Lecture 4

Internet Applications and Network Programming

Topics Covered

- Connectionless communication
- Connection-Oriented Communication
- The Client-Server Model of Interaction
- Server Programs and Server-Class Computers
- Multiple Clients and Multiple Servers
- Server Identification and De-multiplexing
- Concurrent Servers
- Peer-to-Peer Interactions
- Network Programming and the Socket API
- Socket Programming in Java

- Chapter 3 and online materials

Introduction

- Internet is the **communication infrastructure**
  - *The Internet does not provide services. Instead,*
  - *The Internet only provides communication,*
  - *and **application programs provide all services**.*
- Internet provides a general purpose mechanism on which
  - all services are built
  - and individual services are supplied by application programs that run on computers attached to the Internet
- It is possible to devise new services without changing the Internet

Two Basic Internet Communication Paradigms

- The Internet supports two basic communication paradigms:
  - Stream Transport on the Internet (TCP)
  - Message Transport on the Internet (UDP)
- Figure 3.1 summarizes the differences

Stream Paradigm	Message Paradigm
Connection-oriented	Connectionless
1-to-1 communication	Many-to-many communication
Sequence of individual bytes	Sequence of individual messages
Arbitrary length transfer	Each message limited to 64 Kbytes
Used by most applications	Used for multimedia applications
Built on TCP protocol	Built on UDP protocol

Figure 3.1 The two paradigms that Internet applications use.

Stream Transport on the Internet

- Stream denotes a paradigm in which a sequence of bytes flows from one application program to another
- **Duplex** communications: both can send/receive at the same time
- The stream mechanism transfers a sequence of bytes without attaching meaning to the bytes and without inserting boundaries
- The data communications are done in the same way as file input/output (I/O)
- Stream transport provides 1-to-1 communication
- Based on TCP service ➔ **reliable**

Connection-Oriented Communication

- The Internet stream service is connection-oriented
- It operates analogous to a telephone call:
  1. two applications must request that a connection be created
  2. once it has been established, the connection allows the applications to send data in either direction
  3. finally, when they finish communicating, the applications request that the connection be terminated

Algorithm 3.1

Purpose:

Interaction over a connection-oriented mechanism

Method:

A pair of applications requests a connection

The pair uses the connection to exchange data

The pair requests that the connection be terminated

Algorithm 3.1 Communication over a connection-oriented mechanism.

Message Transport on the Internet

- In a message paradigm, the network accepts and delivers messages
- Each message delivered to a receiver corresponds to a message that was transmitted by a sender
  - the network never delivers part of a message, nor does it join multiple messages together
  - if a sender places exactly *n* bytes in an outgoing message, the receiver will find exactly *n* bytes in the incoming message
- The message paradigm allows delivery in different forms:
  - Unicast
    - a message can be sent from an application on one computer directly to an application on another, 1-to-1
  - Multicast
    - a message can be multicast to some of the computers on a network, 1-to-many
  - Broadcast
    - a message can be broadcast to all computers on a given network, 1-to-all

Message Transport in the Internet

- Message service does not make any guarantees, so messages may be
  - Lost (i.e., never delivered)
  - Duplicated (more than one copy arrives)
  - Delivered out-of-order
- It is connectionless communication
- A programmer who uses the message paradigm must insure that the application operates correctly
  - even if packets are lost or reordered
- Programmers tend to use the stream service except in special situations
  - such as video, where multicast is needed, and the application provides support to handle packet reordering and loss
- Based on UDP service ➔ not reliable

The Client-Server Model of Interaction

- Most Internet applications are based on client-server computing model
- A traditional program is divided into two programs
  - A client program (or just client)
  - A server program (or just server)
  - they can run on a single computer or networked computers

- Figure 3.2 summarizes the interaction
- Application programs known as clients and servers handle all services on the Internet

The Client-Server Model of Interaction

Server Application	Client Application
Starts first	Starts second
Does not need to know which client will contact it	Must know which server to contact
Waits passively and arbitrarily long for contact from a client	Initiates a contact whenever communication is needed
Communicates with a client by both sending and receiving data	Communicates with a server by sending and receiving data
Stays running after servicing one client, and waits for another	May terminate after interacting with a server

Figure 3.2 A summary of the client-server model.

Server Programs and Server-Class Computers

- Term server refers to a program that waits passively for communication
  - Not to the computer on which it executes
- However, when a computer is dedicated to running one or more server programs,
  - the computer itself is sometimes called a server
- Hardware vendors contribute to the confusion
  - because they classify computers that have fast CPUs, large memories, and powerful operating systems as server machines
- Figure 3.3 illustrates the definitions

Server Programs and Server-Class Computers

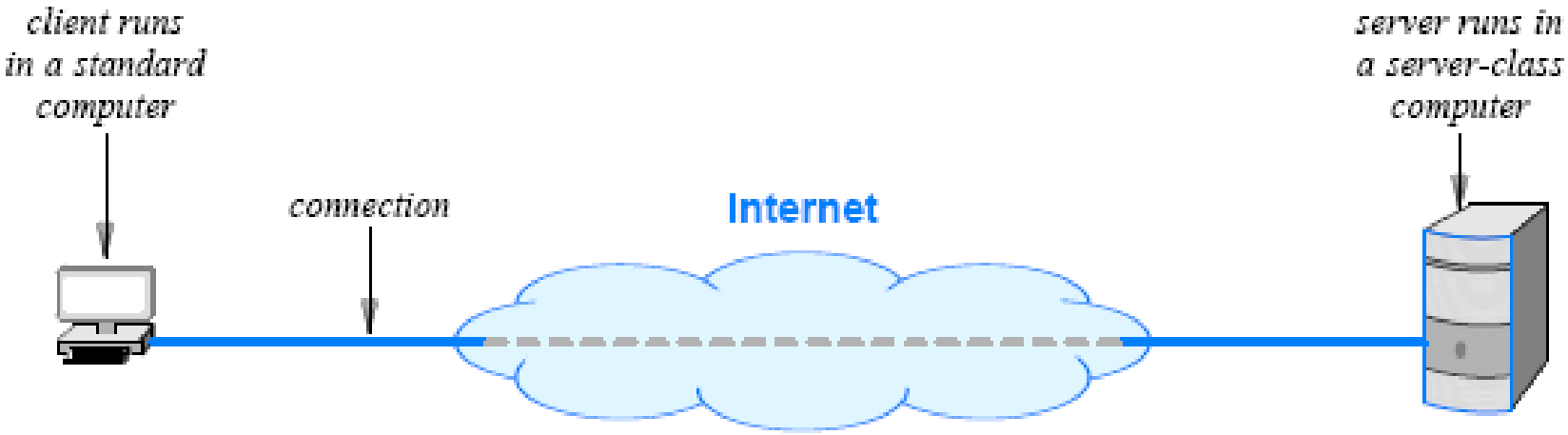


Figure 3.3 Illustration of a client and server.

Multiple Clients and Multiple Servers

- Allowing a given computer to operate multiple servers is useful because
  - the hardware can be shared
  - a single computer has lower system administration overhead than multiple computer systems
  - experience has shown that the demand for a server is often sporadic
    - a server can remain idle for long periods of time
    - an idle server does not use the CPU while waiting for a request to arrive
- If demand for services is low, consolidating servers on a single computer can dramatically reduce cost
  - without significantly reducing performance

Multiple Clients and Multiple Servers

- A computer can run:
  - A single client
  - A single server
  - Multiple instances of a client that contact a given server
  - Multiple clients that each contact a particular server
  - Multiple servers, each for a particular service
- Allowing a computer to operate multiple clients is useful
  - because services can be accessed simultaneously
- For example, a user can have three (3) windows open simultaneously running three (3) applications:
  - one that retrieves and displays email
  - another that connects to a chat service
  - and a third running a web browser

Server Identification and Demultiplexing

- How does a client identify a server?
- The Internet protocols divide identification into two pieces:
  - An identifier for the computer on which a server runs
  - An identifier for a service on the computer
- Identifying a computer?
  - Each computer in the Internet is assigned a unique 32-bit identifier known as an Internet Protocol address (IP address)
  - A client must specify the server's IP address
  - To make server identification easy for humans, each computer is also assigned a name, and the Domain Name System (DNS)
  - Thus, a user specifies a name such as *www.cisco.com* rather than an integer address

Server Identification and Demultiplexing

- Identifying a service?
  - Each service available in the Internet is assigned a unique 16-bit identifier known as a protocol port number (or port number)
    - Examples, email → port number 25, and the web → port number 80
  - When a server begins execution
    - it registers with its local OS by specifying the port number for its service
  - When a client contacts a remote server to request service
    - the request contains a port number
  - When a request arrives at a server
    - software on the server uses the port number in the request to determine which application on the server computer should handle the request
- Figure 3.4 summarizes the basic steps

Server Identification and Demultiplexing

- Start after server is already running
- Obtain server name from user
- Use DNS to translate name to IP address
- Specify that the service uses port N
- Contact server and interact

- Start before any of the clients
- Register port N with the local system
- Wait for contact from a client
- Interact with client until client finishes
- Wait for contact from the next client…

Figure 3.4 The conceptual steps a client and server take to communicate.

Concurrent Servers

- The steps in Figure 3.4 imply that a server handles one client at a time
- Although a serial approach works in a few trivial cases, most servers are concurrent
- Concurrent implementation depends on the OS being used
  - Process
  - Thread
- In Operating Systems part, you will learn concurrent programming and execution

Peer-to-Peer Interactions

- If a single server provides a given service
  - the network connection between the server and the Internet can become a bottleneck
- Figure 3.5 illustrates the architecture

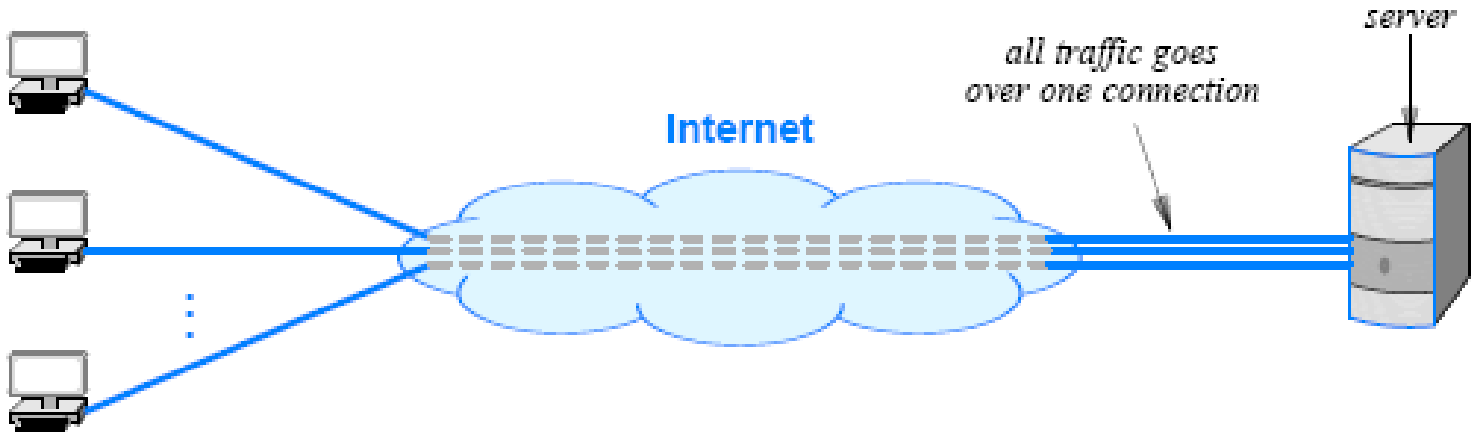


Figure 3.5 The traffic bottleneck in a design that uses a single server.

Peer-to-Peer Interactions

- Can Internet services be provided without creating a central bottleneck?
  - One way to avoid a bottleneck forms the basis of file sharing known as a peer-to-peer (P2P) architecture
- The scheme avoids placing data on a central server
  - data is distributed equally among a set of N servers
  - and each client request is sent to the appropriate server
  - a given server only provides 1/N of the data
    - the amount of traffic between a server and the Internet is 1/N as much as in the single-server architecture
- Server software can run on the same computers as clients
- Figure 3.6 illustrates the architecture

Peer-to-Peer Interactions

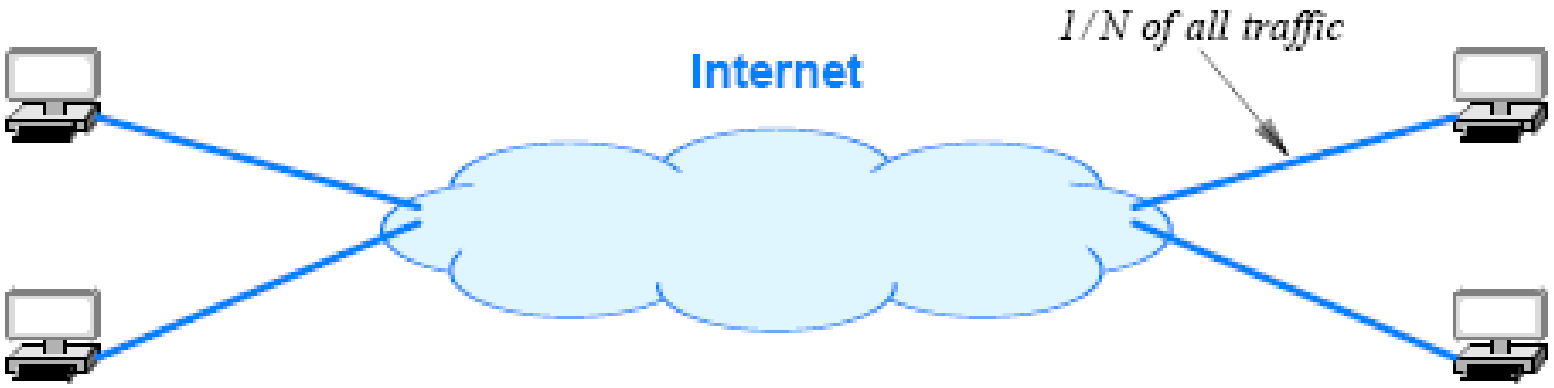
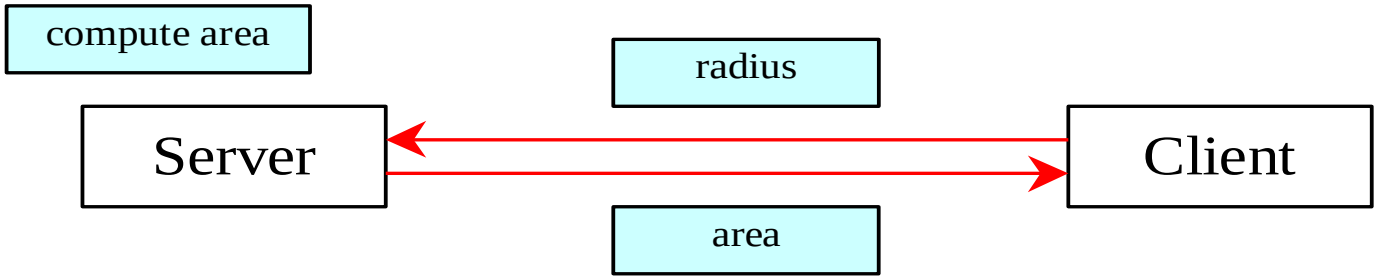


Figure 3.6 Interaction in a peer-to-peer system.

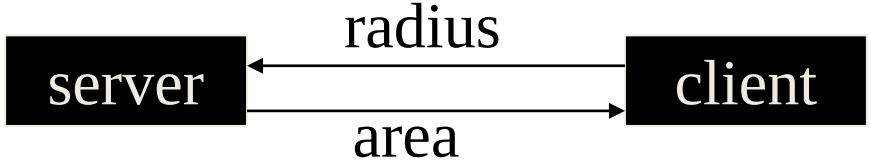
A Client/Server Application Example

- *Problem*: implement an Internet service that computes the area of a circle.
- Client: gets a radius from user, sends it to server, gets the area, displays the area for the user
- Server: gets the radius from a client, computes the area, sends the area to the client



Example

- Implement a server application that computes the area of the circle.
- Protocol (syntax and semantics)
  - double number is used for radius and area (8 bytes)
  - Client → server: a double number (8 bytes) representing a radius
  - Client ← server: a double number (8 bytes) representing the area
- Client application
  - Read a radius from console
  - Send it to the server
  - Get the area from the server and display it
- Server application
  - Read radius from client
  - Compute the area
  - Send it back to the client



Network Programming and the Socket API

- The interface an application uses to specify communication is known as an Application Program Interface (API)
- Details of an API depend on the OS and language
- socket API is the de facto standard for software that communicates over the Internet
- The socket API is available for many OS
  - such as Microsoft's Windows systems
  - as well as various UNIX systems, including Linux

What is a socket?

- Socket
  - The combination of an IP address and a port number. (RFC 793 ,original TCP specification)
  - The name of the Berkeley-derived *application programming interfaces* (APIs) for applications using TCP/IP protocols.
  - Two types
    - Stream socket -- using TCP in layer 4
    - Datagram socket – using UDP in layer 4

- Socket pair
  - Specified the two end points that uniquely identifies each TCP connection in an Internet.
  - 4-tuple: (client IP address, client port number, server IP address, server port number)

Internet Programming in Java

- Socket based communication
  - In Package **java.net**
  - java.net.Socket for client
  - java.net.ServerSocket for server
- Server implementation
  - Create a server socket and attach it to a port

    ServerSocket serverSocket= new **ServerSocket(** port **);**
  - Server is in passive mode to wait for the client service request

    Socket *socket* = ***serverSocket.accept( );***
  - Data exchange using input/output streams
    - Same way as file input/output
    - ***socket.getInputStream( )***
    - ***socket.getOutputStream( )***
  - *Close the streams and socket*

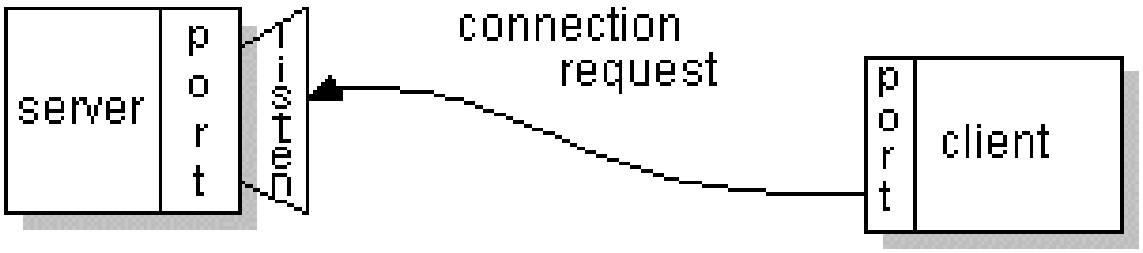
Internet Programming in Java

- Client implementation
  - Make a service request to the server

    Socket *socket*= ***new Socket(server, port);***
  - Data exchange using input/output streams
    - Same way as file input/output
    - ***socket.getInputStream( )***
    - ***socket.getOutputStream( )***
  - Close the streams and socket
    - close methods of the streams and socket

Client/server interaction and Socket

- Server application is bounded to a specific port
  - Then waits, listening to the socket for a client's request
- Client makes a connection request to the (server, port)

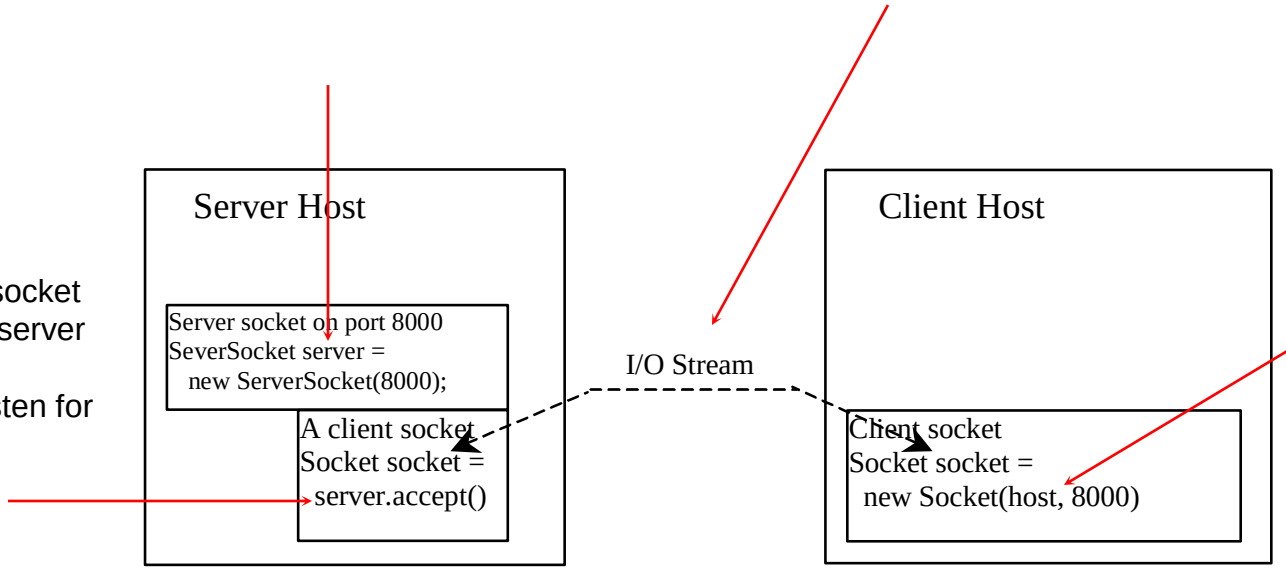


- Server accepts the request and creates a new private socket for communication.

Client/Server Communications

The server must be running when a client starts. The server waits for a connection request from a client. To establish a server, you need to create a server socket and attach it to a port, which is where the server listens for connections.

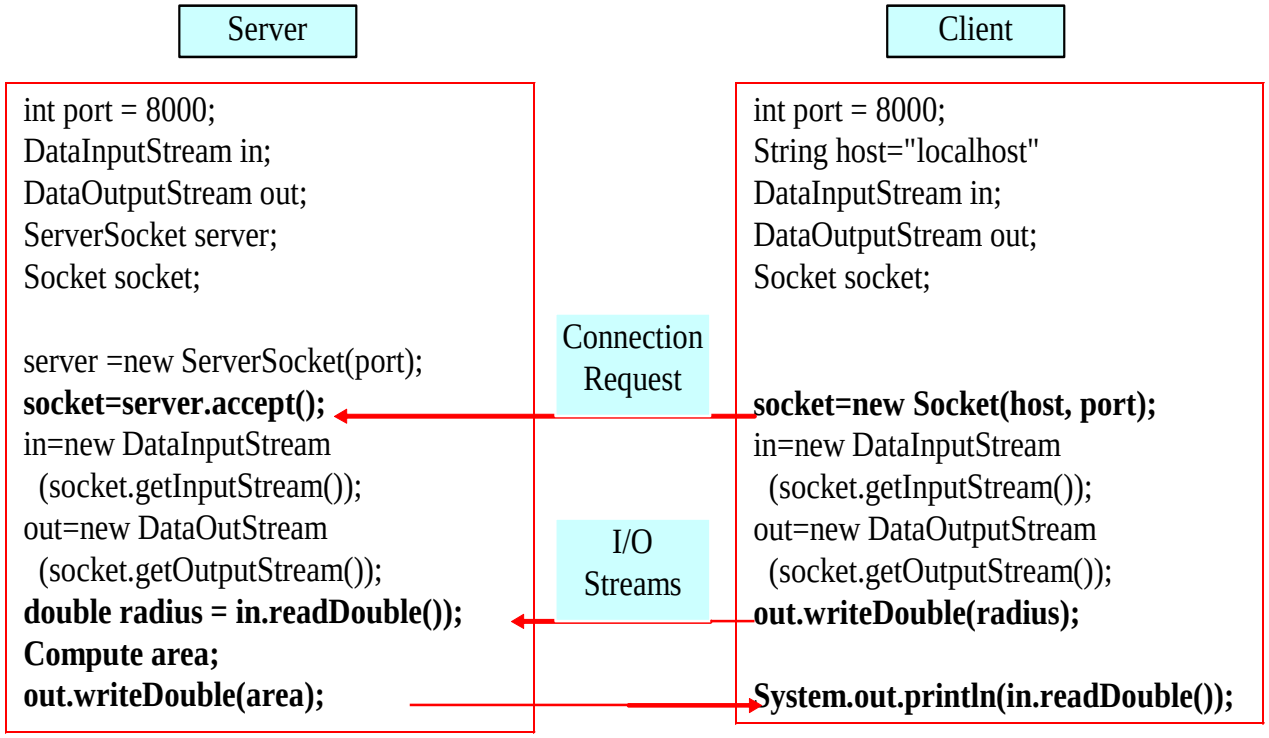
After the server accepts the connection, communication between server and client is conducted the same as for I/O streams.



After a server socket is created, the server can use this statement to listen for connections.

The client issues this statement to request a connection to a server.

Data Transmission through Sockets




```
InputStream input = socket.getInputStream();
OutputStream output = socket.getOutputStream();
```

Something New for Interested?

- **Web Socket API**
  - **http://www.tutorialspark.com/html5/HTML5_WebSockets.php**
  - New to HTML5
  - Allow the high-level communications based HTTP protocol, between a web browser and a web server
- **WebRTC (Web Real-Time Communication)**
  - a free, open-source project that provides web browsers and mobile applications with real-time communication (RTC) via simple APIs.
  - https://webrtc.org/

Compute area—server part

- **codes/java/AreaServer.java**

```
import java.io.*; import java.net.*;
public class AreaServer{
  private ServerSocket serverSocket;
  public static void main ( String [] args ){
     new AreaServer( 8000 );
  }
  public AreaServer( int port ){
      serverSocket = new ServerSocket( port ); // create a server socket
       Socket socket = serverSocket.accept( ) ; // listen for a connection
       // create data input/output streams
       DataInputStream inputFromClient = new DataInputStream(
                                                    socket.getInputStream( ));
      DataOutputStream outputToClient = new DataOutputStream(
                                                    socket.getOutputStream( ) );
       double radius = inputFromClient.readDouble(); //receive radius from client
       double area = Math.PI * radius * radius; //compute area = pi * radius ^2
       outputToClient.writeDouble( area ); //send area to client
}}
```

Compute area—client part

- **codes/java/AreaClient.java**

```
import java.io.*,  java.net.*,  java.util.Scanner;
public class AreaClient{
    public static void main ( String [] args ){
       new AreaClient(  server, port );
}
public AreaClient( String server, int port ){
      DataInputStream inputFromServer; DataOutputStream outputToServer;
      socket = new Socket( server, port ); // create a socket to connect to the
     server Socket
      // create data input/output streams
      inputFromServer = new DataInputStream( socket.getInputStream( ) );
      outputToServer = new DataOutputStream( socket.getOutputStream( ) );
      Scanner sc = new Scanner ( System.in ); // create Scanner object for
     reading
      double radius = sc.nextDouble( ); // get a radius
      outputToServer.writeDouble( radius ); outputToServer.flush( ); // send
     radius to sever
     double area = inputFromServer.readDouble( ); // get area from server
      System.out.println("Radius is " + radius + " Area is " + area);
} }
```

Compute area— limitation

- **Only one area is computed**
  - **Server exits**
  - **Using a loop for the server running forever**
- **Any time, only one client can be serviced by the server**
  - Using threaded server to provide the concurrent services to multiple clients
  - You learn multi-threading in Operating Systems part later

Sequential Server

- Server completely handles one request before accepting additional requests
  - Also called iterative
  - Single thread (process) of control
  - Handles one request at a time

- Pseudo-code

```
for ( ; ; ){
  wait for a client request on the listening port
  create a private duplex communication channel to the client
  while (no error on the private communication channel)
      read from the client
      process the request
      respond to the client
  shutdown the connection
 }
```

- Example– Area client/server applications

WireShark to Capture the live packets

- Start the server AreaServer first ➔ IP= ?, port = ?
  java AreaServer if using port of 8000
  java AreaServer 6000 // using port 6000

- Start WireShark protocol analyzer, and start to capture the live packets

- Start AreaClient
  java AreaClient 194.47.40.104 8000

- Enter radius and watch the result of the area

- Stop WireShark for capturing the packet
- Enter display filter tcp.port == 8000 && data
- View and analyze the data exchange between client and server apps.
- Use online conversion from hexadecimal double representation to decimal representation to verify the results http://babbage.cs.qc.cuny.edu/IEEE-754/
- Question: can client and server applications written in different languages (java and C, for example)?

Concurrent Server

- A parent server creates a child process to handle the actual service to the client
  - –Parent server can respond to multiple clients at the same time
- It can be threaded model
  - –Main thread create a new thread for handling a new request
- Execution proceeds “in parallel”
  - –Can handle multiple requests at a time

References on Java networking programming

- Java Networking Overview

http://docs.oracle.com/javase/8/docs/technotes/guides/net/overview/overview.html

Socket Programming in Java: a tutorial,
http://www.javaworld.com/article/2077322/core-java/core-java-sockets-programming-in-java-a-tutorial.html

- Java Tutorial– All about sockets

http://docs.oracle.com/javase/tutorial/networking/sockets/index.html

- Java I/O– any Java textbook

Summary

- What you have learnt in this lecture?