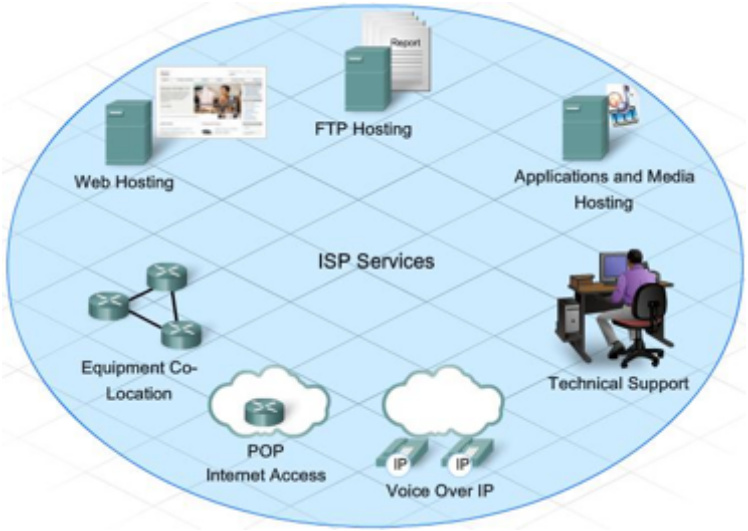
Chapter 3

Internet Applications and Network Programming

Introduction

- The Internet offers users a rich diversity of services
  - none of the services is part of the underlying communication infrastructure
- 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



Two Basic Internet Communication Paradigms

- The Internet supports two basic communication paradigms:
  - Stream Transport in the Internet
  - Message Transport in the Internet

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

1-to-1/ One-to-Many or multicasting

Stream Transport in the Internet

- Stream denotes a paradigm in which a sequence of bytes flows from one application program to another
  - without inserting boundaries
    - thus, there is no frame concept!
  - can choose to generate one byte at a time, or can generate blocks of bytes
- The network chooses the number of bytes to deliver at any time
  - the network can choose to combine smaller blocks into one large block or can divide a large block into smaller blocks

Message Transport in the Internet

- In a message paradigm, the network accepts and delivers messages
  - 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 (in anycasting destination is not identified)
  - 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

- Most applications require delivery guarantees
- 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

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

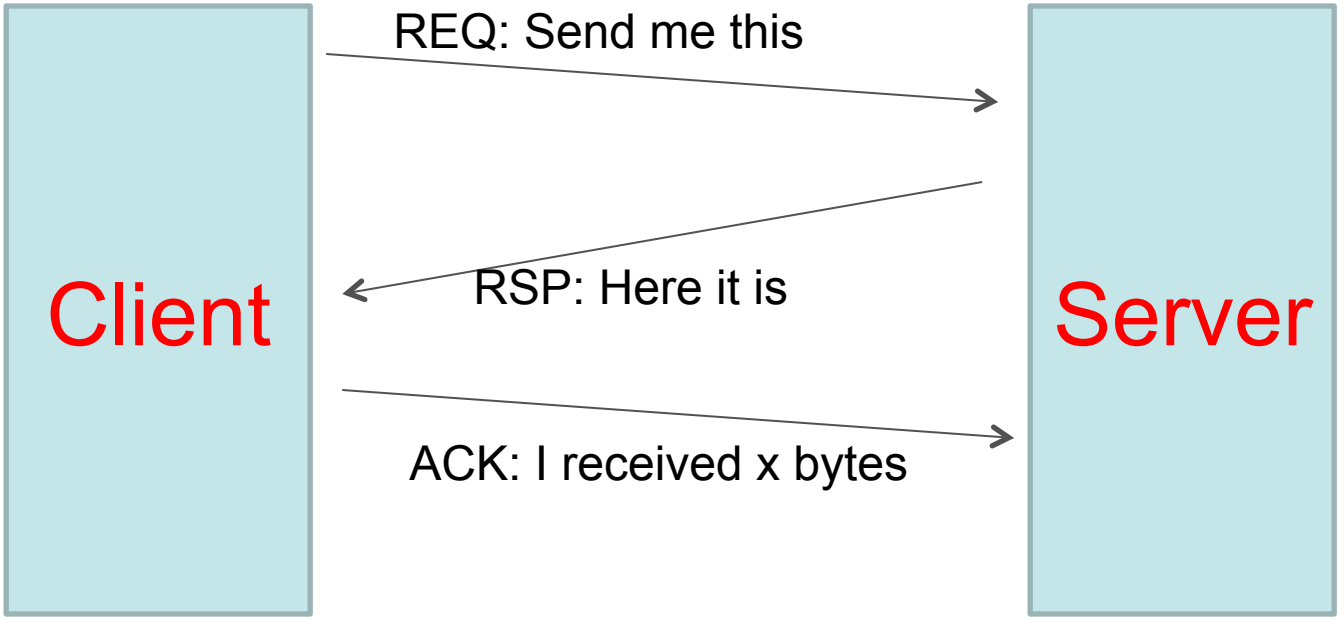
The Client-Server Model of Interaction

- Main question:
  - how can a pair of applications that run on two independent computers coordinate to guarantee that they request a connection at the same time?
- The answer lies in a form of interaction known as the client-server model
  - First: A server starts and awaits contact
  - Second: A client starts and initiates the connection

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

Evolution: main frames / PC / Client-Server / Cloud!

The Client-Server Model of Interaction



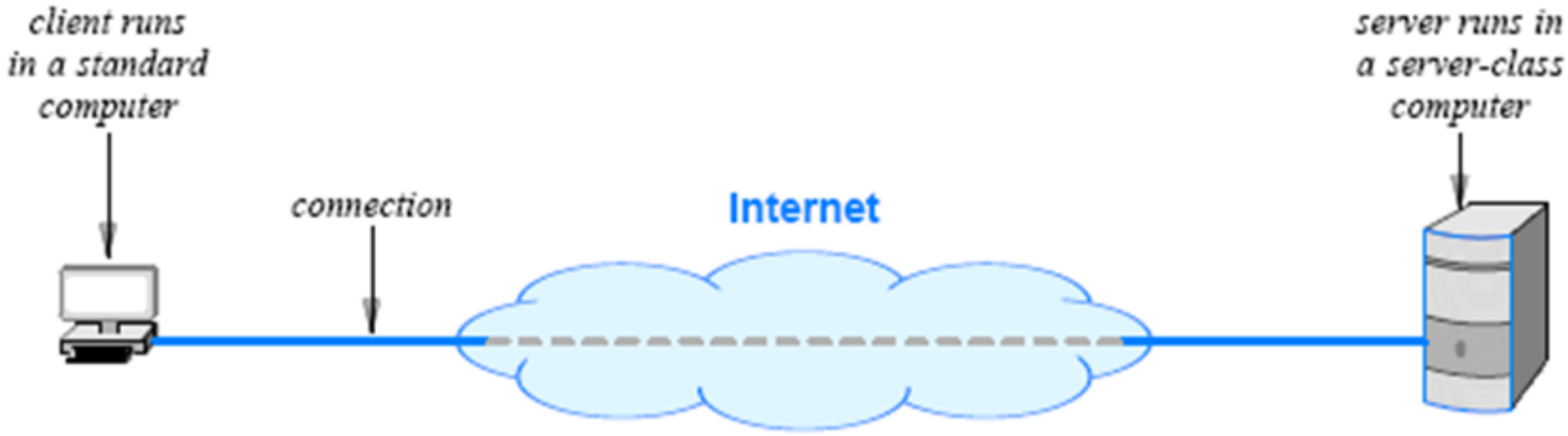
Characteristics of Clients and Servers

- Most instances of client-server interaction have the same general characteristics
- A client software:
  - Is an arbitrary application program that becomes a client temporarily when remote access is needed
  - Is invoked directly by a user, and executes only for one session
  - Runs locally on a user's personal computer
  - Actively initiates contact with a server
  - Can access multiple services as needed, but usually contacts one remote server at a time
  - Does not require especially powerful computer hardware
- A server software:
  - Is a special-purpose, privileged program
  - Is dedicated to providing one service that can handle multiple remote clients at the same time
  - Is invoked automatically when a system boots, and continues to execute through many sessions
  - Runs on a large, powerful computer
  - Waits passively for contact from arbitrary remote clients
  - Accepts contact from arbitrary clients, but offers a single service
  - Requires powerful hardware and a sophisticated operating system (OS)

Example: Outlook and Exchange server / Browser and Web server

Requests, Responses, and Direction of Data Flow

- Once contact has been established, two-way communication is possible (i.e., data can flow from a client to a server or from a server to a client)
- In some cases, a client sends a series of requests and the server issues a series of responses (e.g., a database client might allow a user to look up more than one item at a time)

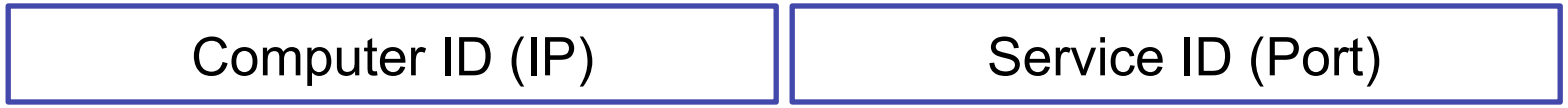


Multiple Clients and Multiple Servers

- 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
  - In many cases 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

Server Identification and Demultiplexing

- How does a client identify/find 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 (application) 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 (132.98.12.70)
- 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



Server Identification and Demultiplexing

4 • Start after server is already running

5 Obtain server name from user

6 Use DNS to translate name to IP address

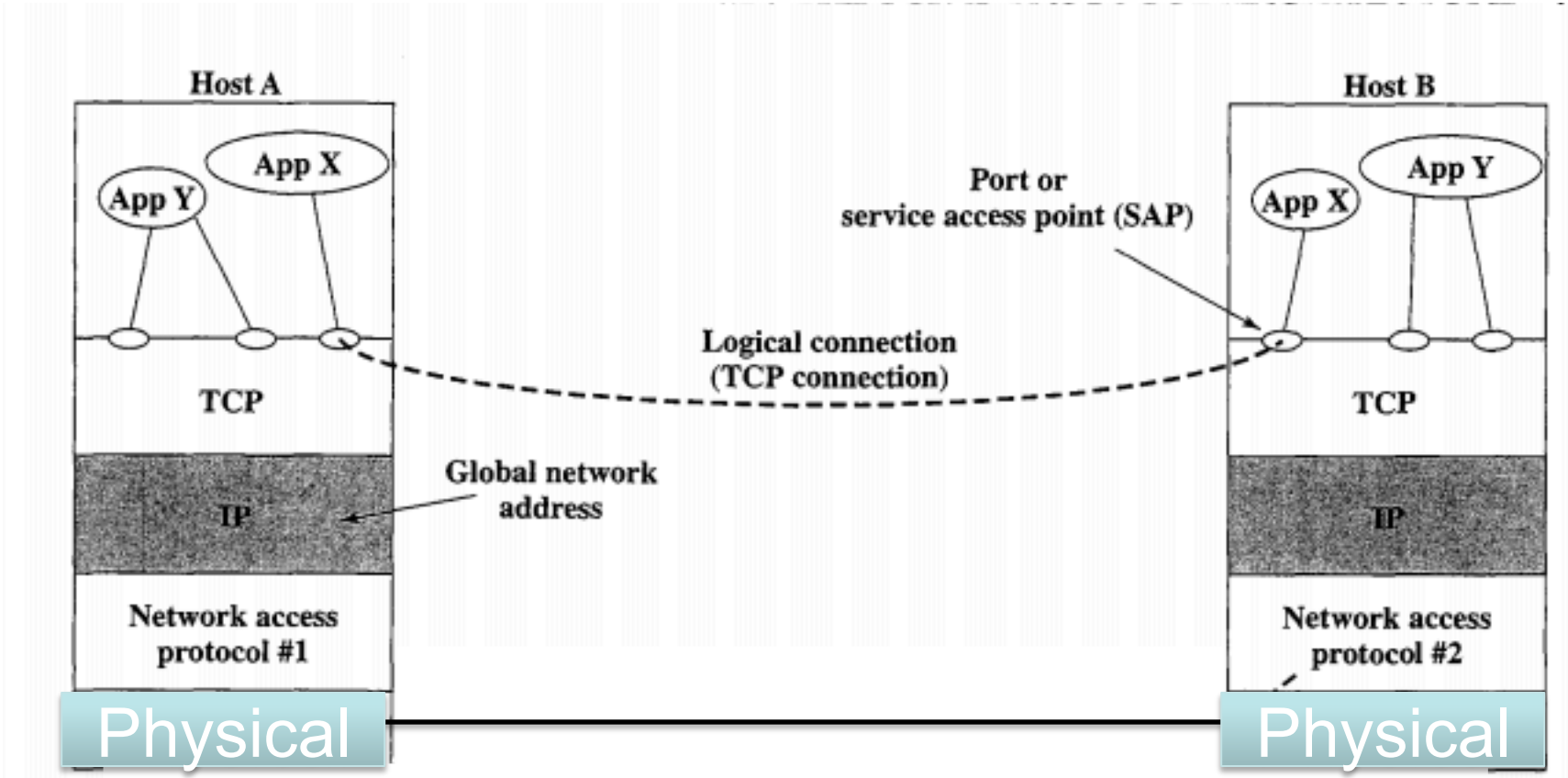
- Specify that the service uses port N
- Contact server and interact

- Start before any of the clients 1
- Register port N with the local system 2
- Wait for contact from a client 3
- Interact with client until client finishes
- Wait for contact from the next client…

See Notes!

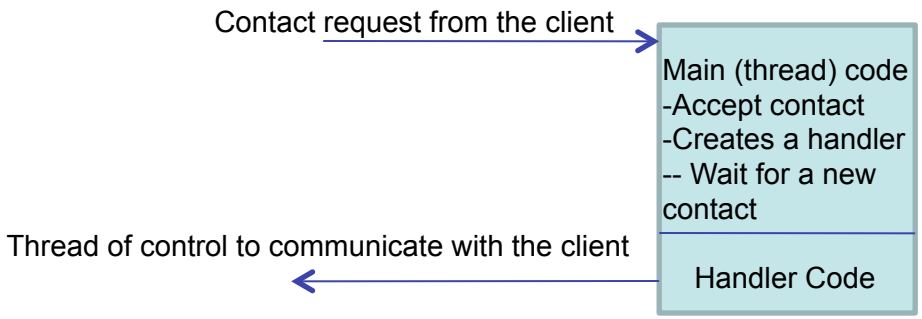
DNS: Domain Name Server
http://www.who.is/whois-lookup/

Applications and Ports



Concurrent Servers

- Most servers are concurrent
  - That is, a server uses more than one thread of control
- Concurrent execution depends on the OS being used
- Concurrent server code is divided into two pieces
  - a main program (thread)
  - a handler
- The main thread accepts contact from a client and creates a thread of control for the client
- Each thread of control interacts with a single client and runs the handler code

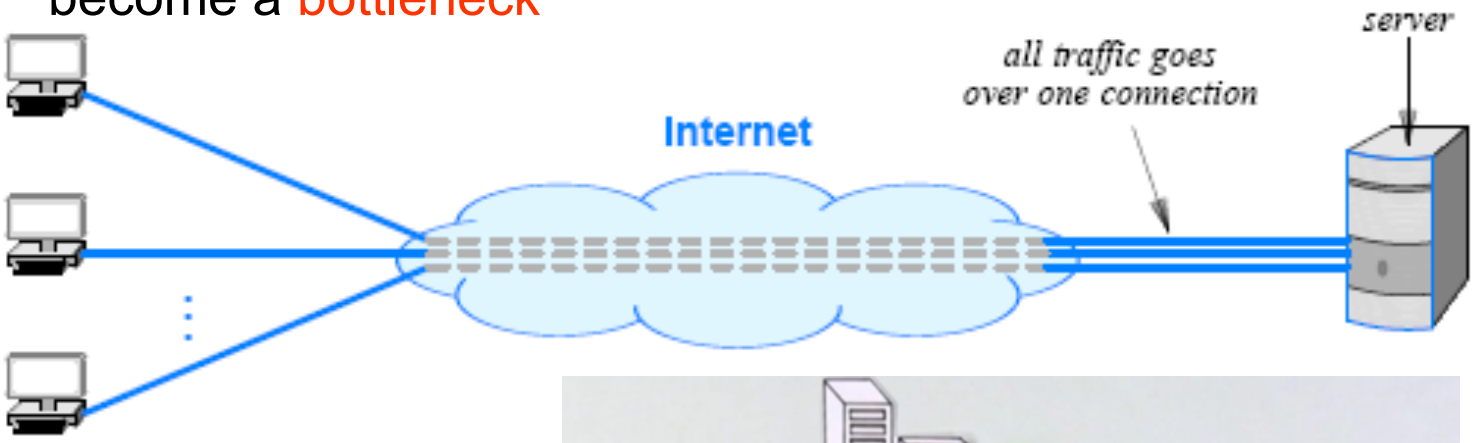


Concurrent Servers

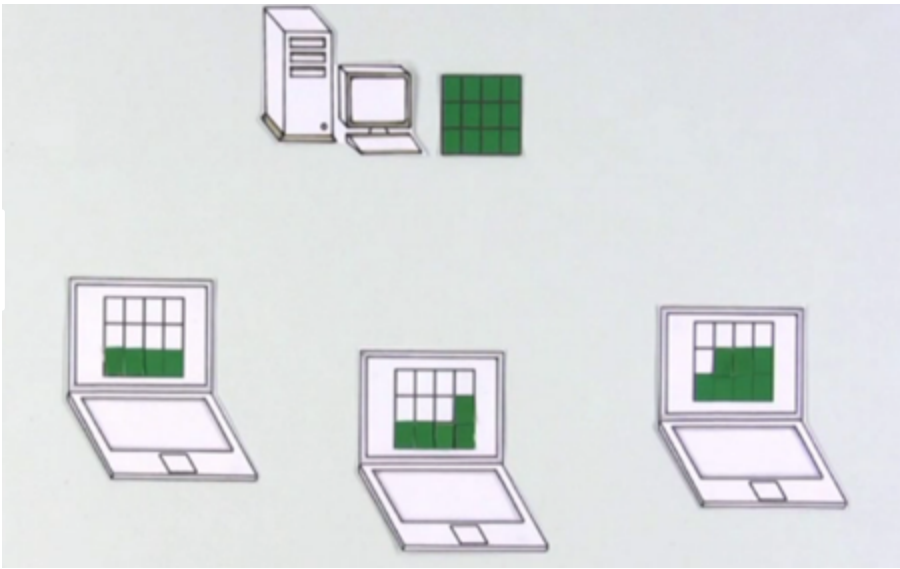
- After handling one client the thread terminates
- The main thread keeps the server alive after creating a thread to handle a request
  - the main thread waits for another request to arrive
- If N clients are simultaneously using a concurrent server, N +1 threads will be running:
  - the main thread (1) is waiting for additional requests
  - and N threads are each interacting with a single client

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



Central Bottleneck → Slow download!

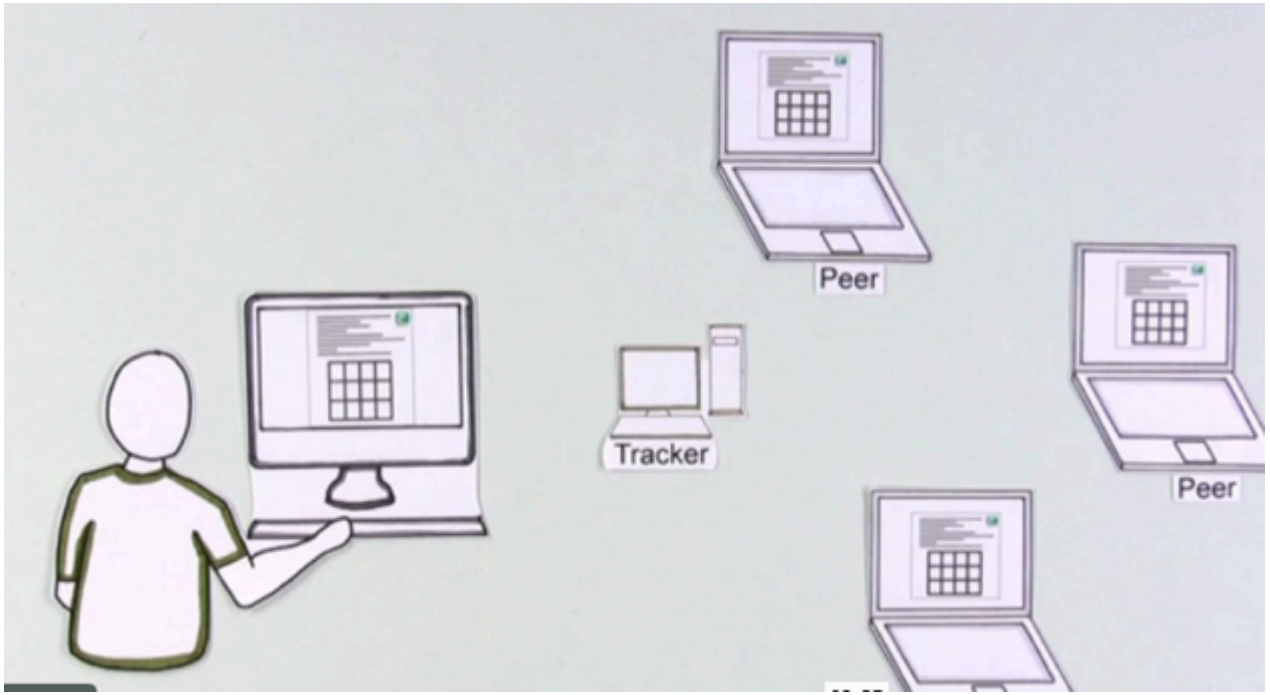


Peer-to-Peer Interactions

- 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

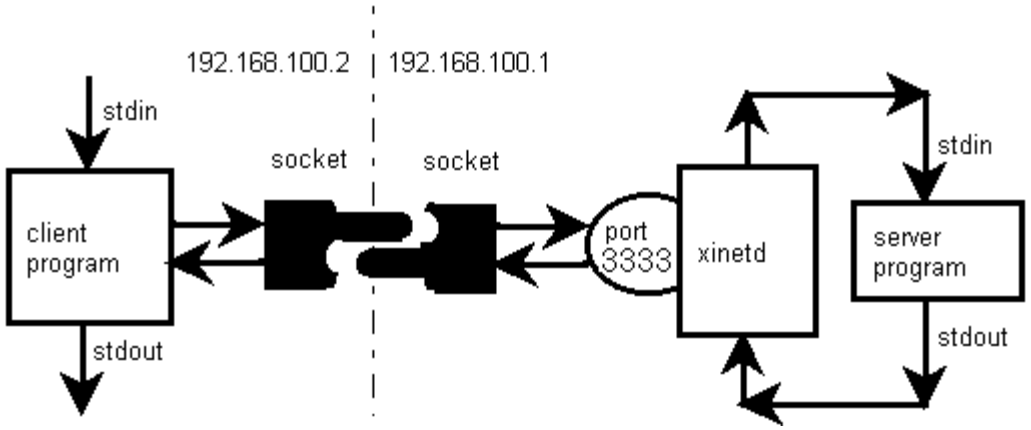
Bittorrent

- Smarter Peer-to-Peer approach
- A browser-like application to download files faster
  - http://player.vimeo.com/video/15228767
- More users downloading results in faster download!
- Eliminate the central bottleneck



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
- One particular API has emerged as the de facto standard for software that communicates over the Internet
  - known as the socket API, and commonly abbreviated sockets

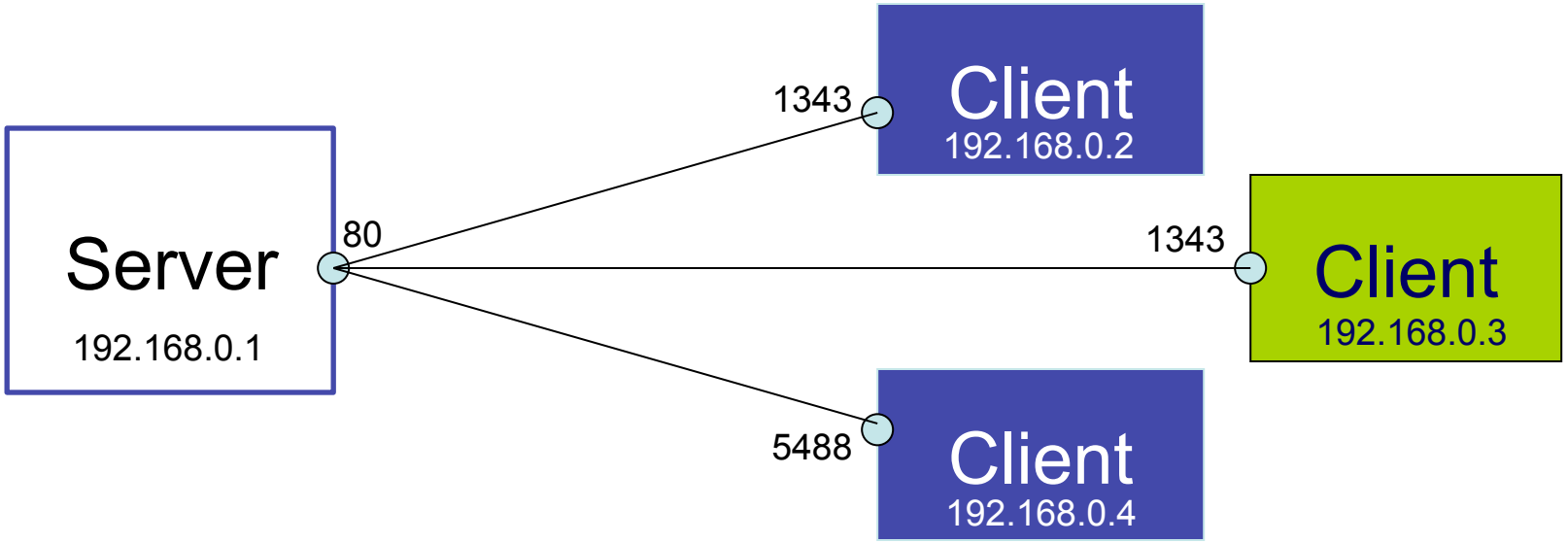


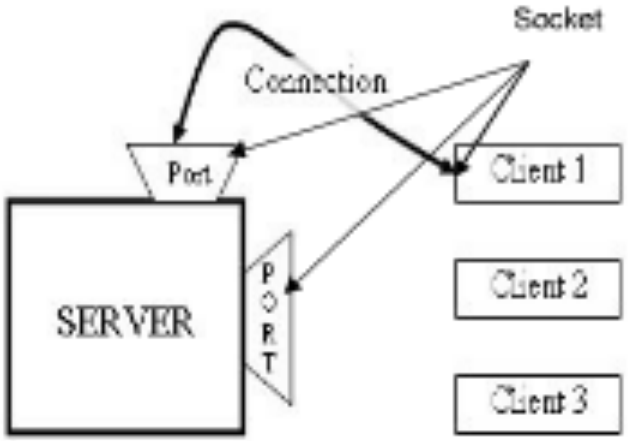
Sockets are a software methodology to connect different processes (programs) on the same computer or on different computers. The name "socket" reminds us that once we "plug in" one process into another process's socket, they can talk to each other by reading and writing the socket.

http://www.troubleshooters.com/codecorn/sockets/#Introduction

Introduction to Sockets

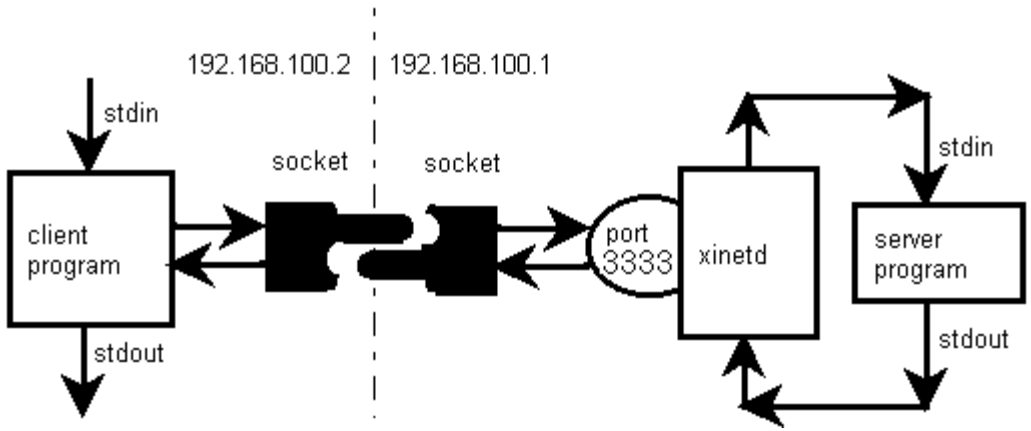
- What exactly creates a Socket?
  - <IP address, Port #> tuple
- What makes a connection?
  - {Source<IP address, Port #> , Destination <IP address, Port #>} i.e. source socket – destination socket pair uniquely identifies a connection.
- Example





Network Programming and the Socket API

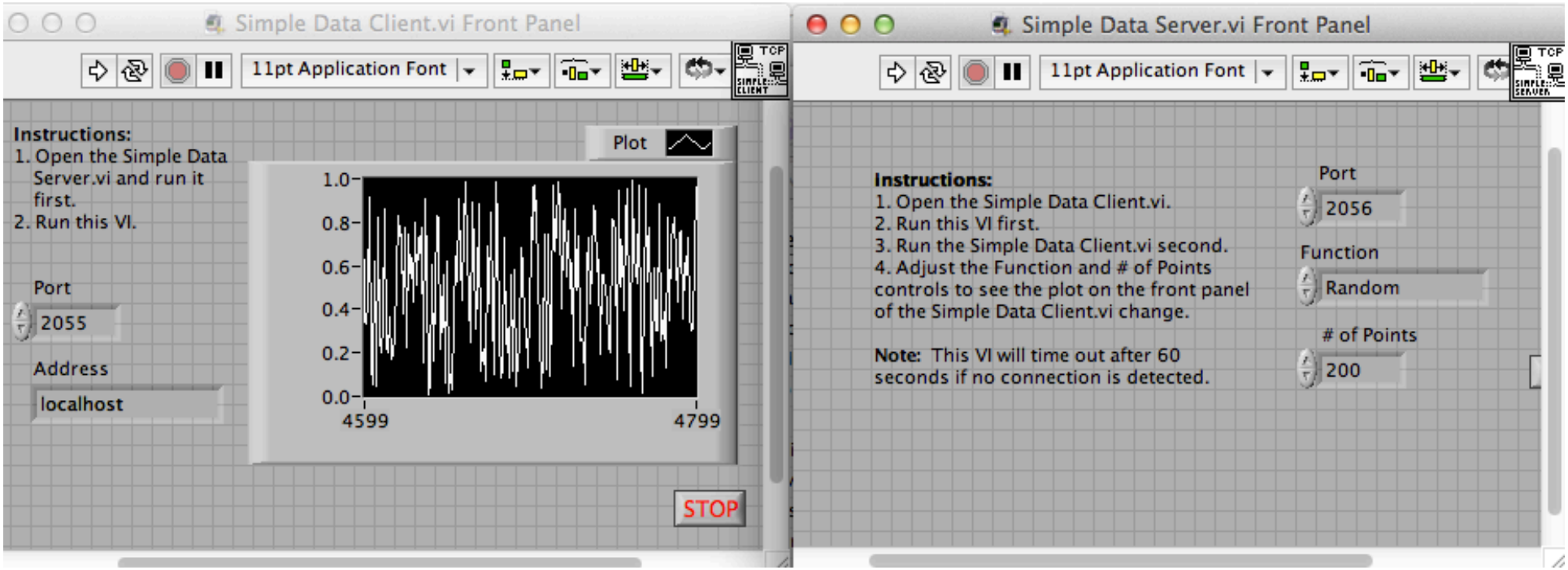
- telnet 192.168.100.1 3333



Using telnet as the client program → connecting the client to port 3333 on 192.168.100.1

Example

- In this case we have two application programs:
  - Server and client
  - Note that they both have to be pointing to the same port!



Sockets, Descriptors, and Network I/O

- When an application creates a socket to use for Internet
  - OS returns a small integer descriptor that identifies the socket
- The application then passes the descriptor as an argument
  - when it calls functions to perform an operation on the socket (e.g., to transfer data across the network or to receive data)
- An application must specify many details, such as
  - the address of a remote computer
  - the protocol port number
  - and whether the application will act as a client or as a server
- To avoid having a single socket function with many parameters, designers of the socket API chose to define many functions
- Remember: An application creates a socket, and then invokes functions for details

Socket Example

- Creating a set of functions that handle communications to write network applications
  - Socket<PrototypeFamily Type Protocol>
  - Send<socket data length flag>

  - PrototypeFamily: TCP or UDP
  - Type: Connectionless
  - Protocol: Transport

Name	Used By	Meaning
accept	server	Accept an incoming connection
bind	server	Specify IP address and protocol port
close	either	Terminate communication
connect	client	Connect to a remote application
getpeername	server	Obtain client's IP address
getsockopt	server	Obtain current options for a socket
listen	server	Prepare socket for use by a server
recv	either	Receive incoming data or message
recvmsg	either	Receive data (message paradigm)
recvfrom	either	Receive a message and sender's addr.
send (write)	either	Send outgoing data or message
sendmsg	either	Send an outgoing message
sendto	either	Send a message (variant of sendmsg)
setsockopt	either	Change socket options
shutdown	either	Terminate a connection
socket	either	Create a socket for use by above

Figure 3.7 A summary of the major functions in the socket API

Programming Client-Server Using Sockets

- Different programming languages can be used: C, JAVA, Shell