Today’s Lecture

- Project 4
  - Overview / Discussion
- Applications – Part 1
  - DNS
  - SMTP
  - HTTP
  - P2P

Project 4

- Implementation
  - Multi-threaded server
  - Performance will count
- Experimentation
  - Latency
  - Loss

Due: Thursday, April 29th, 5 PM
Premise

- Spaceship racing game in alpha development
  - C# / WPF / Windows
  - Up to 4 players
  - Test network functionality
- Your task
  - Write a multi-threaded server that allows clients to connect and relays messages back and forth
- Twist
  - The protocol is changing
    - Need to make your server agnostic with respect to message content

Operation

- Client starts up
- Client sets its player name to a unique value
  - Responsibility of client, not server
- Client connects to the server
  - TCP, port defined in the client
- Client periodically sends data
  - Send status / information to server
  - Server relays that information to all other clients

Client App

- Provided the alpha version
  - Core functionality
- Will also be provided a beta version (next week)
  - Additional polish ➔ different message contents
- Near-final version
  - Different set of messages
  - Only for evaluation of your code
Your Task - Server

- Write a multi-threaded server
  - Listen for new TCP connections on a particular port
  - Receive messages / relay to all other clients
    - Receive and forward
    - No inspection or modification
- What is different
  - Server needs to be robust
    - Client quits, error detection, etc.
  - Thread safety
    - Potential for clients to exit, arrive while receiving / sending messages
    - Mutexes
    - Performance
    - Speedy or else the clients will lag

Notes - Implementation

- Console input
  - Queries / status
- Thread – listen / accept
- Thread – each client
- Global object
  - List of sockets / etc.
  - Guard on write
  - Guard on modifying the list

Experimentation

- Human experimenting
  - Alpha version – raw racing field
  - Beta version – collision detection / etc.
- Tinker with delay / loss
  - Slider bar ➔ inject loss
    - Fails to report a new status
  - Slider bar ➔ inject delay
    - Slightly queues the status report
    - Where does it start to seem off?
Miscellaneous

Outline
Domain Name System
Peer-to-Peer Networks

Name Service

- Names versus addresses
- Location transparent versus location-dependent
- Flat versus hierarchical
- Resolution mechanism
- Name server
- DNS: domain name system

Examples

- Hosts
  - cheltenham.cs.princeton.edu → 192.12.69.17
  - 192.12.69.17 → 80:23:A8:33:5B:9F
- Files
  - /usr/lilp/tmp/foo → fileid
Examples (cont)

- Mailboxes

  ![Diagram of mailboxes]

Domain Naming System

- Hierarchy

  ![Diagram of hierarchy]

- Name

  wizard.cse.nd.edu

Name Servers

- Partition hierarchy into zones

  ![Diagram of name servers]

- Each zone implemented by two or more name servers
Resource Records

- Each name server maintains a collection of resource records (Name, Value, Type, Class, TTL)
- Name/Value: not necessarily host names to IP addresses
- Type
  - A: IP addresses
  - NS: value gives domain name for host running name server that knows how to resolve names within specified domain.
  - CNAME: value gives canonical name for a host; used to define aliases.
  - MX: value gives domain name for host running mail server that accepts messages for specified domain.
- Class: allows other entities to define types
- TTL: how long the resource record is valid

Root Server

(princeton.edu, cit.princeton.edu, NS, IN)
(cit.princeton.edu, 128.196.128.233, A, IN)

(cisco.com, thumper.cisco.com, NS, IN)
(thumper.cisco.com, 128.96.32.20, A, IN)

Princeton Server

(cs.princeton.edu, optima.cs.princeton.edu, NS, IN)
(optima.cs.princeton.edu, 192.12.69.5, A, IN)

(we.princeton.edu, helios.we.princeton.edu, NS, IN)
(helios.we.princeton.edu, 128.196.28.166, A, IN)

(jupiter.physics.princeton.edu, 128.196.4.1, A, IN)
(saturn.physics.princeton.edu, 128.196.4.2, A, IN)
(mars.physics.princeton.edu, 128.196.4.3, A, IN)
(venus.physics.princeton.edu, 128.196.4.4, A, IN)
Name Resolution

- Strategy
  - Local server
    - need to know root at only one place (not each host)
    - site-wide cache

Electronic Mail

- RFC 822: header and body
- MIME: Multi-purpose Internet Mail Extensions

```plaintext
From: Alice Smith
       Alice@cisco.com
To: Bob@cse.nd.edu
Subject: look at the attached image!
Date: Mon, 07 Sep 1998 19:45:19 -0400

Bob,
here's the jpeg image I promised.

-- Alice

---
```
SMTP

- Mail reader, mail daemon, mail gateway
- SMTP messages: HELO, MAIL, RCPT, DATA, QUIT; server responds with code.

World Wide Web

- URL: uniform resource locator
  http://www.cse.nd.edu
- HTTP:
  START_LINE <CRLF>
  MESSAGE_HEADER <CRLF>
  <CRLF>
  MESSAGE_BODY <CRLF>

HTTP

- Request:
  - GET: fetch specified web page
  - HEAD: fetch status information about specified page
  GET http://www.cse.nd.edu/index.html HTTP/1.1
- Response:
  - HTTP/1.1 202 Accepted
  - HTTP/1.1 404 Not Found
  - HTTP/1.1 301 Moved Permanently
    Location: http://www.nd.edu/cs/index.html
HTTP

- **HTTP 1.0**: separate TCP connection for each request (each data item).
- **HTTP 1.1**: persistent connections
- **Caching**:
  - client: faster retrieval of web pages
  - server: reduced load
  - location: client, sitewide cache, ISP, etc.
  - EXPIRES header field (provided by server)
  - IF-MODIFIED-SINCE header field (issued by cache)

SNMP

- Request/reply protocol (on top of UDP)
- 2 main operations:
  - GET: retrieve state info from hosts
  - SET: set new state on host
- Relies on Management Information Base (MIB)
  - system: uptime, name, …
  - interfaces: physical address, packets sent/received, …
  - address translation: ARP (contents of table)
  - IP: routing table, number of forwarded datagrams, reassembly statistics, dropped packets, …
  - TCP: number of passive and active opens, resets, timeouts, …
  - UDP: number of packets sent/received, …

P2P

- **Overview**:
  - centralized database: Napster
  - query flooding: Gnutella
  - intelligent query flooding: KaZaA
  - swarming: BitTorrent
  - unstructured overlay routing: Freenet
  - structured overlay routing: Distributed Hash Tables
Napster

• Centralized Database:
  – **Join**: on startup, client contacts central server
  – **Publish**: reports list of files to central server
  – **Search**: query the server \(\Rightarrow\) return someone that stores the requested file
  – **Fetch**: get the file directly from peer

Gnutella

• **Query Flooding**:
  – **Join**: on startup, client contacts a few other nodes; these become its “neighbors”
  – **Publish**: no need
  – **Search**: ask neighbors, who ask their neighbors, and so on... when/if found, reply to sender.
  – **Fetch**: get the file directly from peer

KaZaA (Kazaa)

• In 2001, Kazaa created by Dutch company KaZaA BV.
• Single network called FastTrack used by other clients as well: Morpheus, giFT, etc.
• Eventually protocol changed so other clients could no longer talk to it.
• 2004: 2nd most popular file sharing network, 1-5million at any given time, about 1000 downloads per minute. (June 2004, average 2.7 million users, compare to BitTorrent: 8 million)
KaZaA

- "Smart" Query Flooding:
  - **Join**: on startup, client contacts a "supernode"... may at some point become one itself
  - **Publish**: send list of files to supernode
  - **Search**: send query to supernode, supernodes flood query amongst themselves.
  - **Fetch**: get the file directly from peer(s); can fetch simultaneously from multiple peers

---

KaZaA

"Super Nodes"

---

KaZaA: File Insert

```sql
insert(X, 123.2.21.23) ...
```

Publish

I have X!

123.2.21.23
KaZaA: File Search

- More than one node may have requested file...
- How to tell?
  - must be able to distinguish identical files
  - not necessarily same filename
  - same filename not necessarily same file...
- Use Hash of file
  - KaZaA uses UUHash: fast, but not secure
  - alternatives: MD5, SHA-1
- How to fetch?
  - get bytes [0,1000] from A, [1001,2000] from B

KaZaA: Fetching

- More than one node may have requested file...
- How to tell?
  - must be able to distinguish identical files
  - not necessarily same filename
  - same filename not necessarily same file...
- Use Hash of file
  - KaZaA uses UUHash: fast, but not secure
  - alternatives: MD5, SHA-1
- How to fetch?
  - get bytes [0,1000] from A, [1001,2000] from B

KaZaA

- Pros:
  - tries to take into account node heterogeneity:
    - bandwidth
    - host computational resources
  - rumored to take into account network locality
- Cons:
  - mechanisms easy to circumvent
  - still no real guarantees on search scope or search time
BitTorrent

• In 2002, B. Cohen debuted BitTorrent
• Key motivation:
  – popularity exhibits temporal locality (flash crowds)
  – e.g., Slashdot effect, CNN on 911, new movie/game release
• Focused on efficient Fetching, not Searching:
  – distribute the same file to all peers
  – files split up in pieces (typically 250kBytes)
  – single publisher, multiple downloaders
  – each downloader becomes a publisher (while still downloading)
• Has some “real” publishers:
  – Blizzard Entertainment using it to distribute the beta of their new games

BitTorrent

• Swarming:
  – Join: contact centralized “tracker” server, get a list of peers.
  – Publish: run a tracker server.
  – Search: out-of-band, e.g., use Google to find a tracker for the file you want.
  – Fetch: download chunks of the file from your peers.
    Upload chunks you have to them.
BitTorrent: Fetch

BitTorrent: Sharing Strategy

- Employ “Tit-for-tat” sharing strategy
  - “I’ll share with you if you share with me”
  - be optimistic: occasionally let freeloaders download
    - otherwise no one would ever start!
    - also allows you to discover better peers to download from when they reciprocate

- Approximates Pareto Efficiency
  - game theory: “No change can make anyone better off without making others worse off”

BitTorrent

- Pros:
  - works reasonably well in practice
  - gives peers incentive to share resources; avoids freeloaders

- Cons:
  - central tracker server needed to bootstrap swarm
Freenet

- In 1999, I. Clarke started the Freenet project
- Basic idea:
  - employ Internet-like routing on the overlay network to publish and locate files
- Additional goals:
  - provide anonymity and security
  - make censorship difficult

FreeNet

- Routed Queries:
  - Join: on startup, client contacts a few other nodes it knows about; gets a unique node id
  - Publish: route file contents toward the file id. File is stored at node with id closest to file id
  - Search: route query for file id toward the closest node id
  - Fetch: when query reaches a node containing file id, it returns the file to the sender

Distributed Hash Tables DHT

- In 2000-2001, academic researchers said “we want to play too!”
- Motivation:
  - Frustrated by popularity of all these “half-baked” P2P apps :)  
  - We can do better! (so we said)
  - Guaranteed lookup success for files in system
  - Provably bounds on search time
  - Provably scalability to millions of node
- Hot Topic in networking ever since
DHT

• Abstraction: a distributed “hash-table” (DHT) data structure:
  – put(id, item);
  – item = get(id);

• Implementation: nodes in system form a distributed data structure
  – Can be Ring, Tree, Hypercube, Skip List, Butterfly Network, ...

DHT

• Structured Overlay Routing:
  – Join: On startup, contact a “bootstrap” node and integrate yourself into the distributed data structure; get a node id
  – Publish: Route publication for file id toward a close node id along the data structure
  – Search: Route a query for file id toward a close node id. Data structure guarantees that query will meet the publication.
  – Fetch: Two options:
    • Publication contains actual file => fetch from where query stops
    • Publication says “I have file X” => query tells you 128.2.1.3 has X, use IP routing to get X from 128.2.1.3

DHT Example: Chord

• Associate to each node and file a unique id in an uni-dimensional space (a Ring)
  – E.g., pick from the range [0...2^m]
  – Usually the hash of the file or IP address

• Properties:
  – Routing table size is O(log N), where N is the total number of nodes
  – Guarantees that a file is found in O(log N) hops
DHT: Consistent Hashing

A key is stored at its successor: node with next higher ID

DHT: Chord Basic Lookup

N90 has K80

"Where is key 80?"

DHT: Chord Finger Table

- Entry i in the finger table of node n is the first node that succeeds or equals n + 2i
- In other words, the ith finger points 1/2^i way around the ring
DHT: Chord Join

• Assume an identifier space \([0..7]\)

• Node n1 joins

\[
\begin{array}{c|c|c}
\text{id} & \text{succ} \\
0 & 2 \ 1 \\
1 & 3 \ 1 \\
2 & 5 \ 1 \\
\end{array}
\]

Succ. Table

DHT: Chord Join

• Node n2 joins

\[
\begin{array}{c|c|c}
\text{id} & \text{succ} \\
0 & 3 \ 1 \\
1 & 4 \ 1 \\
2 & 6 \ 1 \\
\end{array}
\]

Succ. Table

DHT: Chord Join

• Nodes n0, n6 join

\[
\begin{array}{c|c|c}
\text{id} & \text{succ} \\
0 & 2 \ 2 \\
1 & 4 \ 2 \\
2 & 6 \ 2 \\
\end{array}
\]

Succ. Table
DHT: Chord Join

- Nodes: n1, n2, n0, n6
- Items: f7, f1

<table>
<thead>
<tr>
<th>i</th>
<th>id</th>
<th>succ</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

Succ. Table

<table>
<thead>
<tr>
<th>i</th>
<th>id</th>
<th>succ</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>6</td>
</tr>
</tbody>
</table>

Succ. Table

<table>
<thead>
<tr>
<th>i</th>
<th>id</th>
<th>succ</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>6</td>
</tr>
</tbody>
</table>

Succ. Table

Items

<table>
<thead>
<tr>
<th>i</th>
<th>id</th>
<th>succ</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Succ. Table

DHT: Chord Routing

- Upon receiving a query for item id, a node:
  - Checks whether stores the item locally
  - If not, forwards the query to the largest node in its successor table that does not exceed id

DHT

- Pros:
  - Guaranteed Lookup
  - $O(\log N)$ per node state and search scope
- Cons:
  - No one uses them? (only one file sharing app)
  - Supporting non-exact match search is hard
P2P Summary

- Many different styles; remember pros and cons of each
  - centralized, flooding, swarming, unstructured and structured routing
- Lessons learned:
  - Single points of failure are very bad
  - Flooding messages to everyone is bad
  - Underlying network topology is important
  - Not all nodes are equal
  - Need incentives to discourage freeloaders
  - Privacy and security are important
  - Structure can provide theoretical bounds and guarantees