

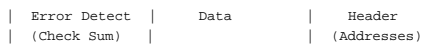
The Internet: Networking with Stream-based Sockets

The Internet

- A Global Network of Networks
- ARPANet : SRI, Utah, UCLA, UCSB, (1969)
 - Defense Dept. Advanced Research Projects Agency (DARPA)
 - Stanford Research Institute (Doug Engelbart)
 - Designed to survive bomb attacks
 - Distributed control, Expandable
- Ethernet
 - Global standard for interconnecting computers
 - Xerox PARC (Early 70s)
 - Client/Server architecture
- Exponential Growth
 - Tens of Millions of Computers
 - Hundreds of millions of Users

The Internet

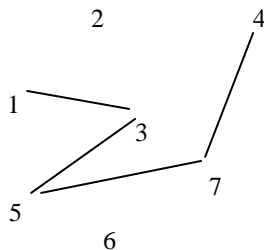
- A Packet Switched Network
 - Like Postal System
 - Messages broken up into packets (like envelopes)



Computer Node Addresses:

- IP (Internet Protocol)
 - 32 bit numeric address in four 8-bit fields:
 - 128.226.6.4 (bingsons IP Address)
- | | |
|--------------|-----------------|
| | |
| network | computer |
| (city/state) | (street/number) |
- TCP (Transmission Control Protocol):
 - Send Site: Breaks message into packets
 - Receive Site: Collects & Reassembles packets in proper order

“Best” routing is chosen using Routers



Domain Names

- Synonyms for IP Addresses
- bingsuns.binghamton.edu
 - |
 - individual machine
 - |
 - largest domain
 - Synonym for 128.226.6.4
- Internet Domain Name Server (DNS) software maps domain names to IP addresses

Common High-Level Domain Names

- com: commercial
- edu: educational
- gov: government
- mil: military
- org: other organization
- net: network resources
- --: country name
 - e.g., ca = Canada

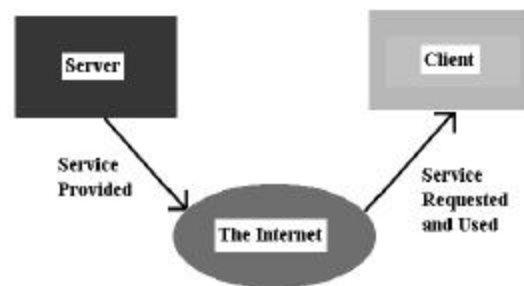
The .NET Dns Class

- In System.Net namespace
- **Dns**: a static class that retrieves information about a specific host from the DNS
 - Info returned in an instance of the IPHostEntry class
 - If the specified host has more than one entry in the DNS database, IPHostEntry contains multiple IP addresses and aliases
 - Dns.GetHostByName(string hostName) method
 - Returns an IPHostEntry object containing host information for the address specified in *hostName*
 - That object's AddressList property can be used to set up an array of IPAddresses that corresponds to the hostnames
- See GetIPAddress example program

Networking Software

- Client/Server Model
 - Client Program -- seeks a service from remote computer
 - Server Program -- provides a service to a client running on a remote computer
 - Computers usually connected over a network
 - Examples
 - Print Server
 - File Server
 - Information Server

Client/Server Model



Information Servers

- Program handles requests for information
- Some examples
 - e-mail: electronic mail service
 - telnet: remote logon service
 - ftp: file transfer service
 - gopher: net browsing service (text based)
 - archie/veronica: automated net search services
 - WAIS: automated file content search service
 - Net News: network bulletin board service
 - WWW: hypermedia access to internet (Web page service)

Telnet Client

- A remote logon client
- You need an account on remote machine
- Starting Windows Telnet client (from command prompt):

```
telnet
```

 - You'll be prompted for a domain name or IP address
 - Then can look at commands by entering 'help'
- Another way of starting telnet:

```
telnet domain-name or IP-address
```

 - You provide logon ID & password
 - Starts a session with the remote machine
- Also available in BU's BUICK Suite

FTP--File Transfer Protocol

- Many "anonymous" ftp servers
 - provide access to public or password-protected files
 - Usually used to transfer files between computers
- Starting Windows ftp client (from DOS command prompt):
ftp domain-name or IP-address
- Response:
ftp> User:
ftp> Password:
- Getting help:
ftp> help
- WS_FTP GUI version available in BU's BUICK Suite

Network Communication Between Computers

- Applications running on different computers can communicate with each other
 - Server App: Waits for other apps on other computers to open a communication connection
 - Client App: Attempts to open a connection
- When connection is established, data can be exchanged
- Either can close the communication
- Connections:
 - Two programs running on different computers that are communicating with each other form a connection
 - Data is sent and received along the connection

Socket Stream

- Basic object used to perform network communication
- Used to read/write messages going between applications
 - (Like a file stream in file I/O)
- A Socket is a communication "endpoint"
 - There's a socket at each end of the connection
- Windows support for sockets: in the Winsock API
 - MFC encapsulates this in the CAsyncSocket base class
 - Provides complete, event-driven socket communications
 - Lowest level support
 - Notes at: www.cs.binghamton.edu/~reckert/360/17b_sockets_f03.html
 - Higher level support from derived classes like CSocket
- .NET encapsulates socket support in:
 - System.Net.Sockets namespace
 - With sockets, networking is viewed like file I/O
 - Read from or write to a socket connection as easily as to a file

Making a Socket Connection to a Process Running on Another Computer

- Specify the IP Address of computer where other application is running
 - Identifies a machine
- Also specify the Port the application is listening on
 - Identifies the program that should handle the communication
 - e.g. port 80 is reserved for web document transfer
 - Like telephone communication: (Dial number and extension)
 - Can be any number from 0 to 65535
 - But numbers 0 to 1023 may be used by the operating system
 - So use numbers greater than 1023

Details of Establishing a Simple Server (Using Socket Streams)

1. Create a TcpListener class object
 - TcpListener myListener = new TcpListener(5000);
 - Parameter is port number to bind the server to on the machine it's running on
2. Call TcpListener object's Start() method to start listening for connection requests
 - myListener.Start();
3. Use TcpListener's AcceptSocket() to accept an incoming request and establish the connection
 - Socket myConnection = myListener.AcceptSocket();
 - Returns a Socket object
 - » Socket object will be null if connection was not made
 - » Its Connected property will be true after socket is connected
4. Create a NetworkStream associated with the socket
 - NetworkStream myNetStream = new NetworkStream(myConnection);
 - This will be used to do the reading and writing as in File I/O

Using the Server Socket Stream Connection

5. Create BinaryReader and BinaryWriter objects for transferring data across the stream

```
BinaryWriter myWriter = new BinaryWriter(myNetStream);
BinaryReader myReader = new BinaryReader(myNetStream);
```
6. Use BinaryReader/BinaryWriter objects to read/write data

```
string receiveStr, sendStr;
receiveStr = myReader.ReadString();
– Reads a line of text from the stream (sent by the client)
myWriter.Write(sendStr);
– Writes the string to the stream (to the client)
```
7. When done, close readers, writers, network stream, and connection socket

```
myReader.Close(); myWriter.Close();
myNetStream.Close(); myConnection.Close();
```

Details of Establishing a Simple Client (Using Socket Streams)

1. Create a `TcpClient` class object
`TcpClient myClient = new TcpClient();`
2. Try to connect to a server
 - Call object's `Connect(IP address, port)` method
 - Specify IP address (or domain name) of machine server is running on and server's port number in the two parameters
 - If successful, a positive integer is returned
 - `myClient.Connect("localhost", 5000);`
 - "localhost" = "loopback" = 127.0.0.1 means same machine as server
 - If successful, and underlying socket will be created for communication
 - Will throw an exception if no Server available
3. Get a `NetworkStream` associated with the `TcpClient`
`NetworkStream myNetStream = myClient.GetStream();`
 - This will be used to do the reading and writing as in File I/O

Using the Client Socket Stream Connection

4. Create `BinaryReader` and `BinaryWriter` objects for transferring data across the stream
`BinaryWriter myWriter = new BinaryWriter(myNetStream);`
`BinaryReader myReader = new BinaryReader(myNetStream);`
5. Use `BinaryReader/BinaryWriter` objects to read/write data
`string receiveStr, sendStr;`
`receiveStr = myReader.ReadString();`
 - Reads a line of text from the stream (sent by the server)`myWriter.Write(sendStr);`
 - Writes the string to the stream (to the server)
6. When done, close readers, writers, network stream, and TCP client
`myReader.Close(); myWriter.Close();`
`myNetStream.Close(); myClient.Close();`

Using Threads with Sockets

- Whenever we try to establish and use a connection, the thread we do it in blocks until the connection is established
 - Blocking also takes place when reading or writing data
- To avoid the entire application from freezing, run this code in a separate thread

A Network Chat Client/Server System

- A Server and a Client Application
 - See Chapter 23 in your Deitel text book
 - Also the `ChatServer` and `ChatClient` example program code on the CS-360 Sample Programs web pages
- `ChatServer` application waits for a client application to connect to a specified port on its computer
- `ChatClient` application attempts to connect to that port on that machine
- Both `ChatServer` and `ChatClient` have a single-line "input" text box and a multi-line "display" text box
- When a connection is established, either can type text in its input text box and the text will appear in the other's display text box
- The communication is done through socket streams

ChatServer Application

- Form's constructor starts a new thread to accept client connections
 - Thread's `RunServer()` method does the work (executes when thread starts)
 - Creates and starts a `TcpListener` on port 5000
 - Listens for a connection attempt from a client
 - Connection is made (socket obtained) with listener's `AcceptSocket()` method
 - Uses socket's `NetworkStream()` method to get a socket stream
 - Creates binary reader and writer to read/write data over the socket stream connection
 - Enters into a do/while loop that continually uses the binary reader to read a string from the socket stream
 - Any string read is added to the text displayed in the "display" text box
 - Do/While loop continues until the socket is disconnected or a ">>CLIENT TERMINATE" string is received
 - After do/while loop exits, the reader, writer, network stream, and socket are closed
- Input text box's `KeyDown` handler
 - Writes the text in the input text box to the socket stream using its binary writer whenever the user types <Enter> as long as the connection is valid
 - If the text entered is "TERMINATE", closes the connection socket
- An event handler for the form's "Closing" event is added
 - Calls `System.Environment.Exit(System.Environment.ExitCode)` to close the app
 - `Exit()` method of `Environment` class closes all threads associated with the app

ChatClient Application

- Same overall structure as the `ChatServer`
- Form's constructor starts a new thread to connect to the server
 - Thread's `RunServer()` method does the work
 - Instantiates a `TcpClient` and run its `Connect("localhost", 5000)` method
 - Connects to the server on the same machine
 - This call blocks until connection request is accepted
 - Uses `TcpClient`'s `GetStream()` method to get a socket stream
 - Creates a binary reader and a binary writer to read/write data over the socket stream connection
 - Enters into a do/while loop that continually uses the binary reader to read a string from the socket stream and display it in the form's "Display" text box
 - After do/while loop exits, the reader, writer, `NetworkStream`, and `TcpClient` are all closed and app is closed using the `Application.Exit()` method
- Input text box's `KeyDown` handler
 - Write the text in the input box to the socket stream using its binary writer as in the `ChatServer` app
- For both the server and the client, it would be much better to use `Try/Catch` blocks, as in the examples on the website