Bits

www.penny-arcade.com

Homework 1

- Login to one of the CRC computers
  - netscale01.cse.nd.edu
  - netscale02.cse.nd.edu, …
  - 1-4, 9-16
- Use SSH
  - First floor Fitzpatrick computer lab
    - ssh striegel@netscale01.cse.nd.edu
  - Download F-Secure SSH from OIT
    - oit.nd.edu
More Introduction

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

Building Blocks

- **Nodes**
  - Send / receive communications
  - Examples
    - **Host**
      - Computer, smart phone
    - **Router**
      - Speaks IP
    - **Switch**
      - Speaks Layer 2
      - Ethernet, Wireless (802.11)

Abstract representation

Building Blocks

- **Links**
  - Connect the nodes
  - Hosts can have multiple *interfaces*
  - Types
    - **Point-to-point**
      - Host to host
      - Fiber
    - **Multiple access**
      - Multiple hosts share
Network

- Two or more nodes connected by a link

Multiple access

Point-to-point

Network of Networks

Network of Networks
Strategies

- Circuit switching: carry bit streams
  - Original telephone network
  - **PSTN** – Public Switched Telephone Network

- Packet switching: store-and-forward messages
  - Ethernet **LAN** (Local Area Network)
  - Internet

Addressing

- **Address**
  - Byte string that identifies a node
  - Usually unique but not always
  - Examples
    - Ethernet address -> 00:FE:AB:23:12:34
    - IP address -> 129.74.250.100
  - Most protocols involve two addresses
    - Source
    - Destination
Routing

- **Routing**
  - Process of forwarding messages to the destination node based on its address
  - Involve multiple nodes, multiple links

- **Address Types**
  - **Unicast**: node-specific
    - Go to cnn.com
  - **Broadcast**: all nodes on the network
    - TV signal
  - **Multicast**: subset of nodes
    - Contact all students in CSE 30264

Multiplexing

- What happens when we don’t need a dedicated link?
  - **Multiplex** or aggregate multiple users together
- When do I get the link?
  - Fixed or dynamic

- **Time-Division Multiplexing (TDM)**
- **Frequency-Division Multiplexing (FDM)**
- **Wavelength Division Multiplexing (WDM)**
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*

![Diagram showing statistical multiplexing with FIFO and FCFS](image)

Inter-Process Communication

- Do useful stuff
  - Move my data
- Fill gap between what applications expect and what the underlying technology provides.

![Diagram showing inter-process communication](image)

IPC Abstractions

- **Request/Reply**
  - Distributed file systems
  - Digital libraries (web)

- **Stream-Based**
  - Video: sequence of frames
    - 1/4 NTSC = 352x240 pixels
    - (352 x 240 x 24) / 8 = 270.72KB
    - 30 fps = 7500KBps = 60Mbps
  - Video applications
    - On-demand video
    - Video conferencing
Layering

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

Protocols

- **Rules** of a network architecture
- Each protocol has two different interfaces
  - **Service**: Operations on this protocol
    - I need a socket with a timeout
  - **Peer-to-peer**: Messages exchanged with peer(s)
    - Send this message to the other host
- Term "protocol" is overloaded
  - Specification of peer-to-peer interface
  - Module that implements this interface

Interfaces
Protocols

• Two primary types
  – RRP – Request / Reply Protocol
    • Discrete
    • Telegram / Datagram
    • Very aware of packetized nature of network
  – MSP – Message Stream Protocol
    • On-going connection
    • Pushing data back / forth

Protocols vs. Layers

• Layering Model
  – Wrap, pass down
  – Stack -> network stack
• To achieve efficiency
  – Use pointers (C)
  – Network stack -> kernel level
• What does layering mean?
  – Downward awareness
  – Ignore context from above
  • Block of data vs. HTTP GET request
  – Higher layers
    • Looser structure, more complex
  – Lower layers
    • Rigid structure, KISS, Gig E -> 12 microsec / packet (full size)

Protocol Content

• Header
  – Prepend as it goes down the stack
  – Information added by a particular protocol
    • Address, type, length, options
• Processing Instructions
  – Identifier of upper layer type
    • Ethernet -> Type / Len field (0x0800 = IP)
    • IP -> Protocol field (6 = TCP, 17 = UDP)
• Integrity
  – Did the message arrive correctly?
    • Checksum, hash
Machinery

- Encapsulation (header/body)
  - Wrap and pass down

<table>
<thead>
<tr>
<th>HHP – Host to Host Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP</td>
</tr>
<tr>
<td>IP</td>
</tr>
<tr>
<td>802.3</td>
</tr>
</tbody>
</table>

OSI Architecture

Internet Architecture

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

Performance Metrics

- Bandwidth (throughput)
  - Data transmitted per time unit
  - Link, end-to-end
  - Notation
    - KB = $2^{10}$ bytes
    - Mbps = $10^6$ bits per second

- Latency (delay)
  - Time to send message from point A to point B
  - One-way, round-trip time (RTT)
  - Components
    - Propagation + Transmit + Queue
    - Distance / Speed
    - Size / Bandwidth

- Relative importance
  - Timeliness requirements - elastic vs. inelastic
  - Size of data
  - Jitter
Delay x Bandwidth Product (DBP)

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

Client-Server Paradigm

Outline
- Client-Server Paradigm
- Types of Server Implementations

What is a Server?

- Provides specific kind of service
- Examples:
  - Web
  - Email
  - Database
  - Printer
  - Ftp
  - File
  - Name
  - ...

[Diagram of Client-Server Paradigm]
Client-Server Model

- Two computers
  - One provides a service (server)
  - One issues service requests (client)
- Specific form of interaction known:
  - a server starts first and awaits contact
  - a client starts second and initiates the connection

Client 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

Server 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)
Client-Server Model

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

Figure 5.2 A summary of the client-server model.

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

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)
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

- 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 windows open simultaneously running three applications:
  - one that retrieves and displays email
  - another that connects to a chat service
  - and a third running a web browser

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?
  - Unique 32-bit identifier known as an Internet Protocol address (IP address)
  - Client must specify the server’s IP address
  - 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
Identification

- Identifying a service?
  - 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)

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

![Diagram](Image)

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

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
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

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 these servers exhaust resources
- The potential for circularity is especially high when services are designed independently
  - because no single programmer controls all servers

Wrap Up

- Review
  - Nodes, links
  - Protocol vs. layers
  - Client / server
  - Performance metrics
  - Concurrency