CSCI 339
Distributed Systems

Lecture 6
Internet Services and Naming
Feb 20, 2023
Administrative Details

- Class Thur?? TBD!
- No paper writeup on Glow this week (but paper is still posted for your reference)
- Pick up textbooks in bookstore if you asked for one
- Web server due Wed/Thur
  - Extra help hours this week (see email from yesterday)
  - Commit/push code and writeup to evolene
  - See assignment page for thought questions to include
  - Questions about server? Questions about writeup?
  - Short code reviews/demos for grading? Details TBD.
- Project 2 will be officially posted next week
  - Easier than Project 1 wrt to code (it’s not C!)
Last Time

- Wrapped up networks review
  - TCP congestion and flow control
  - TCP sawtooth
Recap: TCP in Action
Recap: TCP in Action

Slow Start + Congestion Avoidance + Fast Retransmit + Fast Recovery = TCP Sawtooth
Today’s Outline

• Final thoughts on Layer 4 and TCP
• Discuss “Lessons from Giant-Scale Services”
• Discuss challenges associated with naming
**Layer 4 Wrapup**

<table>
<thead>
<tr>
<th>TCP</th>
<th>UDP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection oriented</td>
<td>Connection-less</td>
</tr>
<tr>
<td>On-going conversation</td>
<td>No notion of conversation</td>
</tr>
<tr>
<td>Heavy-weight</td>
<td>Light-weight</td>
</tr>
<tr>
<td>Reliable delivery</td>
<td>No reliability</td>
</tr>
<tr>
<td>In-order delivery</td>
<td>No in-order delivery</td>
</tr>
<tr>
<td>Connection setup and tear down required</td>
<td>No connection setup or tear down</td>
</tr>
<tr>
<td>Flow &amp; congestion control</td>
<td>No flow or congestion control</td>
</tr>
<tr>
<td>What apps need TCP?</td>
<td>What apps don't need TCP (and can use UDP)?</td>
</tr>
</tbody>
</table>
What if Two TCP Connections Share Link?

- Reach equilibrium independent of initial bandwidth (assuming equal RTTs)
What if TCP and UDP Share Link?

- Independent of initial rates, UDP will get priority! TCP will take what's left.

![Graph showing cwnd (congestion window) over round-trip times]

- The red line represents UDP, and the blue line represents TCP.
- Initially, UDP's cwnd increases rapidly, while TCP's cwnd increases more gradually.
- After a short period, both lines stabilize, indicating that UDP has priority and TCP is taking what's left.
Cheating TCP: ACK splitting

Sender

Round-Trip Time (RTT)

Data 1:1461

ACK 486

ACK 973

ACK 1461

Data 1461:2921

Data 2921:4381

Data 4381:5841

Data 5841:7301

Receiver

• Rule: grow window by one full-sized packet for each valid ACK received

• Send M ACKs for one pkt

• Growth factor proportional to M!

• Modern implementations prevent this hack
Cheating TCP:
10 line change to Linux TCP

Page fetch from CNN.com

- Sequence Number (bytes)
- Time (sec)

Modified Client
Normal Client
TCP Summary

• TCP is designed around the premise of cooperation
• There are a bunch of seemingly arbitrary magic numbers
  • Decrease by 2x, increase by 1, 3 duplicate acks, initial timeout = 3 seconds, etc
• But overall it actually works really well…
  • Are you surprised?
Lessons from Giant-Scale Services
By Eric Brewer

Eric Brewer (scientist)

Eric Allen Brewer is professor emeritus of computer science at the University of California, Berkeley and vice-president of infrastructure at Google. His research interests include operating systems and distributed computing. He is known for formulating the CAP theorem about distributed network applications in the late 1990s.

In 1996, Brewer co-founded Inktomi Corporation (bought by Yahoo! in 2003) and became a paper billionaire during the dot-com bubble. Working with the United States federal government during the presidency of Bill Clinton, he helped to create USA.gov, which launched in 2000. His research also included a wireless networking scheme called WILDNet, which promises to bring low-cost connectivity to rural areas of the developing world. He has worked at Google since 2011.

Education
Brewer received a Bachelor of Science in electrical engineering and computer science (EECS) from UC Berkeley where he was a member of the Pi Lambda Phi fraternity. Later he earned a Master of Science and PhD in EECS from MIT. He received tenure from UC Berkeley in 2000.

Awards
In 1999, he was named to the MIT Technology Review TR100 as one of the top 100 innovators in the world under the age of 35.

In 2007, Brewer was inducted as a Fellow of the Association for Computing Machinery "for the design of scalable, reliable internet services." That same year, he was also inducted into the National Academy of Engineering "for the design of highly scalable internet services."

Brewer is the 2009 recipient of the ACM-Infosys Foundation Award in the Computing Sciences "for his contributions to the design and development of highly scalable Internet services."

In 2009, Brewer received the SIGOPS Mark Weiser Award.

In 2013, the ETH Zurich honored him with the title Dr. sc. tech. (honoris causa).
Paper Takeaways

- Clusters as building blocks for large scale services
  - Are clouds the new clusters?
  - Commodity processors vs “super-computers”
  - Incremental scalability, independent components, cost and performance
  - Brewer advocates for extreme symmetry
- Load management
  - Round-robin DNS in 1995
    - Map hostname to multiple IP addresses, hand out in RR fashion
    - Does not hide failure or inactive servers, exposes structure
  - Fancy L4 and L7 switches inspect TCP/HTTP state
    - E2E violation?
- Replication vs Partitioning
Service Replication

- IP network
- Single-site server
- Round-robin DNS
- Simple replicated store
Service Partitioning
Replication versus Partitioning

- **Replication**
  - Any replica can serve any request
  - Failure reduces system capacity but not data availability
  - Must make sure replicas are kept in-sync

- **Partitioning**
  - Nodes are no longer identical so certain requests need to be sent to individual nodes
  - No need for coherence traffic for syncing data
  - Failure reduces data availability and may reduce capacity

- Optimal solution? Which is better?
Harvest and Yield, DQ

• yield = queries completed/queries offered
• harvest = data available/complete data
• Should faults affect yield, harvest, or both?

• DQ Principle
  • Data per query * queries per second → constant
• At high utilization, can increase queries per second by reducing data per query (or vice versa)
• Adding/removing nodes changes the constant
Degradation and Evolution

• Graceful degradation
  • Cost-based admission control
    • Search engine denies expensive query (in terms of D)
    • Rejecting expensive query may allow cheaper ones to complete
  • Priority-based admission control
    • Some requests given different priority relative to others
  • Reduced data freshness
    • Reduce required data movement by allowing stale data

• Evolution and growth
  • Fast reboot – simultaneous reboot all machines
  • Rolling upgrade – upgrade one machine at a time in a “wave”
  • Big flip – upgrade half, flip load balancer to point to upgraded machines
Moving on…
Introduction to Naming

• Naming is a fundamental issue that is often overlooked in distributed system design

• Names refer to a variety of resources
  • Examples?
  • Computers, services, files, remote objects, people (email addresses), etc

• Names are required to locate the desired data
  • For example, consider hostnames in URLs
Terminology

• **Pure names** – un-interpreted bit patterns; must be “looked up” before they are used [Needham 1993]

• **Non-pure names** - contain information about the object they name (such as location or address); inadequate for object identification over time

• **Resolution** - translate a name into useful data about an object

• **Binding** - association between name and object

• **Attribute** - value of a property associated with an object (names are often bound to attributes)

• **Contexts** - sets of bindings between names and attributes
Name Services

• Name services store collections of naming contexts
• Main operation is to support name resolution
  • Look up attributes from a given name
• Also need to support creating new bindings, deleting bindings, and listing bound names
• Goals:
  • **Unification** - resources managed by different services use same naming scheme
  • **Integration** - enable sharing and naming of resources across administrative domains
Design Goals

• Name services must:
  • Handle an arbitrary number of names
  • Have a long (infinite?) lifetime
  • Provide high availability
  • Hide (or isolate) faults
  • Tolerate mistrust
Case Study:
Domain Name System
Motivation

- 1982: A single hosts.txt file stored and distributed (via FTP) from a central site to all computers
  - Flat-naming scheme based on simple table lookup
- Maintained by one man (Jon Postel)
- Contained all hostname to IP address mappings
- Problems
  - Centralized control did not fit with distributed management (organizations wanted control of their names)
  - Scalability - Exponential growth as the number of hosts changed from the number of timesharing systems to number of individual workstations
- A general name service was needed…
- Options?
Jon Postel

- [http://www.wired.com/2012/10/joe-postel/](http://www.wired.com/2012/10/joe-postel/)
- Involved with ARPANET
- RFC Editor
- Original and long-time .us Top-Level-Domain admin
- “god of the Internet”
Motivation

• 1982: A single hosts.txt file stored and distributed (via FTP) from a central site to all computers
  • Flat-naming scheme based on simple table lookup
• Maintained by one man (Jon Postel)
• Contained all hostname to IP address mappings
• Problems
  • Centralized control did not fit with distributed management (organizations wanted control of their names)
  • Scalability - Exponential growth as the number of hosts changed from the number of timesharing systems to number of individual workstations
• A general name service was needed…
• Options?
Domain Name System

• Hierarchical namespace
  • Highest level domain is on right side of name
  • com, edu, gov, mil, net, org, int, …
• Control delegated in hierarchical fashion (i.e., a tree)
  • Convince node “above” you to delegate control to you
• Designed to be extensible, robust, scalable
• DNS database is distributed (through combination of replication and data partitioning) across logical network of servers
• 1985: Some hosts solely utilize DNS
Hierarchical Design

Top Level Domains
gTLDs

. (root)

org    mil    edu    com    uk    ca

ucsd    amherst    williams    harvard    mit

www    cs    oit

limia    lohani    sysnet
Domain Name System (DNS)

- Translate human understandable names to machine understandable names
  - E.g., www.cs.williams.edu ➔ 137.165.8.2
- Hierarchical structure
  - Every DNS server knows where the “root” is
  - The root can tell you how to get to .edu
  - .edu server can tell you how to find williams.edu
  - williams.edu tells you about cs.williams.edu
  - cs.williams.edu translates www.cs.williams.edu ➔ 137.165.8.2
- Caching along the way to improve performance
Query Processing

• Clients (i.e., your computers) query local name server
  • Local name server can provide authoritative or cached answers

• Many name servers support both recursive and iterative queries
  • Recursive queries - servers recursively find answer and return it to resolver (host who originally submitted query)
  • Iterative queries - each server responds to resolver with information about next server to ask
  • (more on next slide)

• If response is not cached locally, locate server **lowest** in the hierarchy with entry in local DB
  • In the worst case, contact root (.)
  • Cache locally with TTL
Recursive DNS Lookup Example

(Note that the servers could also just route the request among themselves rather than going back to the local nameserver each time.)
Iterative DNS Lookup Example

Local nameserver only tells client what it already knows.
Domains and Zones

• Domains
  • Entire branches in tree

• Zones
  • Portions of a domain (anchored at domain nodes)
  • Any contiguous set of nodes in the tree
  • Provide local autonomy
  • Can be grown to arbitrary size
  • Each zone should provide redundant name servers
Zones and Domains

edu Zone

.org mil .(root)

com uk ca

ucsd amherst www williams harvard mit

.edu limia oit sysnet

edu Domain

Zone and Domain
Zones and Domains

- Zone, NOT Domain
- edu Domain
- edu Zone
- . (root)
- org
- mil
- com
- uk
- ca
- ucsd
- amherst
- williams
- harvard
- mit
- www
- cs
- oit
- sysnet
- limia
- lohani
- www.cs.edu
- www.cs.org
- www.cs.mil
- www.cs.uk
- www.cs.ca
Caching

- Caching (and replication) are used to improve performance
- Name servers cache lookups for some period of time
  - Time to keep data in cache determined by TTL value
  - (TTL = time to live)
  - Design decisions:
    - Low TTL -> more accurate results, more lookups
    - High TTL -> less accurate results (stale data), less lookups
- Caching can significantly reduce lookup time and traffic, but only if the right design choices are made
- Cache poisoning
1988 Status

- 20k hosts available through DNS
- 30 top level domain names
- 7 Root servers
  - 1 query per second, driven by tuning of parameters
- Query breakdown
  - All info (25-40%)
  - Hostname to address (30-40%)
  - Address to hostname (10-15%)
  - Mail MX record (<10%)
Discussion

- Where does security fit into DNS?
- What tradeoffs are associated with caching DNS lookups?
- Who controls DNS root servers?
- Who should control them?
- International domain names?
DNS Attacks

- [https://defintel.com/blog/index.php/2017/05/these-6-dns-attacks-threaten-your-business.html](https://defintel.com/blog/index.php/2017/05/these-6-dns-attacks-threaten-your-business.html)

- Distributed DoS attack (DDoS)

- Distributed Reflection DoS attack

- Cache poisoning

- TCP SYN floods

- DNS tunneling

- DNS hijacking/redirect
  - [https://www.wired.com/story/what-is-dns-hijacking/](https://www.wired.com/story/what-is-dns-hijacking/)
DNS Reflection/Amplification
DDOS Attack

https://www.youtube.com/watch?v=xTKjHWkDwP0

DNS TCP SYN flood
DDOS Attack

https://defintel.com/blog/index.php/2017/05/these-6-dns-attacks-threaten-your-business.html
Other Naming Services

• Directory services
  • Name services allow users to find attributes for a given name
  • But how do you find names for a given attribute?
• “Yellow pages services”
• Example:
  • Find all computers running Mac OS X in this building
  • Find closest printer
• Popular directory services
  • Microsoft’s Active Directory Services, X.500, LDAP
• There are also special directory services called discovery services
• Next up: How can we maintain/build directory and discovery services?
Directory Service

• **Formal definition**
  • A service that stores collections of bindings between names and attributes
  • Looks up entries that match attribute-based specifications

• **Popular examples**
  • X.500
  • LDAP (Lightweight Directory Access Protocol)
X.500

- Application level service in OSI set of standards (1988?)
- Data stored in X.500 servers is organized into a tree structure
- Name tree is called the Directory Information Tree (DIT)
- Entire directory structure is called Directory Information Base (DIB)
- Servers are called Directory Service Agents (DSA)
- Clients are Directory User Agents (DUA)
- X.500 is almost like DNS for “people”
X.500

- DIB entry consists of a set of attributes
- Each attribute has a type and one or more values
- Name of DIB entry is determined by selecting distinguished attributes called Distinguished Names (DN)

- Accessing the directory:
  - Read - specify name (similar to domain name) and desired attributes, DSA navigates DIT and returns requested information
  - Search - specify base name and filter expression, DSA returns DNs for all entries below base name for which filters evaluate to true
X.500

- **Updating the DIB**
  - DSA interface supports adding, deleting, modifying entries in DIB
  - Expected that DIB is partitioned and replicated, but X.500 standard does not address implementation issues directly

- **Issues with X.500**
  - Very heavy-weight! Complex and difficult to implement
  - Uses upper layers of network stack
  - Check out Wikipedia…
LDAP to the Rescue

- X.500 is too complex for many applications
- *Lightweight Directory Access Protocol (LDAP)* is based on X.500, but is simplified
- Runs over TCP/IP
- LDAP, unlike X.500, is widely used in Internet applications
- LDAP directory service consists of a number of records made up of (attribute, value) pairs
**LDAP**

- Sample LDAP namespace:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Abbr.</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Country</td>
<td>C</td>
<td>US</td>
</tr>
<tr>
<td>Locality</td>
<td>L</td>
<td>Massachusetts</td>
</tr>
<tr>
<td>Organization</td>
<td>O</td>
<td>Williams College</td>
</tr>
<tr>
<td>OrganizationalUnit</td>
<td>OU</td>
<td>Comp. Sci.</td>
</tr>
<tr>
<td>CommonName</td>
<td>CN</td>
<td>Main server</td>
</tr>
<tr>
<td>Mail_Servers</td>
<td>--</td>
<td>137.164.8.4, 137.165.8.5</td>
</tr>
<tr>
<td>WWW_Server</td>
<td>--</td>
<td>137.165.8.6</td>
</tr>
</tbody>
</table>
LDAP

- Directory entries are again called directory information base (DIB)
- Records are uniquely named so that they can be looked up
- Unique name is derived from sequence of naming attributes
- Use of globally unique names obtained by listing naming attributes in sequence leads to a hierarchy (as in DNS) called directory information tree (DIT)
LDAP

• DIT is partitioned and distributed across several servers (DSAs)
• Each DSA behaves like name server in DNS
• Key difference between LDAP and DNS are the facilities for searching through a DIB
• For example, perhaps we want to know all main servers at Williams College
  • answer = search("&(C=US)(O=Williams College)(OU = *)(CN=Main server)")
• These queries are not possible in DNS
• But searches like this can be expensive to complete
Problem

- DNS is a relatively simple hierarchical *name service* that does not provide “yellow pages” style searching mechanisms.
- X.500 and LDAP are hierarchical *directory services* that provide advanced searching though it can be expensive (have to visit many leaves in the tree).
- How can we avoid expensive searching?
Decentralized Solutions

- What about decentralized and distributed attribute-based naming systems?
- The goal is to provide efficient searching by avoiding an exhaustive (expensive) search
- We can use distributed hash tables to avoid expensive searches and provide efficient lookups
Distributed Hash Tables

• Hash tables map keys (attributes) to values to provide simple and efficient lookups without searching
• DHTs are essentially hash tables that are partitioned and spread across several nodes in a peer-to-peer (P2P) system
• DHTs are built on P2P systems, and tend to be scalable and fault tolerant