Chapter 3

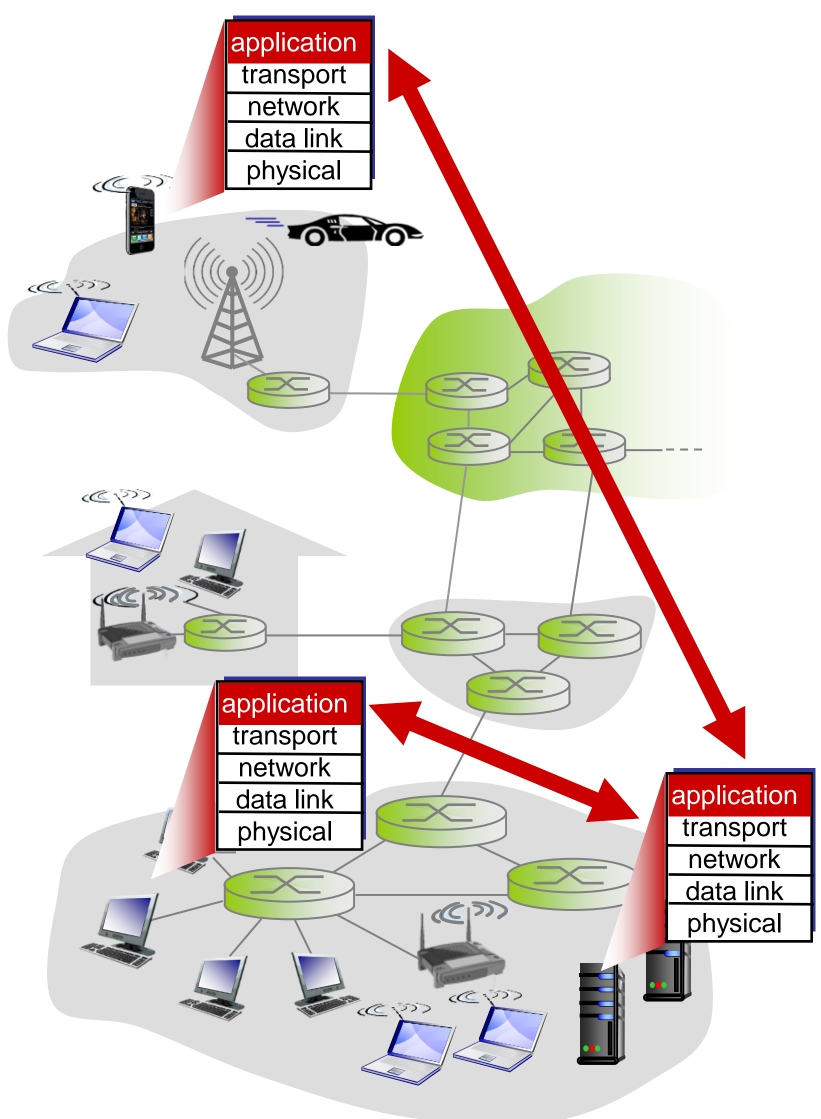
Internet Applications & Network Programming

Topics Covered

- 3.1 Introduction
- 3.2 Two Basic Internet Communication Paradigms
- 3.3 Connection-Oriented Communication
- 3.4 The Client-Server Model of Interaction
- 3.5 Characteristics of Clients and Servers
- 3.6 Server Programs and Server-Class Computers
- 3.7 Requests, Responses, and Direction of Data Flow
- 3.8 Multiple Clients and Multiple Servers
- 3.9 Server Identification and Demultiplexing
- 3.10 Concurrent Servers
- 3.11 Circular Dependencies Among Servers
- 3.12 Peer-to-Peer Interactions
- 3.13 Network Programming and the Socket API
- 3.14 Sockets, Descriptors, and Network I/O
- 3.15 Parameters and the Socket API
- 3.16 Socket Calls in a Client and Server

3.1 Introduction

- The Internet offers users a rich diversity of services
- Services are not part of the underlying communication infrastructure
- Internet provides a general purpose mechanism on which all services are built
- It is possible to create Internet applications without knowing how networks operate
- network-core devices do not run user applications
  - Such as socket Application Programming Interface (socket API)
- However, understanding network protocols and technologies allows them to write efficient and reliable code


application
transport
network
data link
physical
application
transport
network
data link
physical
application
transport
network
data link
physical

3.2 Two Basic Internet Communication Paradigms

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

3.2.1 Stream Transport in the Internet

- Stream denotes flow of a sequence of bytes
- The network accepts an input stream from either application, and delivers the data to another application
- The stream mechanism transfers a sequence of bytes without attaching meaning to the bytes and without inserting boundaries
- A sending application 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

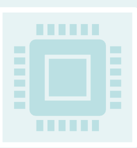
3.2.2 Message Transport in the Internet

- In a message paradigm, the network accepts and delivers message blocks
- 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

3.2.2 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**



A programmer who uses the message paradigm must insure that the application operates correctly even if packets are lost or reordered

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 and connectionless

Criteria	Connection-Oriented	Connection-Less
Connection	Prior connection needs to be established.	No prior connection is established.
Resource Allocation	Resources need to be allocated.	No prior allocation of resource is required.
Reliability	It ensures reliable transfer of data.	Reliability is not guaranteed as it is a best effort service.
Congestion	Congestion is not at all possible.	Congestion can occur likely.
Transfer mode	It can be implemented either using Circuit Switching or VCs.	It is implemented using Packet Switching.
Retransmission	It is possible to retransmit the lost data bits.	It is not possible.
Suitability	It is suitable for long and steady communication.	It is suitable for bursty transmissions.
Signaling	Connection is established through process of signaling.	There is no concept of signaling.
Packet travel	In this packets travel to their destination node in a sequential manner.	In this packets reach the destination in a random manner.
Delay	There is more delay in transfer of information, but once connection	There is no delay due absence of connection establishment phase.

We’ll learn the basic vocabulary for talking about how networks perform

3.3 Connection-Oriented Communication

- The Internet stream service is connection-oriented
- It operates analogous to a telephone call:
  - two applications must request that a connection be created
  - applications can send /receive data in either direction
  - when they finish communicating, the applications request that the connection be terminated
- Algorithm 3.1 summarizes the interaction

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.

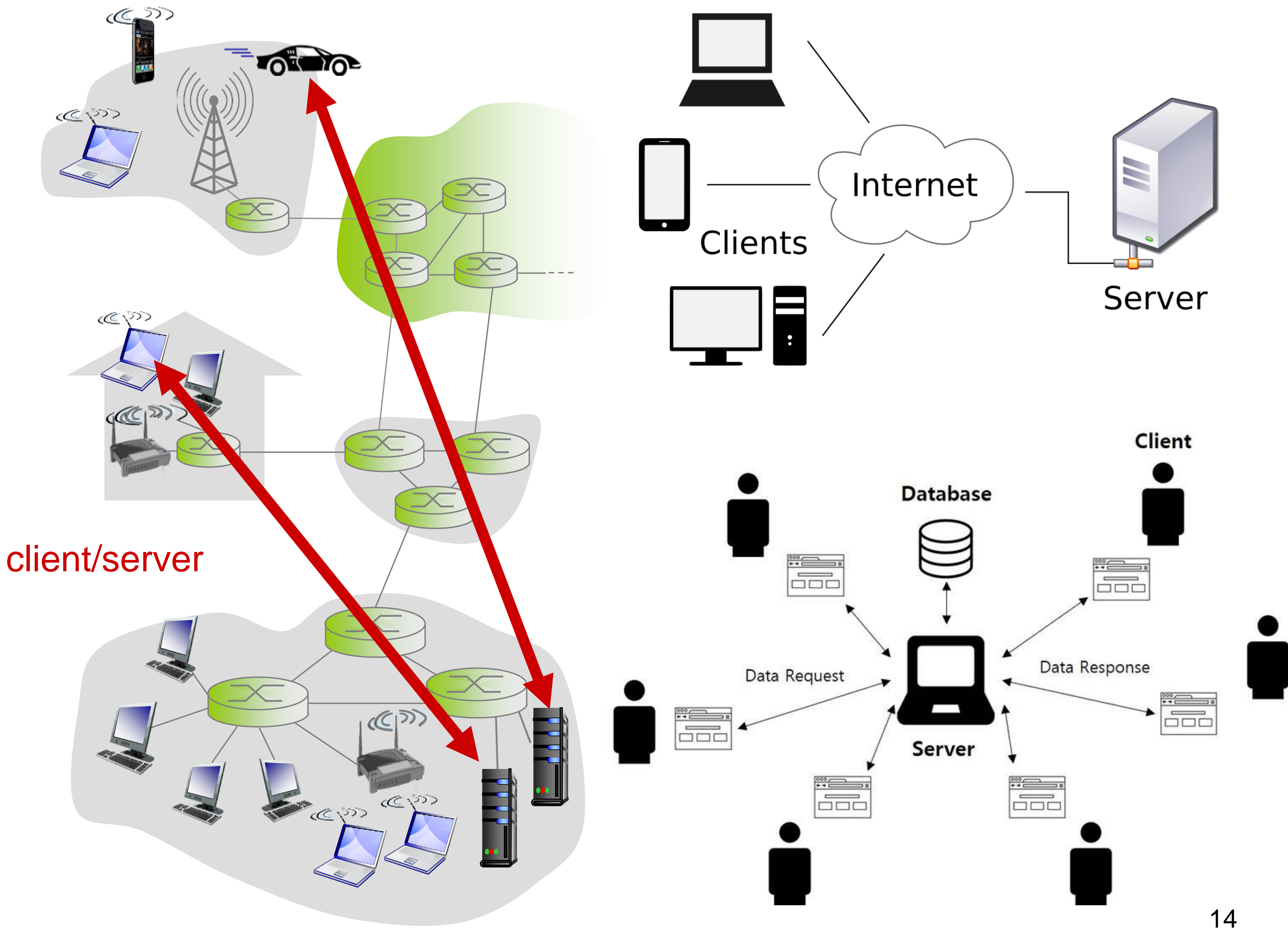
3.4 The Client-Server Model of Interaction

- 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
  - A server starts first and awaits contact
  - A client starts second and initiates the connection
- Figure 3.2 summarizes the interaction
- Subsequent sections describe how specific services use the client-server model
- Application programs known as clients and servers handle all services in the Internet

3.4 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.


client/server
Internet
Clients
Server
Client
Database
Data Request
Data Response
Server

3.5 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, but also performs other computation
  - 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

3.5 Characteristics of Clients and Servers

- 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)

3.6 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

3.6 Server Programs and Server-Class Computers

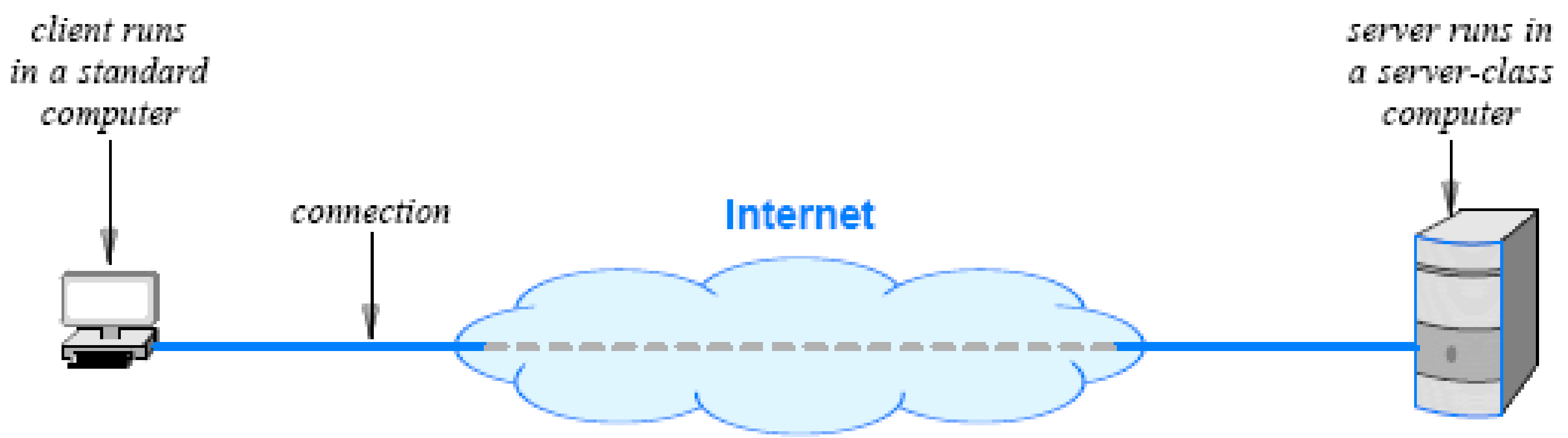
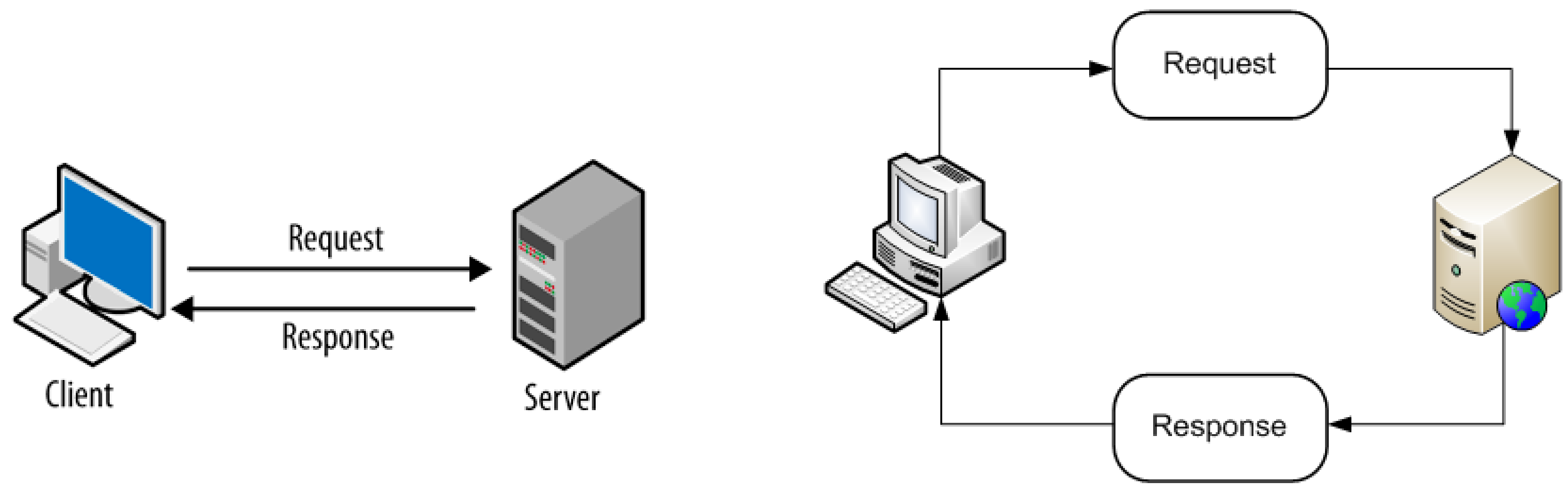


Figure 3.3 Illustration of a client and server.

3.7 Requests, Responses, and Direction of Data Flow

- Which side initiates contact? Client and server?
- Once contact is established, two-way communication is possible
- In some cases, a client sends a series of requests and the server issues a series of responses

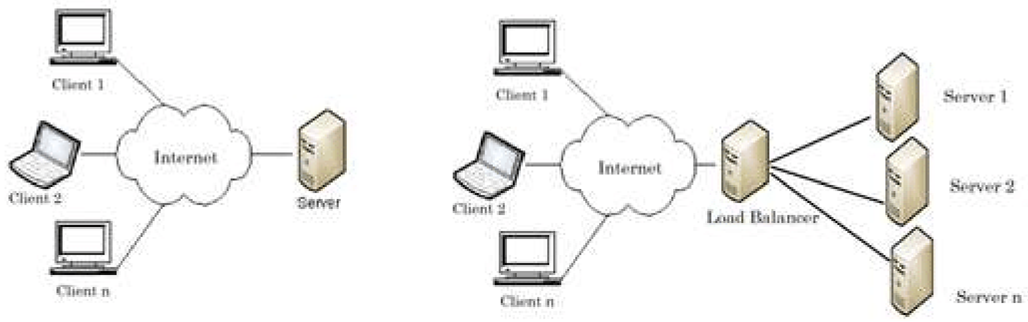


3.8 Multiple Clients and Multiple Servers

- 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

3.8 Multiple Clients and Multiple Servers

- A computer can run:
  - A single client
  - A single server
  - Multiple copies 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

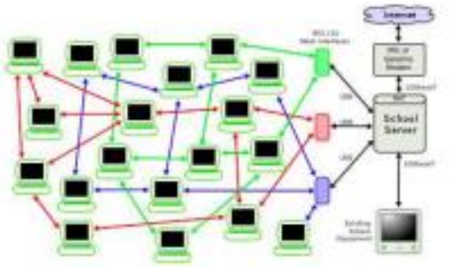


3.9 Server Identification and Demultiplexing

- 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)
  - To make server identification easy for humans, each computer is also assigned a name, and the Domain Name System (DNS)
- 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)
  - Figure 3.4 summarizes the basic steps in identifying an application

So far you know how to build a Local Area Network

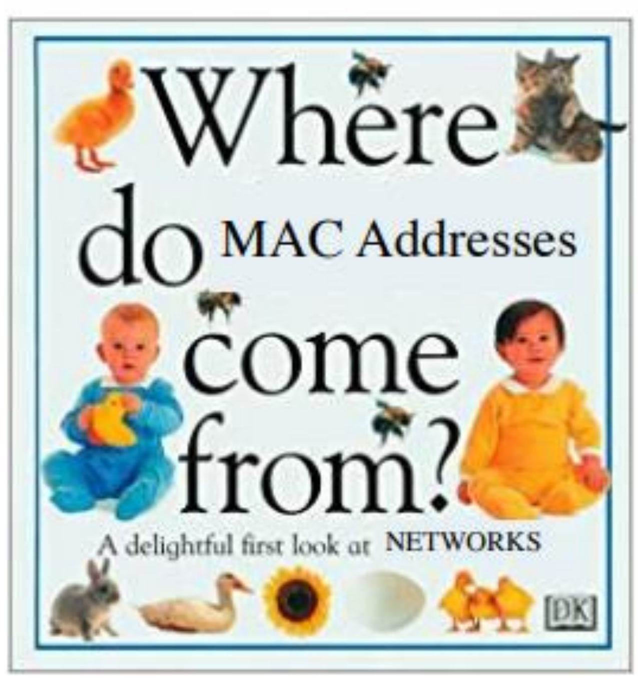
How do we get them to talk to each other?



MAC Addresses

- “Media Access Control Address”
- 48 bits long, written as a sequence of hexadecimal numbers
  - e.g. 34:f3:e4:ae:66:44
- Quick — how many possible MAC addresses are there?
  - 281,474,976,710,656
- Used as part of a protocol called *Ethernet*.

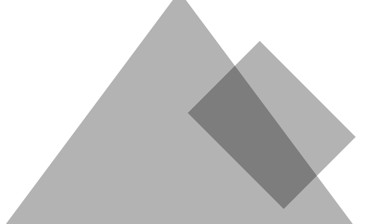
MAC Addresses



All MAC addresses are assigned by your device's manufacturer.

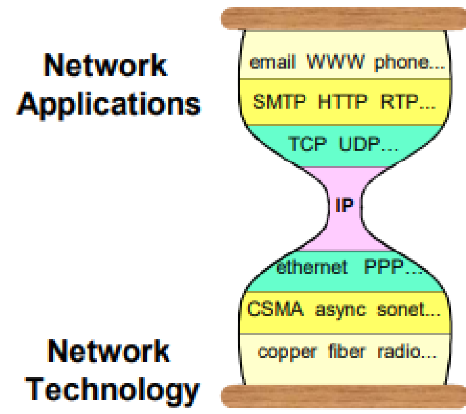
Every wireless adaptor, ethernet port, bluetooth connector you have has a unique number assigned to it by the manufacturer.

I found this crazy when I learned this!



Solution: Internet Protocol (IP)

- Inter-network connectivity provided by the Internet protocol
- Hosts use Internet Protocol to send packets destined across networks.
- IP creates abstraction layer that hides underlying technology from network application software
  - Allows range of current & future technologies
  - WiFi, traditional and switched Ethernet, personal area networks, …

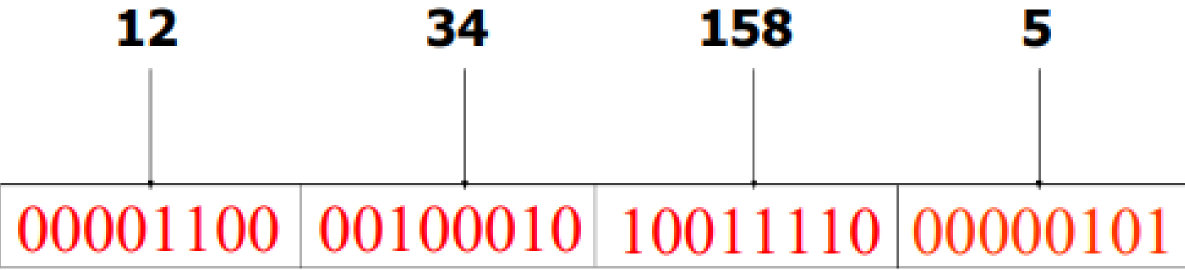


IP Addresses (IPv4)

- Unique 32-bit number associated with a host

00001100 00100010 10011110 00000101

- Represented with the "dotted quad" notation
  - e.g., 12.34.158.5

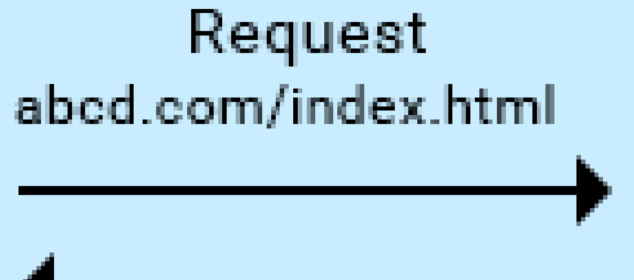


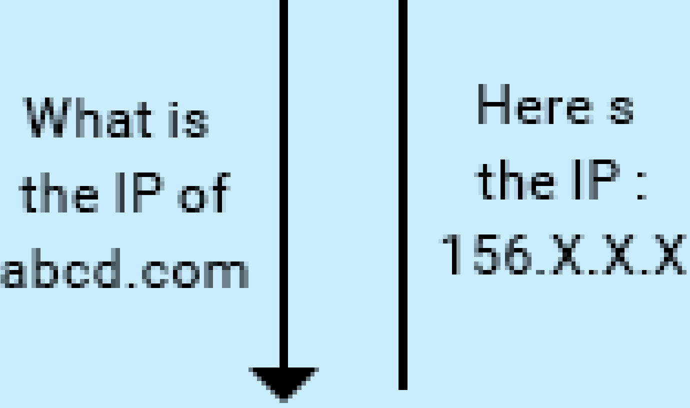
3.9 Server Identification and Demultiplexing

- 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.

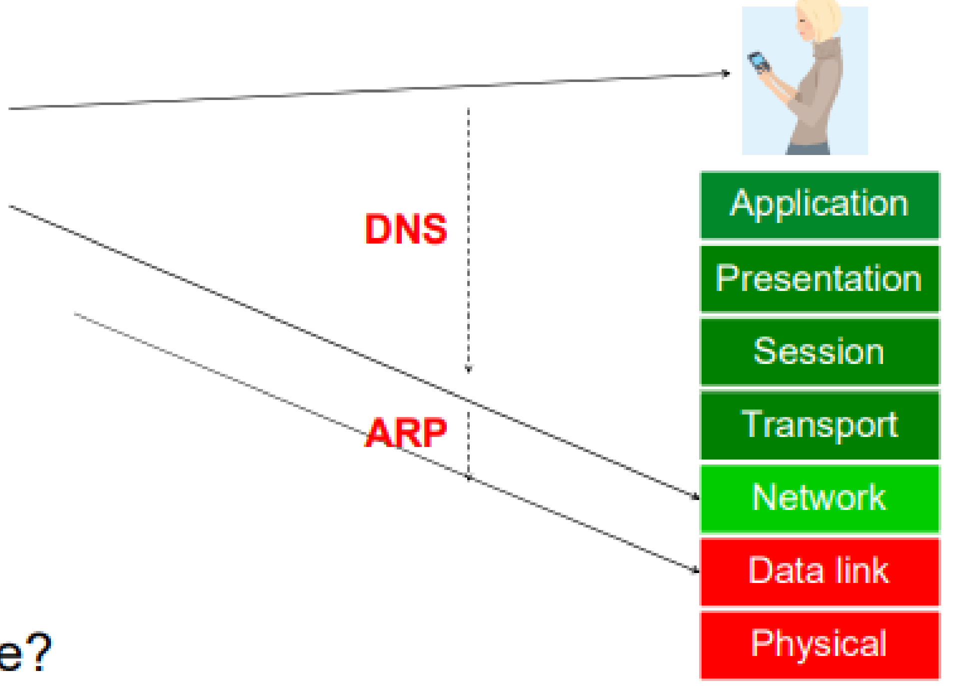




What is a DNS Server ?

Too Much of a Good Thing?

- Hosts have a
  - host name
  - IP address
  - MAC address
- There is a reason ..
  - Remember?
- But how do we translate?



Host Names & Addresses

- Host addresses: *e.g., 169.229.131.109*
  - a number used by protocols
  - conforms to network structure (the "where")
- Host names: *e.g., linux.andrew.cmu.edu*
  - mnemonic name usable by humans
  - conforms to organizational structure (the "who")
- The Domain Name System (DNS) is how we map from one to the other
  - a directory service for hosts on the Internet

DNS provides Indirection

- Addresses can change underneath
  - Move www.cnn.com to a new IP address
  - Humans/apps are unaffected
- Name could map to multiple IP addresses
  - Enables load-balancing
- Multiple names for the same address
  - E.g., many services (mail, www, ftp) on same machine
- Allowing “host” names to evolve into “service” names

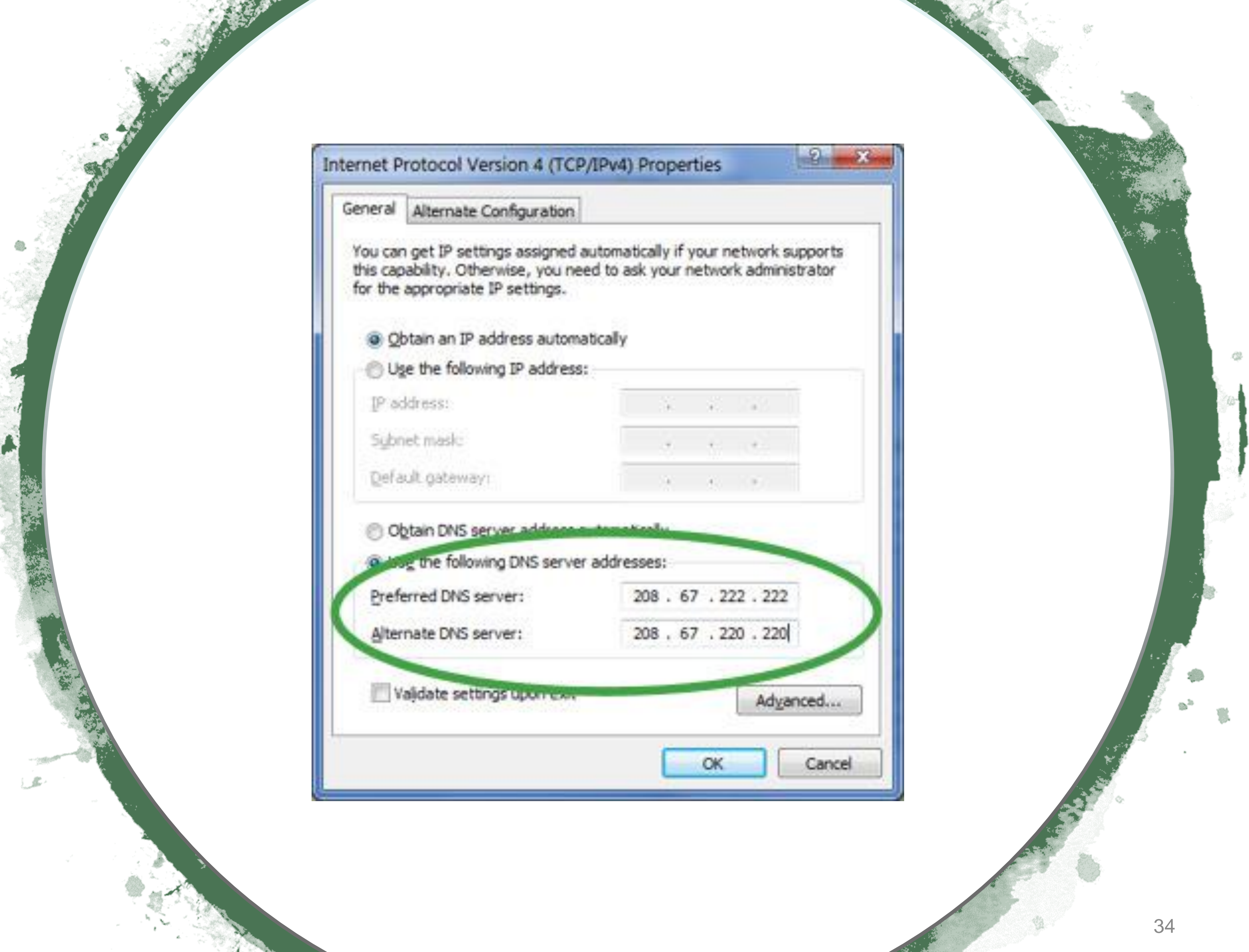
Command Prompt

```
Microsoft Windows [Version 10.0.17134.228]
(c) 2018 Microsoft Corporation. All rights reserved.

Z:\>NSLOOKUP google.com
Server:  dc1.ishik.edu.iq
Address:  10.1.1.1

Non-authoritative answer:
Name:    google.com
Addresses:  2a00:1450:4017:800::200e
          172.217.17.174


Z:\>_
```


Internet Protocol Version 4 (TCP/IPv4) Properties
General
Alternate Configuration
You can get IP settings assigned automatically if your network supports this capability. Otherwise, you need to ask your network administrator for the appropriate IP settings.
Obtain an IP address automatically
Use the following IP address:
IP address:
Subnet mask:
Default gateway:
Obtain DNS server
the following DNS server addresses:
Preferred DNS server:
208 . 67 . 222 . 222
Alternate DNS server:
208 . 67 . 220 . 220
Validate settings
Advanced...
OK
Cancel

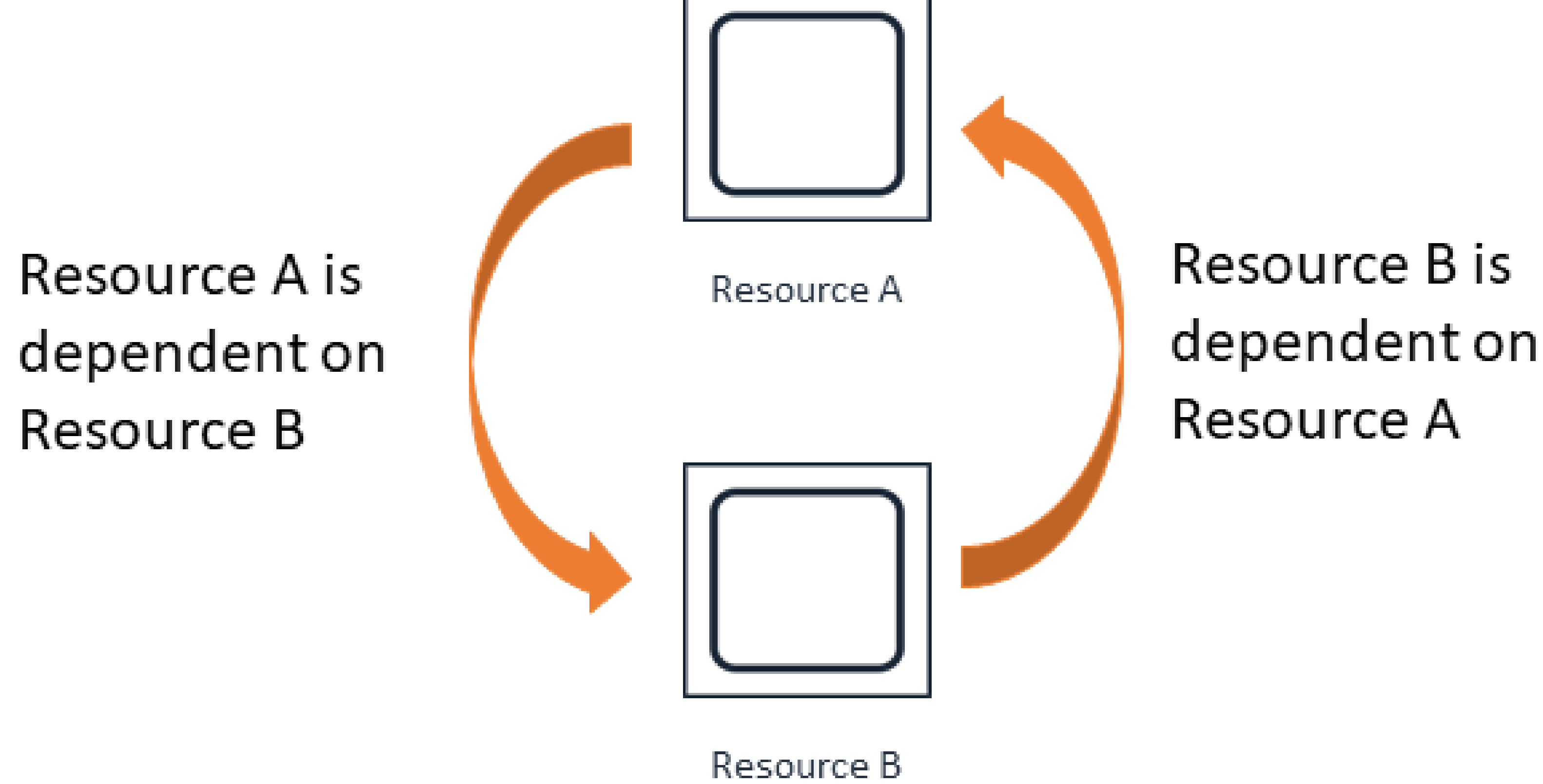
3.10 Concurrent Servers

- Most servers are concurrent
- 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
- After handling one client the thread terminates, but the main thread keeps the server alive
  - the main thread waits for another request to arrive

3.11 Circular Dependencies Among Servers

- A server for one service can act as a client for another
  - For example, before it can fill in a web page, a web server may need to become a client of a database
  - A server may also become the client of a security service (e.g., to verify that a client is allowed to access the service).
- Programmers must be careful to avoid circular dependencies among servers
  - since chain of requests can continue indefinitely until all three servers exhaust resources

Dependency



3.12 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

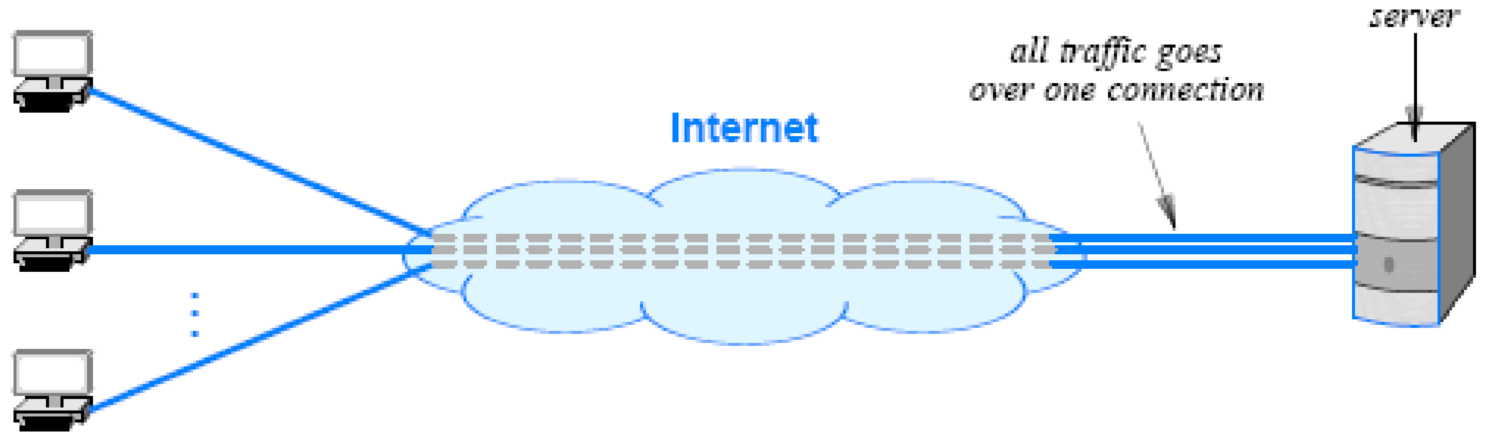
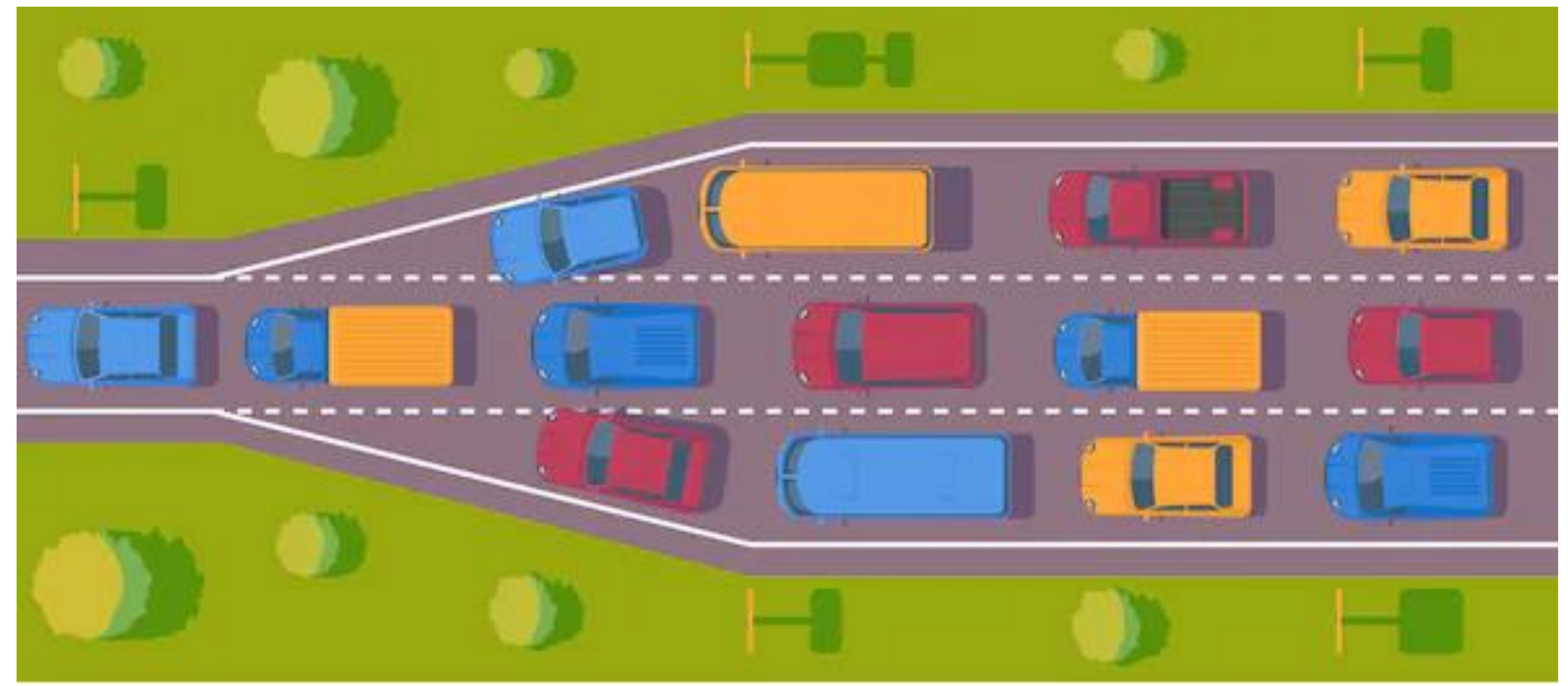


Figure 3.5 The traffic bottleneck in a design that uses a single server.

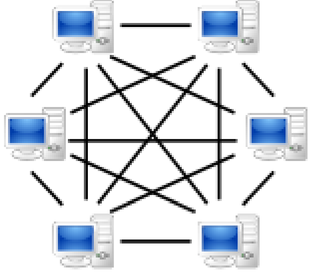
Speed Bottleneck.



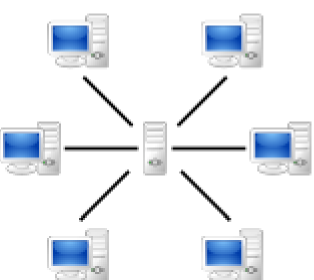
3.12 Peer-to-Peer Interactions

- 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

- Figure 3.6 illustrates the architecture



Peer-to-peer



Client/Server

3.12 Peer-to-Peer Interactions

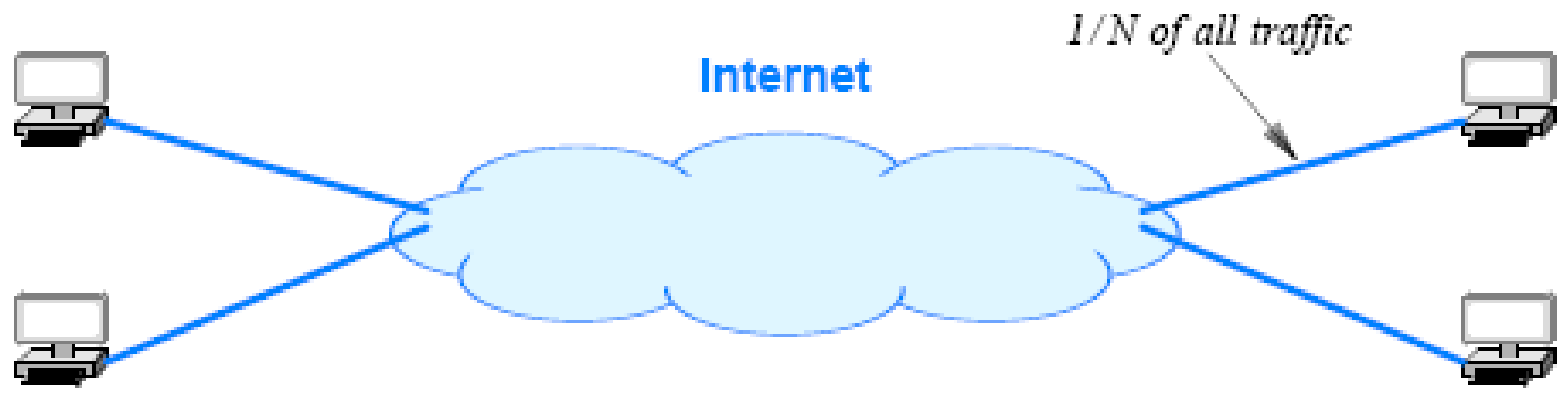


Figure 3.6 Interaction in a peer-to-peer system.

3.13 Network Programming and the Socket API

- Applications uses a communication interface 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
- The socket API is available for many OS
  - such as Microsoft's Windows systems
  - as well as various UNIX/Linux systems

3.14 Sockets, Descriptors, and Network I/O

- The socket API was originally developed as part of the UNIX OS, so the socket API is integrated with I/O
- When an application creates a socket to use for Internet , the OS returns a small integer descriptor identifying a 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)
- In many OS, socket descriptors are integrated with other I/O descriptors

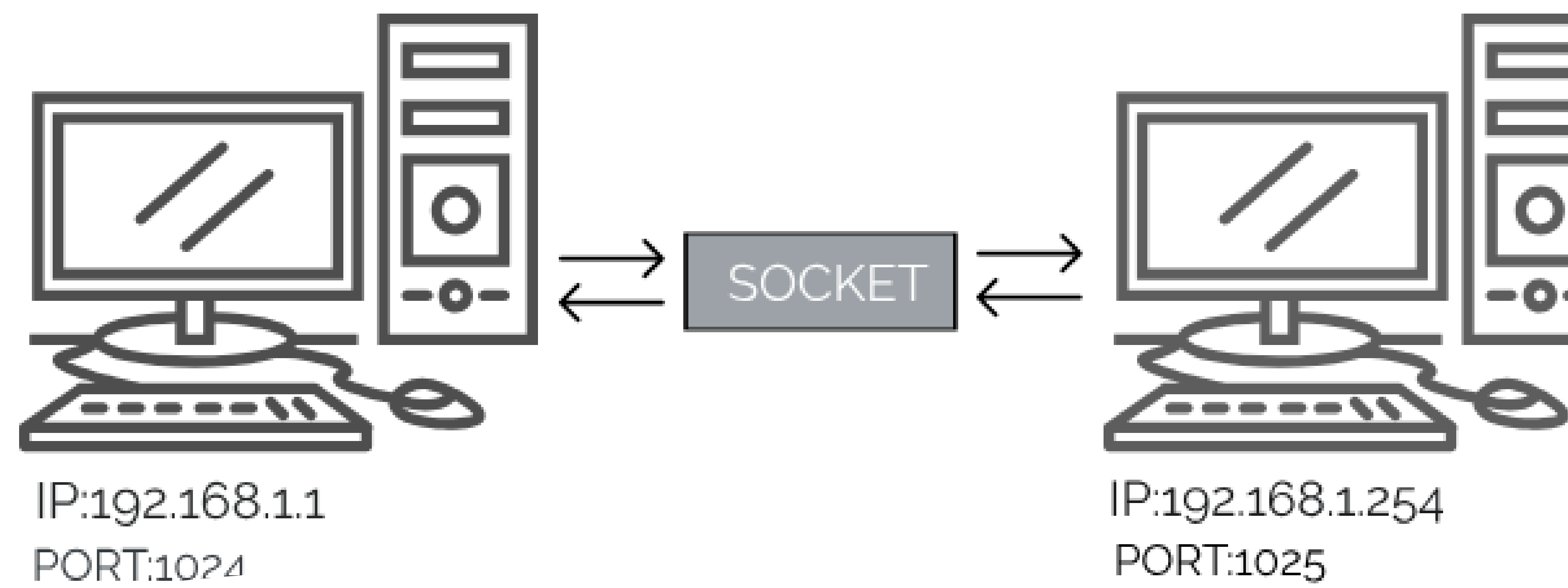
3.15 Parameters and the Socket API

- Socket programming differs from conventional I/O
- 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
  - an application creates a socket, and then invokes functions for details
- The advantage of the socket approach is that most functions have three or fewer parameters
- The disadvantage is that a programmer must remember to call multiple functions when using sockets

- Figure 3.7 summarizes key functions in the socket API

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



3.16 Socket Calls in a Client and Server

- Figure 3.8 illustrates the sequence of socket calls made by a typical client and server that use a stream connection
  - The client sends data first and the server waits to receive data
- In practice, some applications arrange for the server to send first (i.e., *send* and *recv* are called in the reverse order)

CLIENT SIDE

socket
connect
send
recv
close

SERVER SIDE

socket
bind
listen
accept
recv
send
close

Figure 3.8 Illustration of the sequence of socket functions called by a client and server using the stream paradigm.