

More Introduction

Outline

- Computer Networks Overview
- Statistical Multiplexing
- Inter-Process Communication
- Network Architecture
- Performance Metrics
- Implementation Issues

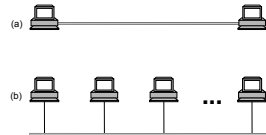
Spring 2009

CSE 30264

26

Building Blocks for Data Communications

- Nodes: PC, special-purpose hardware...
 - hosts
 - switches
- Links: coax cable, optical fiber...
 - point-to-point
 - multiple access



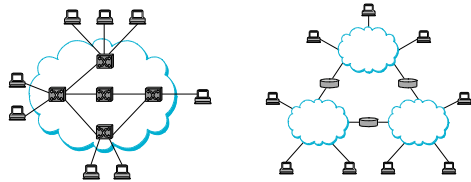
Spring 2009

CSE 30264

27

Switched Networks

- A network can be defined recursively as...
 - two or more nodes connected by a link, or
 - two or more networks connected by a node



Spring 2009

CSE 30264

28

Strategies

- Circuit switching: carry bit streams
 - original telephone network
- Packet switching: store-and-forward messages
 - Internet

Spring 2009

CSE 30264

29

Addressing and Routing

- Address: byte-string that identifies a node
 - usually unique
- Routing: process of forwarding messages to the destination node based on its address
- Types of addresses
 - unicast: node-specific
 - broadcast: all nodes on the network
 - multicast: some subset of nodes on the network

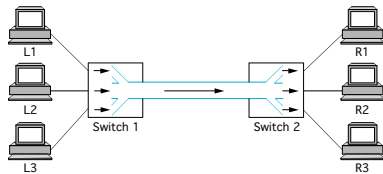
Spring 2009

CSE 30264

30

Multiplexing

- Time-Division Multiplexing (TDM)
- Frequency-Division Multiplexing (FDM)



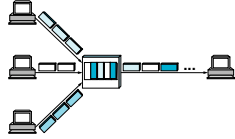
Spring 2009

CSE 30264

31

Statistical Multiplexing

- On-demand time-division
- Schedule link on a *per-packet* basis
- Packets from different sources interleaved on link
- Buffer packets that are *contending* for the link
- Buffer (queue) overflow is called *congestion*



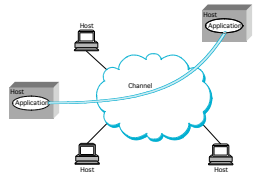
Spring 2009

CSE 30264

32

Inter-Process Communication

- Turn host-to-host connectivity into process-to-process communication.
- Fill gap between what applications expect and what the underlying technology provides.



Spring 2009

CSE 30264

33

IPC Abstractions

- Request/Reply
 - distributed file systems
 - digital libraries (web)
- Stream-Based
 - video: sequence of frames
 - 1/4 NTSC = 352x240 pixels
 - $(352 \times 240 \times 24) / 8 = 247.5 \text{KB}$
 - 30 fps = 7500KBps = 60Mbps
 - video applications
 - on-demand video
 - video conferencing

Spring 2009

CSE 30264

34

What Goes Wrong in the Network?

- Bit-level errors (electrical interference)
- Packet-level errors (congestion)
- Link and node failures

- Packets are delayed
- Packets are delivered out-of-order
- Third parties eavesdrop

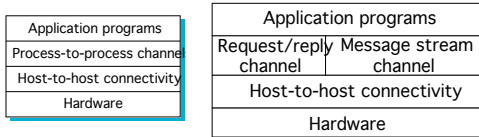
Spring 2009

CSE 30264

35

Layering

- Use abstractions to hide complexity
- Abstraction naturally lead to layering
- Alternative abstractions at each layer



Spring 2009

CSE 30264

36

Protocols

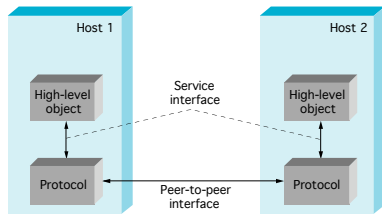
- Building blocks of a network architecture
- Each protocol object has two different interfaces
 - *service interface*: operations on this protocol
 - *peer-to-peer interface*: messages exchanged with peer
- Term “protocol” is overloaded
 - specification of peer-to-peer interface
 - module that implements this interface

Spring 2009

CSE 30264

37

Interfaces



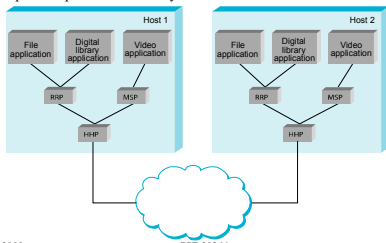
Spring 2009

CSE 30264

38

Protocol Machinery

- Protocol Graph
 - most peer-to-peer communication is indirect
 - peer-to-peer is direct only at hardware level



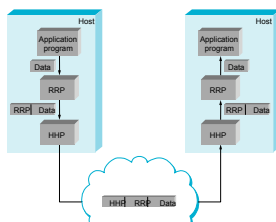
Spring 2009

CSE 30264

39

Machinery (cont.)

- Encapsulation (header/body)
- Multiplexing and Demultiplexing (demux key)



Spring 2009

CSE 30264

40

How Does Data Pass Through Layers?

- Protocol implementations follow the layering model
 - by passing the output from a protocol in one layer to the input of a protocol in the next
- To achieve **efficiency**
 - rather than copy an entire packet, a pair of protocols in adjacent layers pass a **pointer** to the packet
- <http://wps.prenhall.com/wps/media/objects/5959/6102815/layers.html>
 - each computer contains a set of layered protocols
 - when an application sends data
 - it is placed in a packet, and the packet passes down through each layer of protocols
 - once it has passed through all layers of protocols on the sending computer
 - the packet leaves the computer and is transmitted across the physical network
 - when it reaches the receiving computer
 - the packet passes up through the layers of protocols
 - if the application on the receiver sends a **response**, the process is **reversed**

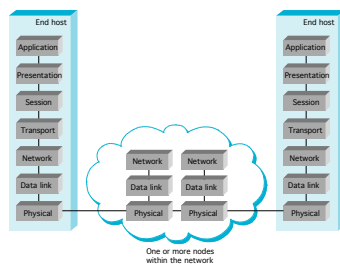
Spring 2009 CSE 30264 41

Headers

- Each layer of protocol software performs **computations**
 - that insure the messages arrive as expected
- To perform such computation, protocol software on the two machines must exchange information
 - each layer on the sender **prepends** extra information onto the packet
 - the corresponding protocol layer on the receiver removes and uses the extra information
- Additional information added by a protocol is known as a **header**
- Headers are added by protocol software on the sending computer
 - that is, the transport layer prepends a header, and then the Internet layer prepends a header, and so on
- In practice headers are not of uniform size
- A physical layer header is optional

Spring 2009 CSE 30264 42

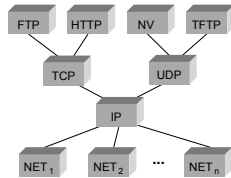
OSI Architecture



Spring 2009 CSE 30264 43

Internet Architecture

- Defined by Internet Engineering Task Force (IETF)
- Hourglass Design
- Application vs Application Protocol (FTP, HTTP)



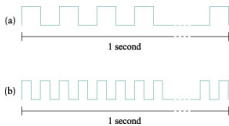
Spring 2009

CSE 30264

44

Performance Metrics

- **Bandwidth (throughput)**
 - data transmitted per time unit
 - link versus end-to-end
 - notation
 - KB = 2¹⁰ bytes
 - Mbps = 10⁶ bits per second
- **Latency (delay)**
 - time to send message from point A to point B
 - one-way versus round-trip time (RTT)
 - components
 - Latency = Propagation + Transmit + Queue
 - Propagation = Distance / c
 - Transmit = Size / Bandwidth
- **Relative importance**
 - 1-byte: 1ms vs 100ms dominates 1Mbps vs 100Mbps
 - 25MB: 1Mbps vs 100Mbps dominates 1ms vs 100ms



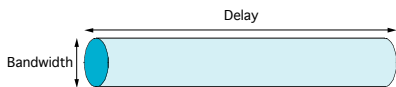
Spring 2009

CSE 30264

45

Delay x Bandwidth Product

- Amount of data “in flight” or “in the pipe”
- Usually relative to RTT
- Example: 100ms x 45Mbps = 560KB



Spring 2009

CSE 30264

46

Client-Server Paradigm

Outline

- Client-Server Paradigm
- Types of Server Implementations

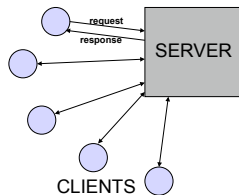
Spring 2009

CSE 30264

47

What is a Server?

- Provides specific kind of **service**
- Examples:
 - web
 - email
 - database
 - printer
 - ftp
 - file
 - name
 - ...



Spring 2009

CSE 30264

48

Client-Server Model

- Relationship between two computers, where one (server) provides a service and responds to service requests and the other (client) issues such service requests
- Specific form of interaction known:
 - a **server** starts first and **awaits** contact
 - a **client** starts second and **initiates** the connection

Spring 2009

CSE 30264

49

Client-Server Model

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.

Spring 2009

CSE 30264

50

Characteristics

- 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

Spring 2009

CSE 30264

51

Characteristics

- 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
 - may require powerful hardware and a sophisticated **operating system (OS)**

Spring 2009

CSE 30264

52

Terminology

- 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 Illustration of a client and server.

Spring 2009

CSE 30264

53

Requests and Responses

- 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)

Spring 2009

CSE 30264

54

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

Spring 2009

CSE 30264

55

Multiple Clients

- A computer can run:
 - a single client
 - multiple copies of a client that contact a given server
 - multiple clients that each contact a particular server
- 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

Spring 2009

CSE 30264

56

Server Identification

- 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)** is used to translate names into addresses
 - thus, a user specifies a name such as *www.cisco.com* rather than an integer address

Spring 2009

CSE 30264

57

Identification

- 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**)
 - 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 (**demultiplexing**)

Spring 2009

CSE 30264

58

Summary


- 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.

Spring 2009

CSE 30264

59

Concurrent Servers

- Although a serial approach works in a few trivial cases, 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

Spring 2009

CSE 30264

60

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

Spring 2009

CSE 30264

61

Pitfall: Circular Dependencies

- In practice, the distinction **blurs** because 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
 - for example, consider what can happen if a server for service **X** becomes a client of service **Y**, which becomes a client of service **Z**, which becomes a client of **X**
 - the chain of requests can continue indefinitely until all three servers exhaust resources
- The potential for **circularity** is especially high when services are designed **independently**
 - because no single programmer controls all servers
