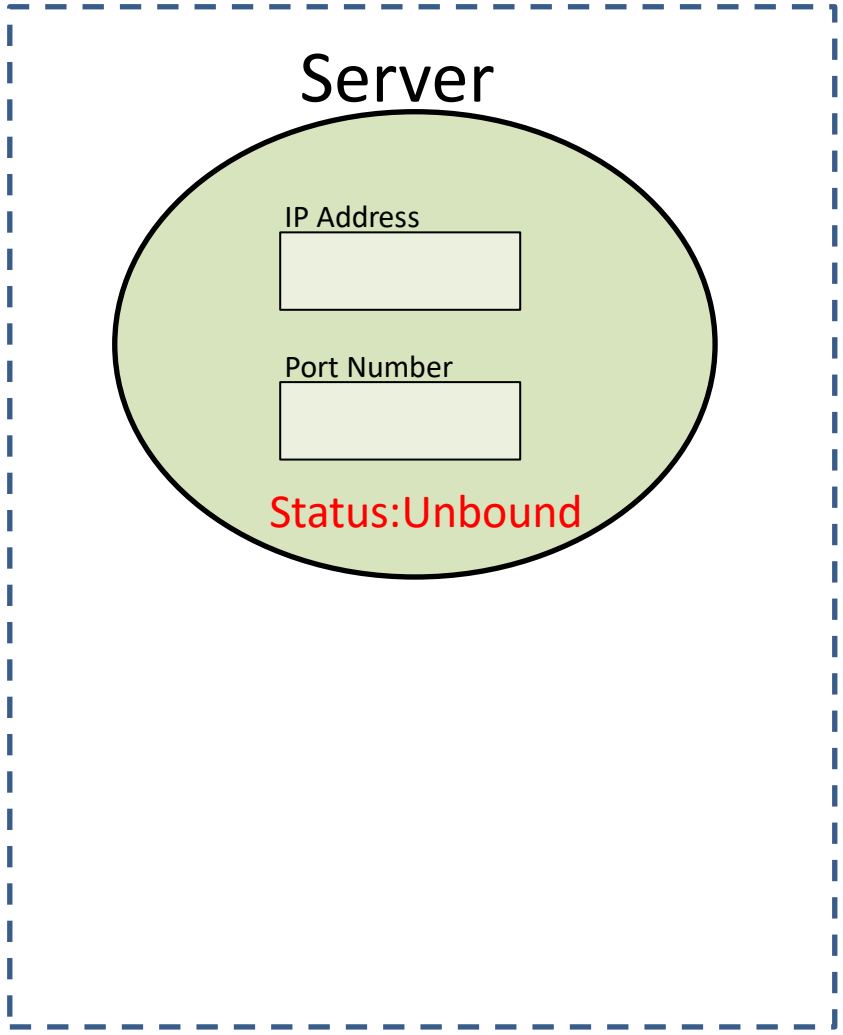
- Port numbers (16 bit address) can be any integer between 1 and 65.535.
- Ports 1..1023 are described as well know ports and are reserved for specific applications (port 21 FTP).
- It is recommended to choose a number over 1024. To be (almost) sure that your desired port isn't already in use.

Sockets

- Definition “A pipe between two computers on a network through which data flows” (Mulholland 2004).
- Almost all Winsock functions operate on a socket, as it's your handle to the connection. Both sides of the connection use a socket.
- Sockets are also two-way, data can be both sent and received on a socket.
- There are two common types for a socket
  - Streaming socket (SOCK_STREAM) TCP
  - Datagram socket (SOCK_DGRAM) UDP

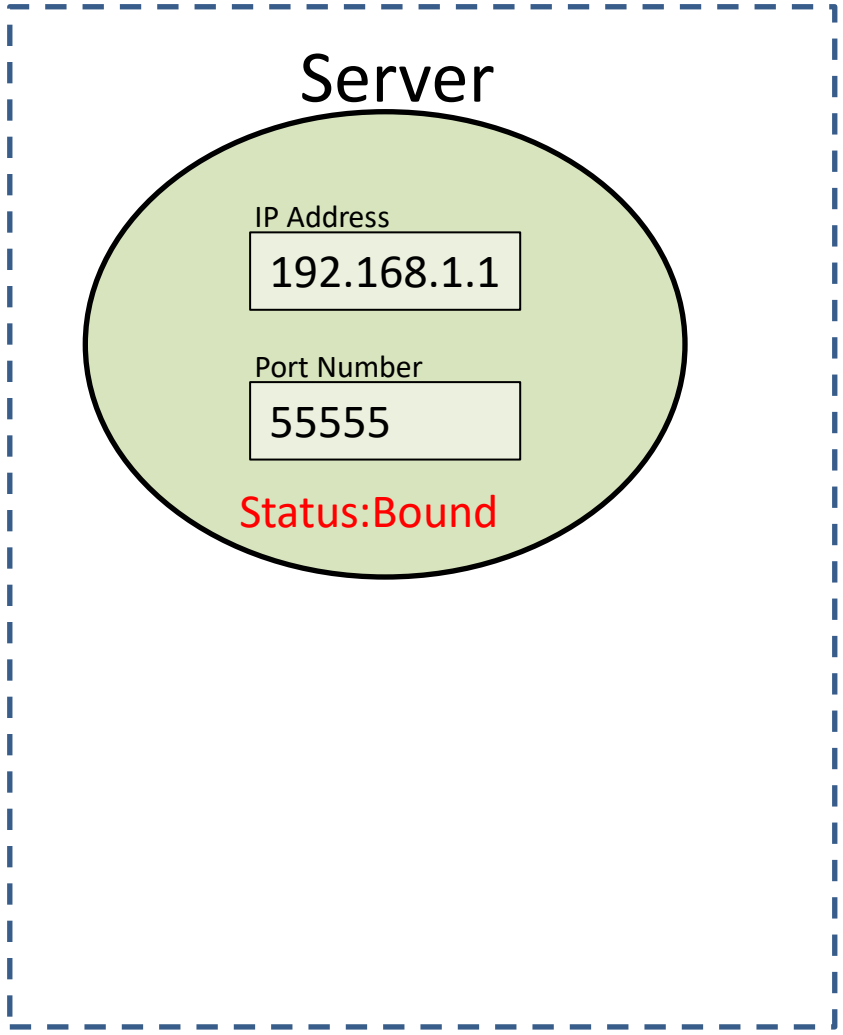
Create A Socket

The server creates a new socket. When created it is yet to be bound to an IP or port number



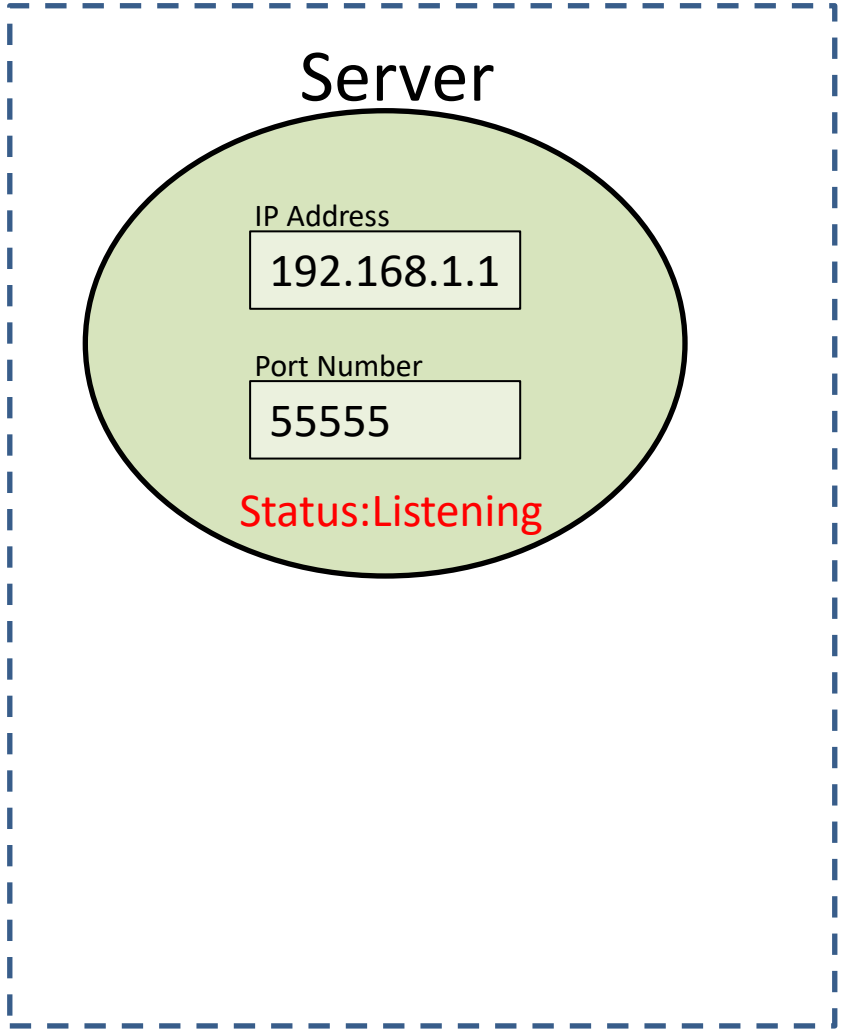
Binding A Socket

Bind the server to a valid IP address and port number.



Listening

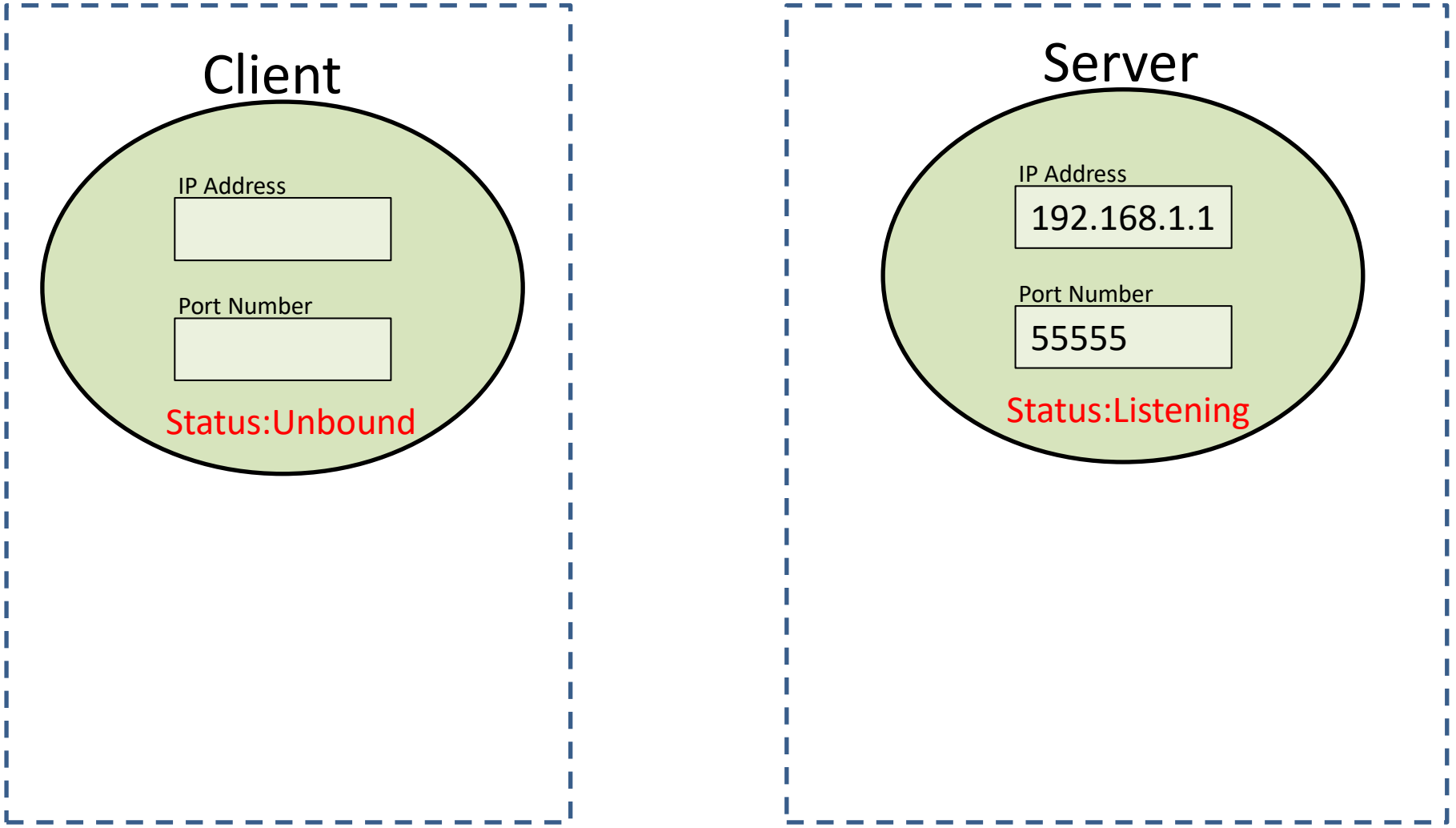
After the socket is bound, it is put into the listening state, waiting for incoming connections on the port (in this example port 5555).



Client Socket

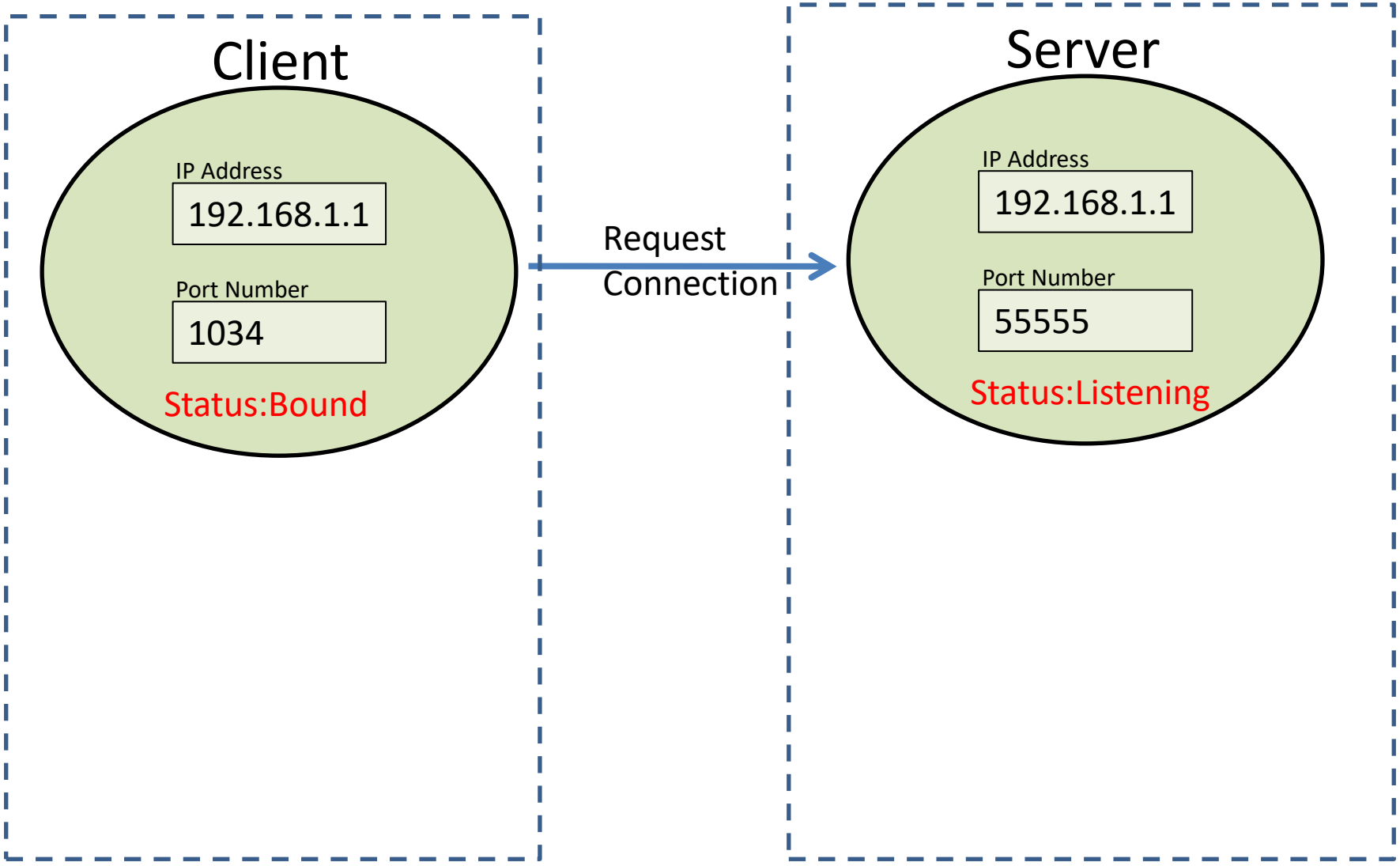
Assume a client in the same local network as the server (192.168.x.x)

it wants to connect to the server. It creates its own socket

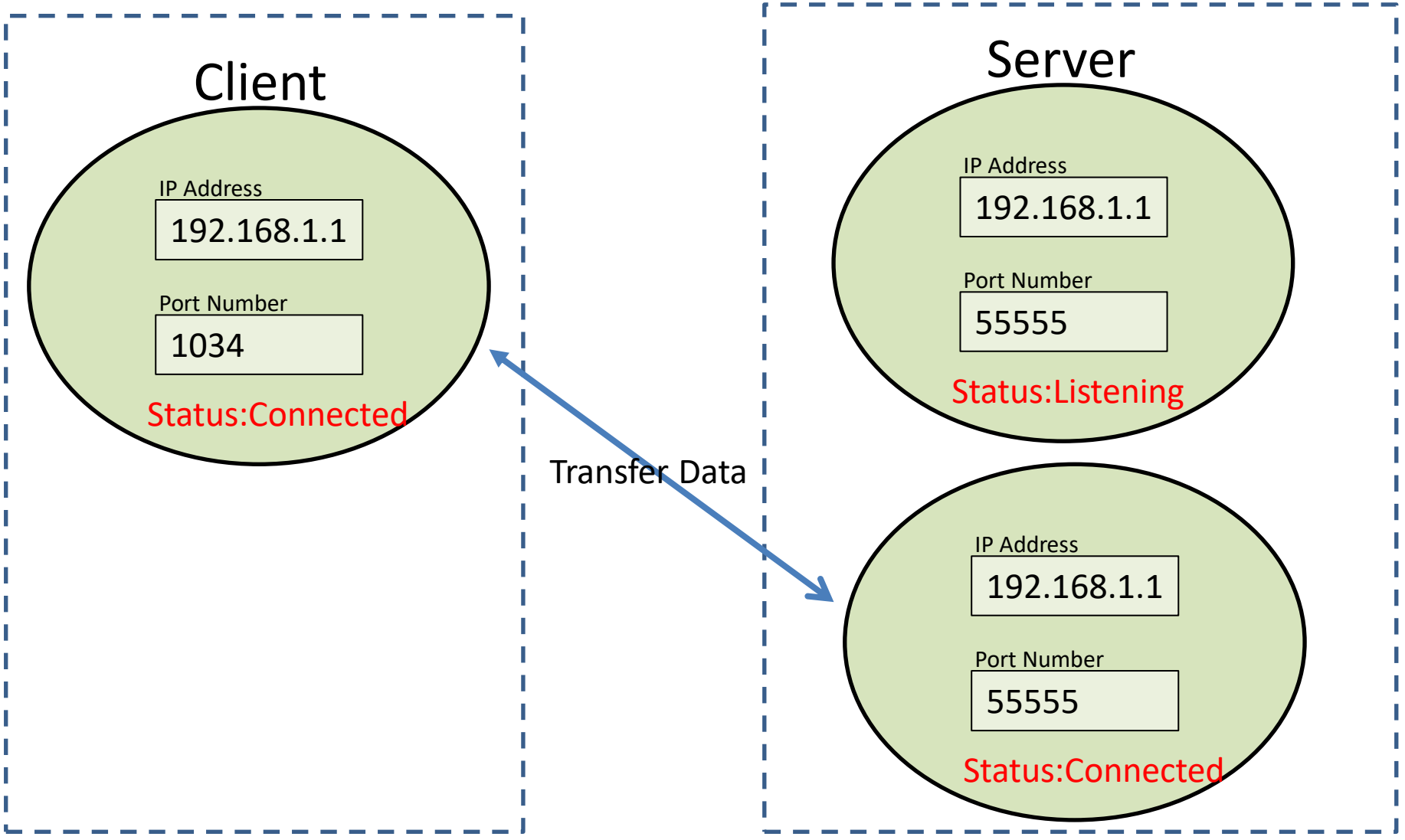


Request a Connection

The client socket tries to connect to the known IP address of the Server.



The listening socket sees some client wants to make a connection. It accepts it by creating a new socket (on the bottom right) bound to its own IP address and valid port. From this point, the client socket and the server connection socket just created will be able to communicate, while the listening socket will keep listening for other connections.



Server Functions

1. Initialize WSA – WSAStartup().
2. Create a socket – socket().
3. Bind the socket – bind().
4. Listen on the socket – listen().
5. Accept a connection – accept(), connect().
6. Send and receive data – recv(), send(), recvfrom(), sendto().
7. Disconnect – closesocket().

Client Functions

1. Initialize WSA – WSAStartup().
2. Create a socket – socket().
3. Connect to the server – connect().
4. Send and receive data – recv(), send(), recvfrom(), sendto().
5. Disconnect – closesocket().

The Server Code

- The Server must load the DLL by invoking WSAStartup
- It then creates a socket specifying the protocol to be used
- It binds the server's IP address to the socket
- Then listens for clients trying to establish connections
- On a client connecting the server creates a new socket to handle the client server communication

Initialising The DLL

1. Initiates use of the Winsock DLL by a process

```
int WSAStartup( WORD wVersionRequested,
                LPWSADATA lpWSAData );
```

2. *wVersionRequested :* The highest version of Windows Sockets specification that the caller can use. The high-order byte specifies the minor version number; the low-order byte specifies the major version number
3. A pointer to the LPWSADATA data structure that is to receive details of the Windows Sockets implementation
4. If successful, the WSAStartup function returns zero.

WSData Structure

The **WSADATA** structure contains information about the Windows Sockets implementation.

```
typedef struct WSAData {
WORD wVersion;
WORD wHighVersion;
char szDescription[WSADESCRIPTION_LEN+1];
char szSystemStatus[WSASYS_STATUS_LEN+1];
unsigned short iMaxSockets;
unsigned short iMaxUdpDg;
char FAR *lpVendorInfo; }
```

The **WSAStartup** function returns a pointer to the **WSADATA** structure in the *lpWSAData* parameter

Example

```
WSADATA wsaData;
int wsaerr;
WORD wVersionRequested = MAKEWORD(2, 2);
wsaerr = WSAStartup(wVersionRequested, &wsaData);
if (wsaerr != 0){
        cout << "The Winsock dll not found!" << endl;
        return 0;
}
else{
        cout << "The Winsock dll found!" << endl;
        cout << "The status: " << wsaData.szSystemStatus << endl;
}
```

Socket

- The **socket** function creates a socket that is bound to a specific transport service provider

```
SOCKET WSAAPI socket( int af,
                      int type,
                      int protocol );
```

- *af* : The address family specification (AF_INET for UDP or TCP).
- *type* : The type specification for the new socket (SOCK_STREAM for TCP and SOCK_DGRAM for UDP).
- *protocol* : The protocol to be used (IPPROTO_TCP for TCP).

Deregister Winsock2 DLL

- The **WSACleanup** function terminates use of the Winsock 2 DLL (*Ws2_32.dll*).
- The return value is zero if the operation was successful. Otherwise, the value SOCKET_ERROR is returned

```
int WSACleanup(void);
```

- When it has completed the use of Windows Sockets, the application or DLL must call **WSACleanup** to deregister itself from a Windows Sockets.
- Multiple applications may share a DLL. Windows tracks the number of applications using each DLL and will only remove the DLL from system memory when it is no longer required.

Example

```
        SOCKET serverSocket = INVALID_SOCKET;
        serverSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
        if (serverSocket== INVALID_SOCKET){
                cout << "Error at socket(): " <<  WSAGetLastError() << endl;
                WSACleanup();
                return 0;
        }
        else {
                cout << "socket() is OK!" << endl;
        }
//.....
WSACleanup();
```

Close Socket

- Closes the socket passed as an argument
- The socket must have previously been opened through a call to socket

```
int closesocket(Socket s);
```

```
SOCKET serverSocket;
serverSocket = INVALID_SOCKET;
serverSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

// Process socket
closesocket(serverSocket);
```

bind

- Associates a local address with a socket.

```
int bind(SOCKET s,
        const struct sockaddr* name,
        int socklen);
```

- s : Descriptor identifying an unbound socket.
- name : Address to assign to the socket from the sockaddr structure.
- socklen : Length in bytes of the address structure.
- If no error occurs, bind() returns zero. Otherwise, it returns SOCKET_ERROR

bind

- The **SOCKADDR_IN** structure is used by Windows Sockets (IP4) to specify a local or remote endpoint address to which to connect a socket

```
struct sockaddr_in{ short sin_family;
                    unsigned short sin_port;
                    struct in_addr sin_addr;
                    char sin_zero[8];
};
```

- *sin_family* Address family (must be **AF_INET**).
- *sin_port* IP port.
- *sin_addr* IP address.
- *sin_zero* Padding to make structure the same size as **SOCKADDR**. The **htons** function returns the value in TCP/IP network byte order.

Bind Example

```
sockaddr_in service;
service.sin_family = AF_INET;
InetPton(AF_INET, _T("127.0.0.1"), &service.sin_addr.s_addr);
service.sin_port = htons(port);
if (bind(serverSocket, (SOCKADDR*)&service, sizeof(service)) == SOCKET_ERROR){
		cout << "bind() failed: " <<  WSAGetLastError() << endl;
		closesocket(serverSocket);
		WSACleanup();
		return 0;
}
else{
		cout << "bind() is OK!" << endl;
}
```

Listen

- Places a socket in a state in which it is listening for an incoming connection.

```
int listen(SOCKET s, int backlog);
```

- s : Descriptor identifying a bound, unconnected socket.
- backlog : The maximum number of connections allowed (also OS dependant).
- If no error occurs, **listen** returns zero. Otherwise, a value of SOCKET_ERROR is returned

Listen Example

```
if ( listen(serverSocket, 1) == SOCKET_ERROR)
        cout << "listen(): Error listening on socket " <<  WSAGetLastError()<< endl;
else
        cout << "listen() is OK, I'm waiting for connections..." << endl;
```

accept

- Permits an incoming connection on a socket.
- This is a blocking function.

```
SOCKET accept(SOCKET s,
              struct sockaddr* addr,
              int* addrlen);
```

- s : Descriptor that identifies a socket that has been placed in a listening state with the listen() function.
- addr : Optional structure containing the client address information
- Addrlen : Optional size of the address structure (if included).
- If no error occurs, accept() returns a value of type SOCKET that is a descriptor for the new socket that is connected to the client. The original listening socket can be used to listen for more incoming calls.

Accept Example

```
SOCKET acceptSocket;
acceptSocket = accept(serverSocket, NULL,NULL);
if (acceptSocket == INVALID_SOCKET){
        cout <<  "accept failed: " << WSAGetLastError() << endl;
                WSACleanup();
                return -1;
}
```

The Client Code

- The Client must also load the DLL by invoking WSAStartup
- It then creates a socket
- It then connects to the server by assigning the servers IP address and port to a sockaddr structure and passing this along with the client socket to the connect function

Connect

- Connects a client to a server (invoked from within the client).
- Within the client binding is performed automatically.

```
int connect(SOCKET s,
                const struct sockaddr* addr,
                 socklen_t  addrlen);
```

- s : Descriptor that identifies a socket.
- addr : Structure containing server IP address and port.
- addrlen : Size in bytes of addr structure
- Connect will wait 75 seconds for server to respond.
- Returns 0 if successful or SOCKET_ERROR if not.

Connect Example

```
sockaddr_in clientService;
clientService.sin_family = AF_INET;
InetPton(AF_INET, _T("127.0.0.1"), &clientService.sin_addr.s_addr);
clientService.sin_port = htons(port);
if ( connect(clientSocket, (SOCKADDR*)&clientService, sizeof(clientService)) == SOCKET_ERROR){
        cout << "Client: connect() - Failed to connect." << endl;
        WSACleanup();
        return 0;
}
else   {
       cout << "Client: connect() is OK." << endl;
       cout << "Client: Can start sending and receiving data..." << endl;
}
```

Summary

- There are 5 steps in required to setup as TCP server.
- The purpose of much of the code in the previous slides is for checking errors.
- The code required to setup a UDP server will be covered in another presentation.