CSE 124
Week 4: Discussion Session
Feb 8, Friday
Outline (Review)

- Routing; CIDR
- Layering
- Sockets
- HTTP
- Caching and Consistency
- RPC
- Concurrency and Parallelism
- Virtualization
- DNS
Routing

- A single network consists of many LANs.

- Dijkstra’s shortest path algorithm- Routing within a single network. We get the IP forwarding table!

- After you’ve reached a subnet, you need to reach the host. Use STP (Spanning Tree Protocol)
CIDR (Classless Inter-Domain Routing)

<table>
<thead>
<tr>
<th>Net/MaskLength</th>
<th>NextHop</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.5.1.0/24</td>
<td>A</td>
</tr>
<tr>
<td>10.5.2.0/24</td>
<td>B</td>
</tr>
<tr>
<td>10.5.1.210/32</td>
<td>C</td>
</tr>
<tr>
<td>10.0.0.0/8</td>
<td>D</td>
</tr>
<tr>
<td>Default (0/0)</td>
<td>E</td>
</tr>
</tbody>
</table>

- 10.210.3.14:
- 10.5.1.210:
- 128.51.10.9:
CIDR (Classless Inter-Domain Routing)

<table>
<thead>
<tr>
<th>Net/MaskLength</th>
<th>NextHop</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.5.1.0/24</td>
<td>A</td>
</tr>
<tr>
<td>10.5.2.0/24</td>
<td>B</td>
</tr>
<tr>
<td>10.5.1.210/32</td>
<td>C</td>
</tr>
<tr>
<td>10.0.0.0/8</td>
<td>D</td>
</tr>
<tr>
<td>Default (0/0)</td>
<td>E</td>
</tr>
</tbody>
</table>

- 10.210.3.14: **D**
- 10.5.1.210: **C**
- 128.51.10.9: **E**
Layering

**LAYERING: A MODULAR APPROACH TO COMM**

- Sub-divide responsibilities
  - Each layer relies on services from layer below
  - Each layer exports services to layer above

- Interface between layers defines interaction
  - Hides implementation details (encapsulation)
  - Layers can change without disturbing other layers (modularity)

- Interface among peers in a layer is a *protocol*
  - If peers speak same protocol, they can interoperate
Application Layer: To a specific application on the same host
Transport Layer: Process-to-process delivery (host-to-host)
Network Layer: Node-to-node delivery
Advantages/Disadvantages Of Layering

- **Advantages:**
  - Modularity
  - Encapsulation

- **Disadvantages:**
  - Space-overhead
  - Time-overhead
## Sockets

### WHAT IS A SOCKET?

- **What is a socket?**
  - The point where a local application process attaches to the network
  - An interface between an application and the network
  - An application creates the socket
- **The interface defines operations for**
  - Creating a socket
  - Attaching a socket to the network
  - Sending and receiving messages through the socket
  - Closing the socket
HTTP

HTTP OVERVIEW

- HTTP is a text oriented protocol.
- HTTP is a request/response protocol
- Requests and responses both look like:
  
  START_LINE <CRLF>
  
  MESSAGE_HEADER <CRLF>
  
  <CRLF>
  
  MESSAGE_BODY <CRLF>

- The first line (START LINE) indicates whether this is a request message or a response message.
Example

Assuming that the following /htdocs directory tree is the web server’s document root, and that \( CR \) represents a carriage-return and \( LF \) represents a line-feed:

/htdocs
/htdocs/index.html
/htdocs/images/kitten.jpg
/htdocs/images/dog.png

(2 points) GET / HTTP/1.1\nUser-Agent: TritonBrowser/1.0\n
(2 points) GET / HTTP/1.1\nHost: www.cs.ucsd.edu\nUser-Agent: TritonBrowser/1.0
Example

Assuming that the following /htdocs directory tree is the web server’s document root, and that \( CR \) represents a carriage-return and \( LF \) represents a line-feed:

```
/htdocs
/htdocs/index.html
/htdocs/images/kitten.jpg
/htdocs/images/dog.png
```

(2 points) GET / HTTP/1.1<CR><LF>
User-Agent: TritonBrowser/1.0<CR><LF>
<CR><LF>

400 Client Error

(2 points) GET / HTTP/1.1<CR><LF>
Host: www.cs.ucsd.edu<CR><LF>
User-Agent: TritonBrowser/1.0<CR><LF>
<CR><LF>

200 OK
Caching and Consistency

CACHING OPTIONS

- Centralized control: Record status of clients (which files open for reading/writing, what cached, ...)
- Read-ahead: Pre-fetch blocks before needed
- Write-through: All writes sent to server
- Write-behind: Writes locally buffered, send as batch
- Consistency challenges:
  - When client writes, how do others caching data get updated? (Callbacks, ...)
  - Two clients concurrently write? (Locking, overwrite, ...)
LOCKS

• A client can request a lock over a file / byte range
  • Advisory: Well-behaved clients comply
  • Mandatory: Server-enforced
• Client performs writes, then unlocks
• Problem: What if the client crashes?
  • Solution: Keep-alive timer: Recover lock on timeout
    • Problem: what if client alive but network route failed?
      • Client thinks it has lock, server gives lock to other: “Split brain”
LEASES

- Client obtains **lease** on file for read or write
  - “A lease is a ticket permitting an activity; the lease is valid until some expiration time.”
- **Read lease** allows client to cache clean data
  - *Guarantee*: no other client is modifying file
- **Write lease** allows safe delayed writes
  - Client can locally modify than batch writes to server
  - *Guarantee*: no other client has file cached
• Before lease expires, client must renew lease
• Client fails while holding a lease?
  • Server waits until the lease expires, then unilaterally reclaims
  • If client fails during eviction, server waits then reclaims
• Server fails while leases outstanding? On recovery,
  • Wait lease period + clock skew before issuing new leases
  • Absorb renewal requests and/or writes for evicted leases
RPC

**WHY RPC?**

- The typical programmer is trained to write single-threaded code that runs in **one place**
- **Goal:** Easy-to-program network communication that makes client-server communication **transparent**
  - Retains the “feel” of writing centralized code
    - Programmer needn’t think about the network
RPC ISSUES

- Heterogeneity
  - Client needs to **rendezvous** with the server
  - Server must **dispatch** to the required function
    - What if server is **different** type of machine?
- Failure
  - What if messages get **dropped**?
  - What if client, server, or network **fails**?
- Performance
  - Procedure call takes $\approx 10 \text{ cycles } \approx 3 \text{ ns}$
  - RPC in a data center takes $\approx 10 \mu s \ (10^3 \times \text{ slower})$
    - In the wide area, typically $10^6 \times \text{ slower}$
Some more problems

• Difference in data representations
• Differences in programming support

• Solution?
  • Interface Description Language
Schemes for handling failures

<table>
<thead>
<tr>
<th>Retry Request</th>
<th>Duplicate Filtering</th>
<th>Retransmit Response</th>
<th>RPC Call Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>NA</td>
<td>NA</td>
<td>Maybe</td>
</tr>
<tr>
<td>Yes</td>
<td>No</td>
<td>Re-execute Procedure</td>
<td>At-least once</td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>Retransmit reply</td>
<td>At-most once</td>
</tr>
</tbody>
</table>
Concurrency & Parallelism

- Concurrency: Progress of multiple applications overlap in time. Time sharing of the resources is involved.

- Parallelism: Progress of multiple applications occur at the same time. We magically have multiple resources. Each such resource take up one application.
Virtualization

• Creating a virtual (rather than actual) version of something.

• Why?
  Hardware changes faster than software
  Ease of portability and code migration
  Isolation of failing or attacked components

Four types of interfaces at three different levels

1. **Instruction set architecture**: the set of machine instructions, with two subsets:
   - Privileged instructions: allowed to be executed only by the operating system.
   - General instructions: can be executed by any program.

2. **System calls** as offered by an operating system.

3. **Library calls**, known as an **application programming interface (API)**
DNS

- Domain Name System (DNS) host name: Variable length, full alphabet of characters. Human Friendly

- IP Address: Fixed length, decimal number. Router Friendly. Hierarchical address space, related to host location

- Goals of DNS: Scalability (decentralized maintenance) Robustness Global scope Good Performance
DNS in operation

- Recursive: DNS server will fetch the answer back to you. During this process, the DNS server might also query other DNS servers on the internet on your behalf, for the answer.

- Iterative: Will not go and fetch the complete answer for your query, but will give back a referral to other DNS servers, which might have the answer.

Why would we need it? DNS server might need to know the source IP address of the client to get the closest server.
Questions?