NAMING, DNS, AND CHORD

George Porter
Oct 30 and Nov 1, 2018

ATTRIBUTION

• These slides are released under an Attribution-NonCommercial-ShareAlike 3.0 Unported (CC BY-NC-SA 3.0) Creative Commons license
• These slides incorporate material from:
  • Michael Freedman and Kyle Jamieson, Princeton University (also under a CC BY-NC-SA 3.0 Creative Commons license)
ANNOUNCEMENTS

Reading for this week:
Van Steen 5 and Birrell and Nelson paper

“Boxed” text in the assigned reading is optional unless otherwise indicated
“Decentralized implementations” to the end of the chapter is optional

HW 5 overview
File download example

User -> Client -> Metadata -> BlockStore

1. User requests file download.
2. Client reads file.
3. Client checks file exists.
4. Client downloads missing blocks.
5. Client merges blocks to form file.
6. Client writes file to disk.
7. Client signals success.

File upload example

User -> Client -> Metadata -> BlockStore

2. Client reads file.
3. Client checks if file exists.
4. Client computes hashlist.
5. Client checks for all blocks in hashlist.
6. Client checks BlockStore for missing blocks.
7. Client responds with missing block list.
8. For any blocks missing in the BlockStore, client checks for block.
9. Client responds with hashlist.
10. Client checks for block.
11. Client responds with hashlist.
Outline

• Naming overview
• Flat naming
• Routing over flat names via Chord
• Hierarchical naming and DNS
• Attribute-based naming and LDAP
NAMING OVERVIEW

- Access point: way of contacting resource in a networked system
- Address: name of an access point
  - Can change over time
- Names
  - Flat
  - Structured
  - Attribute-based
- Example: Ethernet
  - Access point: ?
  - Address: {Flat,Structured,Attribute}?
  - Do Ethernet addresses change over time?
  - Given an Ethernet address, how do you find the corresponding access point?
  - Advantages to flat names? Disadvantages?
  - Advantages to fixed names? To names that change?

EXAMPLE FROM INDUSTRY: MICROSOFT

- “Ananta” web load balancer
- Challenge:
  - How to scale to large number of web clients beyond what a single server can handle?
- Solution:
  - Separate access point(s) from service address
  - A “MUX” maps a single service address to one or more service addresses transparently to the end user

```json
{ "VIP": "1.2.3.4", "Endpoint": [ { "protocol": "tcp", "port": "80", "DIP": [ { "Host": "1.1.1.0", [IP]: "1.1.1.1", "port": "8080", } ] }, "SNAT": [ [IP]: "1.1.1.1", [IP]: "2.2.2.2" ] ]
```
SERVICE NAMING

• In Ananta, name is:
  • Visible to end user, public, fixed (over long time periods), unique
• But underlying services/servers
  • Come and go according to load
  • Fail periodically
  • Get reassigned to other services
• Need way to map name to service access point
  • Scalable, fault tolerant, simple
  • Yet dynamic to overcome failures
  • Dynamic to handle changing load (direct more users to one server as compared to another due to overload)

NAMING COMPONENTS

• www.cs.ucsd.edu
• /home/aturing/docs/paper.pdf
• 03:A8:BF:01:00:2C
• 206.109.21.7
• aturing@cs.ucsd.edu
• aturing
NAMING HIERARCHY FOR SCALE

• Host name: www.ucsd.edu
  • Domain: registrar for each top-level domain (e.g., .edu)
  • Host name: local administrator assigns to each host

• IP addresses: 128.54.70.238
  • Prefixes: ICANN, regional Internet registries, and ISPs
  • Hosts: static configuration, or dynamic using DHCP

• MAC addresses: 58:B0:35:F2:3C:D9
  • OIDs: assigned to vendors by the IEEE
  • Adapters: assigned by the vendor from its block

MAPPING BETWEEN IDENTIFIERS

• Domain Name System (DNS)
  • Given a host name, provide the IP address
  • Given an IP address, provide the host name

• Address Resolution Protocol (ARP)
  • Given an IP address, provide the MAC address
  • To enable communication within the Local Area Network

• Dynamic Host Configuration Protocol (DHCP)
  • Automates host boot-up process
  • Given a MAC address, assign a unique IP address
  • ... and tell host other stuff about the Local Area Network
**Outline**

- Naming overview
- Flat naming
- Routing over flat names via Chord
- Hierarchical naming and DNS
- Attribute-based naming and LDAP

**BROADCAST: ADDRESS RESOLUTION PROTOCOL (ARP)**

- IP forwarding tables: one entry per *network* not *host*
- Thus, routes designed to get packets to proper network
- Network needs to take over from there to get to proper host
- **Address resolution protocol (ARP)** translates IP addresses to link-level addresses (e.g., 48-bit Ethernet addr)
  - Broadcast request over network for IP ➔ link-level mapping
  - Maintain local cache (with timeout)
ARP OVERVIEW

Broadcast: Anyone know the Ethernet address for 152.3.145.240?

Reply: Yes, I’m at 08-00-2b-18-bc-65 (152.3.140.5)

What about inter-networked subnets/networks?

broadcast to handle autoconfiguration

- Host doesn’t have an IP address yet
  - So, host doesn’t know what source address to use

- Host doesn’t know who to ask for an IP address
  - So, host doesn’t know what destination address to use

- Solution: shout to discover a server who can help
  - Install a special server on the LAN to answer distress calls
DHCP

- Broadcast-based LAN protocol algorithm
  - Host broadcasts “DHCP discover” on LAN (e.g. Ethernet broadcast)
  - DHCP server responds with “DHCP offer” message
  - Host requests IP address: “DHCP request” message
  - DHCP server sends address: “DHCP ack” message w/IP address

- Easy to have fewer addresses than hosts (e.g. UCSD wireless) and to *renumber* network (use new addresses)

- What if host goes away (how to get address back?)
  - Address is a “lease” not a “grant”, has a timeout
  - Host may have different IP addresses at different times?

FORWARDING POINTERS

- Simple idea: leave behind pointer

Internet Engineering Task Force (IETF) C. Perkins, Ed.
Request for Comments: 5944 WICHERS Inc.
Obsoletes: 2114 November 2010
Category: Standards Track
ISBN: 1078-1721

**IP Mobility Support for IPv4, Revised**

Abstract

This document specifies protocol enhancements that allow transparent routing of IP datagrams to mobile nodes in the Internet. Each mobile node is always identified by its home address, regardless of its current point of attachment to the Internet. While situated away from its home, a mobile node is also associated with a care-of address, which provides information about its current point of attachment to the Internet. The protocol provides for registering the care-of address with a home agent. The home agent sends datagrams destined for the mobile node through a tunnel to the care-of address. After arriving at the end of the tunnel, each datagram is then delivered to the mobile node.
Generalizing forwarding pointers?

- Can we automate the following of forwarding pointers?
  - Scalable
    - What do we mean by scalable?
  - Fault-tolerant
  - Performant?
Flat Naming and Peer-to-Peer (P2P) Networks

- A **distributed** system architecture:
  - No centralized control
  - Nodes are **roughly symmetric** in function
  - Large number of **unreliable** nodes
P2P ADOPTION

• Successful adoption in some niche areas –

1. Client-to-client (legal, illegal) file sharing
   • Popular data but owning organization has no money

2. Digital currency: no natural single owner (Bitcoin)

3. Voice/video telephony: user to user anyway
   • Issues: Privacy and control

EXAMPLE: CLASSIC BITTORRENT

1. User clicks on download link
   • Gets torrent file with content hash, IP addr of tracker

2. User’s BitTorrent (BT) client talks to tracker
   • Tracker tells it list of peers who have file

3. User’s BT client downloads file from one or more peers

4. User’s BT client tells tracker it has a copy now, too

5. User’s BT client serves the file to others for a while

Provides huge download bandwidth, without expensive server or network links
FLAT NAME LOOKUP PROBLEM

Publisher (N₄)

Internet

N₁ → N₂

get("Star Wars.mov")

N₃ → N₄

put("Star Wars.mov", [content])

N₅ → N₆

CENTRALIZED LOOKUP (NAPSTER)

Publisher (N₄)

N₁ → N₂

SetLoc("Star Wars.mov", IP address of N₄)

N₃

Client

DB

Lookup("Star Wars.mov")

Simple, but O(N) state and a single point of failure
Flooded Queries (Original Gnutella)

Robust, but $O(N = \text{number of peers})$ messages per lookup

Routing DHT Queries (Chord)

Can we make it robust, reasonable state, reasonable number of hops?
SYSTEMATIC FLAT NAME LOOKUPS VIA DHTS

• Local hash table:
  key = Hash(name)
  put(key, value)
  get(key) → value

• **Service**: Constant-time insertion and lookup

> How can I do (roughly) this across millions of hosts on the Internet?
> Distributed Hash Table (DHT)

WHAT IS A DHT (AND WHY)?

• Distributed Hash Table:
  key = hash(data)
  lookup(key) → IP addr *(Chord lookup service)*
  send-RPC(IP address, put, key, data)
  send-RPC(IP address, get, key) → data

• **Partitioning data** in truly **large-scale distributed systems**
  • Tuples in a global database engine
  • Data blocks in a global file system
  • Files in a P2P file-sharing system
WHY MIGHT DHT DESIGN BE HARD?

• Decentralized: no central authority

• Scalable: low network traffic overhead

• Efficient: find items quickly (latency)

• Dynamic: nodes fail, new nodes join

CHORD LOOKUP ALGORITHM PROPERTIES

• **Interface**: lookup(key) → IP address

• **Efficient**: $O(\log N)$ messages per lookup
  • $N$ is the total number of servers

• **Scalable**: $O(\log N)$ state per node

• **Robust**: survives massive failures
CHORD IDENTIFIERS

- **Key identifier** = SHA-1(key)

- **Node identifier** = SHA-1(IP address)

- SHA-1 distributes both uniformly

- **How does Chord partition data?**
  - *i.e.*, map key IDs to node IDs

CONSISTENT HASHING [KARGER ‘97]

Key is stored at its successor: node with next-higher ID
CHORD: SUCCESSOR POINTERS

BASIC LOOKUP

"Where is K80?"

"N90 has K80"
SIMPLE LOOKUP ALGORITHM

**Lookup** (key-id)

succ \(\leftarrow\) my successor

if my-id < succ < key-id \(/\text{next hop}\)

    call Lookup(key-id) on succ

else \(/\text{done}\)

    return succ

- **Correctness** depends only on successors

IMPROVING PERFORMANCE

- **Problem:** Forwarding through successor is slow

- Data structure is a linked list: \(O(n)\)

- **Idea:** Can we make it more like a binary search?

  - Need to be able to halve distance at each step
CHORD INTUITION

• Skip Lists (Pugh, 1989)

• Consider a linked list:

• Lookup time: \( O(n) \)

CHORD INTUITION

• Skip Lists (Pugh, 1989)

• Consider a linked list:

• Add 2\(^{nd}\) row of pointers spaced further apart
  • Still \( O(n) \), but more efficient
  • Use 2\(^{nd}\) row to get as close as possible without going over
  • Then last row to get to the desired element
CHORD INTUTION

- Skip Lists (Pugh, 1989)
- Consider a linked list:

![Linked List Diagram]

- Add log(N) rows
  - Get as close as possible on top row, then drop down a row, then drop down another row, until the bottom row
  - O(log N) lookup time

Figure 5-4. Resolving key 26 from node 1 and key 12 from node 28 in a Chord system.
**AN ASIDE: IS LOG(N) FAST OR SLOW?**

- For a million nodes, it’s 20 hops
- If each hop takes 50 milliseconds, lookups take a second
- If each hop has 10% chance of failure, it’s a couple of timeouts
- So in practice log(n) is better than O(n) but not great

**JOINING: LINKED LIST INSERT**

1. Lookup(36)
JOIN (2)

2. N36 sets its own successor pointer

JOIN (3)

3. Copy keys 26..36 from N40 to N36
**NOTIFY MESSAGES MAINTAIN PREDECESSORS**

```
N25
N40
N36

notify N25

notify N36
```

**STABILIZE MESSAGE FIXES SUCCESSOR**

```
N25
N40
N36

stabilize

“My predecessor is N36.”
```
JOINING: SUMMARY

- Predecessor pointer allows link to new node
- Update finger pointers in the background
- Correct successors produce correct lookups

WHAT DHTS GOT RIGHT

- **Consistent hashing**
  - Elegant way to divide a workload across machines
  - Very useful in clusters: actively used today in Amazon Dynamo and other systems

- **Replication** for high availability, efficient recovery after node failure

- **Incremental scalability**: “add nodes, capacity increases”

- **Self-management**: minimal configuration

- **Unique trait**: no single server to shut down/monitor
Outline

- Naming overview
- Flat naming
- Routing over flat names via Chord
- Hierarchical naming and DNS
- Attribute-based naming and LDAP

DNS HOSTNAME VERSUS IP ADDRESS

- **DNS host name** (e.g. www.cs.ucsd.edu)
  - Mnemonic name appreciated by humans
  - Variable length, full alphabet of characters
  - Provides little (if any) information about location
- **IP address** (e.g. 128.112.136.35)
  - Numerical address appreciated by routers
  - Fixed length, decimal number
  - Hierarchical address space, related to host location
MANY USES OF DNS

- Hostname to IP address translation
- IP address to hostname translation (reverse lookup)
- Host name aliasing: other DNS names for a host
- Alias host names point to canonical hostname
- Email: Lookup domain’s mail server by domain name

ORIGINAL DESIGN OF DNS

- Per-host file named /etc/hosts (1982)
- Flat namespace: each line = IP address & DNS name
- SRI (Menlo Park, California) kept the master copy
- Everyone else downloads regularly
- But, a single server doesn’t scale
  - Traffic implosion (lookups and updates)
  - Single point of failure
- Need a distributed, hierarchical collection of servers
DNS: GOALS AND NON-GOALS

• A wide-area distributed database
• Goals:
  • Scalability; decentralized maintenance
  • Robustness
  • Global scope
    • Names mean the same thing everywhere
  • Distributed updates/queries
  • Good performance
• But don’t need strong consistency properties

DOMAIN NAME SYSTEM (DNS)

• Hierarchical name space divided into contiguous sections called zones
  • Zones are distributed over a collection of DNS servers
• Hierarchy of DNS servers:
  • Root servers (identity hardwired into other servers)
  • Top-level domain (TLD) servers
  • Authoritative DNS servers
• Performing the translations:
  • Local DNS servers located near clients
  • Resolver software running on clients
DNS IS HIERARCHICAL

- Hierarchy of namespace matches hierarchy of servers
- Set of nameservers answers queries for names within zone
- Nameservers store names and links to other servers in tree

DNS ROOT NAMESERVERS

- 13 root servers. Does this scale?

Root

TLDs: com. gov. edu.
cs.ucsd.edu.

Root servers:
A Verisign, Dulles, VA
B USC-ISI Marina del Rey, CA
C Cogent, Herndon, VA
D U Maryland College Park, MD
E NASA Mt View, CA
F Internet Software Consortium, Palo Alto, CA
G US DoD Vienna, VA
H ARL Aberdeen, MD
I Verisign
J Verisign
K Autonomica, Stockholm
L ICANN Los Angeles, CA
M WIDE Tokyo
DNS ROOT NAMESERVERS

- 13 root servers. *Does this scale?*
- Each server is really a *cluster* of servers (some geographically distributed), replicated via *IP anycast*

![Map of DNS Root Nameservers](image)

TLD AND AUTHORITATIVE SERVERS

- Top-level domain (TLD) servers
  - Responsible for com, org, net, edu, etc, and all top-level country domains: uk, fr, ca, jp
  - Network Solutions maintains servers for com TLD
  - Educause non-profit for edu TLD
- Authoritative DNS servers
  - An organization’s DNS servers, providing authoritative information for that organization
  - May be maintained by organization itself, or ISP
LOCAL NAME SERVERS

- Do not strictly belong to hierarchy
- Each ISP (or company, or university) has one
  - Also called default or caching name server
- When host makes DNS query, query is sent to its local DNS server
  - Acts as proxy, forwards query into hierarchy
  - Does work for the client

DNS RESOURCE RECORDS

- DNS is a distributed database storing resource records
- Resource record includes: (name, type, value, time-to-live)

  **Type = A (address)**
  - name = hostname
  - value is IP address

  **Type = CNAME**
  - name = alias for some “canonical” (real) name
  - value is canonical name

  **Type = NS (name server)**
  - name = domain (e.g. princeton.edu)
  - value is hostname of authoritative name server for this domain

  **Type = MX (mail exchange)**
  - name = domain
  - value is name of mail server for that domain
DNS IN OPERATION

- Most queries and responses are UDP datagrams
- Two types of queries:
  - **Recursive**: Nameserver responds with answer or error
  - **Iterative**: Nameserver may respond with a referral

ITERATIVE DNS LOOKUP

1. [nl,vu,cs,ftp]
2. #[nl], [vu,cs,ftp]
3. [vu,cs,ftp]
4. #[vu], [cs,ftp]
5. [cs,ftp]
6. #[cs], [ftp]
7. [ftp]
8. #[ftp]

Nodes are managed by the same server

Referral: .edu NS a.edu-servers.net.
RECURSIVE DNS LOOKUP

1. [nl,vu,cs,ftp]

2. [vu,cs,ftp]

3. [cs,ftp]

4. [ftp]

5. [ftp]

6. [cs,ftp]

7. [vu,cs,ftp]

8. #[nl,vu,cs,ftp]

RECURSIVE LOOKUP STATE

<table>
<thead>
<tr>
<th>Server for node</th>
<th>Should resolve</th>
<th>Looks up</th>
<th>Passes to child</th>
<th>Receives and caches</th>
<th>Returns to requester</th>
</tr>
</thead>
</table>
DNS CACHING

- Performing all these queries takes time
  - And all this before actual communication takes place
- Caching can greatly reduce overhead
  - The top-level servers very rarely change
    - Popular sites visited often
  - Local DNS server often has the information cached
- How DNS caching works
  - All DNS servers cache responses to queries
  - Responses include a time-to-live (TTL) field
    - Server deletes cached entry after TTL expires

Outline

- Naming overview
  - Flat naming
  - Routing over flat names via Chord
- Hierarchical naming and DNS
- Attribute-based naming and LDAP
LDAP OVERVIEW

- LDAP is a hierarchical, extensible, searchable data store
- Used in many networked services
  - Login (ieng6)
  - Active Directory
    - Many UCSD services
  - UCSD Wireless
  - Mobile networks
    - Every time you make a call on AT&T, for example
  - Permissions in Linux

STRUCTURING DATA IN LDAP
**LDIF FILE EXCERPT**

- `dn: dc=example,dc=com`
- `dc: example`
- `description: The best company in the whole world`
- `objectClass: dcObject`
- `objectClass: organization`
- `o: Example, Inc.`

- `dn: ou=people, dc=example,dc=com`
- `ou: people`
- `description: All people in organisation`
- `objectClass: organizationalUnit`

**RELATIVE DNS**

**DN is the sum of all RDNs**

- `dn: dc=example,dc=com`
- `dn: ou=people, dc=example,dc=com`
- `cn=Robert Smith`
- `cn=Billy Smith`
- `cn=John Smith`

- `dn: cn=Robert Smith, ou=people, dc=example,dc=com`
- `dn: cn=Billy Smith, ou=people, dc=example,dc=com`
- `dn: cn=John Smith, ou=people, dc=example,dc=com`