Network Layering, Naming, and Name Resolution

CS 118

Computer Network Fundamentals

Peter Reiher
Outline

• Naming within layers

• Name space issues

• Resolving names
What do names mean?

• Names are just labels
  – Until told otherwise, they *mean* nothing
  – *Unique within a layer*: the only thing you know

• They *can* they mean more
  – Layer
  – Physical location
  – Network location
Impact of multiple layers

- Multiple name spaces
- Multiple names for the same party
- Names for parties, names for relays
- The need for translation
Multiple name spaces

• Names are unique to a layer
  – May or may not be unique for all layers

• Name in one layer != name in other
  – One party might have multiple names
    • For each layer in which it participates
    • For each party it emulates
  – Or no name at all
    • Without a name, what happens?
    • Can it participate? How?
Translation

• Mapping between names (labels)
  – DNS: lever.cs.ucla.edu -> 131.179.128.30
  – Reverse DNS: 131.179.128.30 -> lever.cs.ucla.edu
  – Google: Peter Reiher->
    http://fmg-www.cs.ucla.edu/reiher/
Layering requires translation

Change of layer = change of name

- Each layer has its own namespace

- Need to find translation:
  - Equivalent of a local name in new namespace
Example: my PC to your MAC
Example: my PC to your MAC

• Labels in the common layer
  – My PC = banana
  – Your MAC = apple

• Labels when we’re on the same lower layer:
  – My PC = silver
  – Your MAC = copper
Getting there…

• I send a message to you:
  – I send a message from banana to apple
• Translate to lower layer
  – Source: banana to silver
  – Destination: apple to copper
• Relay at lower layer
  – Message goes from silver to copper
• Translate back
  – Message at copper (you) is re-addressed from banana to apple
Layering

• Nodes have names at multiple layers
  – Endpoint names in the common, higher layer
  – Local names in individual lower layers
Result

• Layering requires resolution
  – Can’t get between different layers otherwise
  – Can’t get across different layers otherwise

• Resolution is more than just another name
  – Enables aliasing, proxying
Name Space Issues

• Semantics

• Human factors (ease of use)

• Management
Name semantics

- Location
  - Physical
  - Logical

- Route
  - Sequence of waypoints (intermediate nodes)

- Ownership
  - Who delegated it
  - Who owns it now
Naming human factors

• Express

• Remember

• Guess
Expressing names

• Character set
  – IPv4 – Decimal
  – IPv6, Ethernet – Hexadecimal
  – Old DNS – a-z, 0-9, “-” but can’t start/end with “-”
  – New DNS – encodes other characters as “old DNS” strings

• Grouping
  – 128.9.160.161 (4 groups of 8b)
  – 2001:0:9d38:6ab8:0:0:0:6b35 (8 groups of 16b)

• Shorthand
  – 2001:0:9d38:6ab8:0:0:0:6b35 =
    2001:0:9d38:6ab8::6b35
Remembering names

• DNS’s original purpose
  – Text shorthand is easier than writing numbers

• lever.cs.ucla.edu vs. 131.179.128.30
Guessing names

• DNS’s evolved purpose

• Apple.com
  – Apple records (Paul McCartney)
  – Apple computers (Steve Jobs)
  – US copyright has multiple categories
    • Literary works, dramatic works, musical works, pantomime and choreography, motion picture works, etc.
  – .com is one category

• Should we be able to “guess” a DNS name?
Naming management

• Assignment

• Dynamics

• Indirection

• Implementation
Name assignment

• Authoritative
  – Externally assigned

• Local
  – Self-assigned
  – Might collide (how do we fix this?)
Name dynamics

• Add names

• Remove names

• Change names
Adding names

• Not just “collision avoidance”
  – US street numbers
    • Odd/even per side
    • In order along a street
    • Can’t just choose any unused house number
  – Between 33 and 35? 33½!
Removing names

• Easy to stop using, but:
  – Who to inform?
  – When to inform?
  – What to do with “in transit”?
Changing names

• Combines add and remove, with a twist
  – Avoid collision
  – Inform of change?
  – Forward in-transit?
Benefits of indirection

• Self-management
  – I can assign lever.cs.ucla.edu’s IP address automatically, or it can discover one for itself.

• Mobility
  – I can move lever.cs.ucla.edu to another IP address without needing to tell you at all.

• Overloading
  – Aliases and proxies
Name overloading

- Aliases
  - Multiple names for one party

- Proxies
  - One name for multiple parties
Uses of aliases

• Support internetworking
  – The bridge has at least three names:
    • Name in the upper, common layer
    • Name in the egress layer (the one you’re leaving)
    • Name in the ingress layer (the one you’re entering)

• Emulation
  – When a node acts as if it were more than one node
Uses of proxies

• Load distribution
  – www.google.com → many IP addresses

• Localization
  – www.netflix.com → “nearby” IP address

• Fault tolerance
  – Backup time servers
Implementation issues

• Name management (support naming itself)
  – Cost to add
  – Cost to delete
  – Cost to index (lookup)
  – Locking (for consistency, if distributed)

• Same as any a large, distributed database
  – Caching, hierarchies, etc.
  – Structure in the names can help us
Terminology

• Resolution
  – Translating a name in one context into a name in a different context
Resolution mapping

Name(type1) $\rightarrow$ name(type2)

- **Aliases**
  - Types can be same: Joe $\rightarrow$ Joseph

- **Address resolution**
  - Name translated to an address
  - An address is just a name used for relaying

- **Either direction**
  - Upper layer to lower layer for relaying
  - Lower layer to upper layer for upper-layer name ops
Resolution mechanisms

• Fixed

• Local

• Central

• Distributed
Fixed

• Link broadcast
  – 255.255.255.255 (IPv4)
  – FF:FF:FF:FF:FF:FF (Ethernet)

• Specific groups
  – All nodes, all routers, all DHCP servers (IPv6)

• Me [loopback]
  – 127.0.0.1 – 127.0.0.255 (IPv4)
  – ::1 (IPv6)
Fixed pros and cons

• Pros
  – Always works

• Cons
  – Useful only for a very small set of names
  – No support for operations (add/delete/change)
Local

• Filename internal to the machine
  – Hosts.txt
    • DNS names to IP addresses
    • myname.here ➔ 10.0.0.2
  – Services.txt
    • Service names to registered port numbers
    • HTTP ➔ 80
    • SMTP ➔ 25
Local pros and cons

• Pros
  – Easy to implement
  – Cheap to resolve

• Cons
  – Hard to manage
  – Doesn’t scale
  – Difficult to keep consistent across endpoints
  – Inefficient operations (add/delete/rename)
Central

- A single server at a known address
  - Server can be implemented various ways
  - ATM LAN emulation (LANE)
  - Napster (originally)

- NOTE: emulates broadcast queries
Central pros and cons

• Pros
  – Easy to coordinate
  – Easy to deploy
  – Can be fairly efficient (e.g., use a real DB engine)
  – Scales bigger than you think (Napster)

• Cons
  – Doesn’t scale, eventually
  – Not fault tolerant
    (but you mitigate this with a “hot spare”)


Distributed

• In-band

• Out-of-band
In-band

• RPC, TCPMUX use
  – Open a connection
    • To the TCPMUX service (port 1)
    • To the RPC portmap/rpcbind (port 111)
  – Indicate the service you want by name
    • Get transferred to that service (TCPMUX)
    • Get told what port to use instead

• Registration
  – RPC programs register with portmap
  – TCPMUX uses services.txt
In-band pros and cons

• Pros
  – Easy to deploy independently
  – Can be very lightweight

• Cons
  – Replication of effort (no shared effort)
  – Difficult to scale
Out-of-band

• Hierarchical
  – Using structure in the name space
  – Matching that to distributed system structure
  – Structure limits where you need to look

• Search
  – Look around in “plausible” places

• Broadcast
  – Flood (look everywhere)
Hierarchical

• Ordered
  – Specific name structure

• Follow through the hierarchy to resolve the name
DNS

• Hierarchical names
  – Right to left
  – Starts with “.ROOT” (implicit)

• Two mechanisms:
  – Iterative
  – Recursive
DNS Hierarchy

- Tree of strings

```
  .arpa
 /     \
|       |
|       |
  .com  .edu
 /     \
|       |
|       |
  .net  .uk  .fr
      /     \
     |       |
     |       |
   .in-addr  .yahoo  .colorado  .berkeley  .xbalanque
       /     \
      |       |
      |       |
        /     \
       |       |
       |       |
     .www
```
Roots

• Need to start somewhere
  – As with any naming, need to KNOW something first
  – 13 DNS root servers

• Most clients don’t start at the root
  – They ask a server (pre-configured)
  – The servers then start at the root (if needed)
Iterative vs. recursive

• Start at the root (in either case)
• In iterative, querier asks DNS servers in turn for help
• In recursive, querier asks root server
  – Which recursively asks the other servers
• If querier knows intermediate point, start there
  – Rather than from the root
Iterative DNS

- (ROOT)
  - .edu = X
  - .com = Y
  - .de = Z

- .edu
  - usc.edu = P
  - ucla.edu = Q

- ucla.edu
  - ee.ucla.edu = J
  - cs.ucla.edu = K

- cs.ucla.edu
  - lever.cs.ucla.edu = R
  - www.cs.ucla.edu = S

- www.cs.ucla.edu

Web Server
Iterative DNS

. (ROOT)
.edu = X
.com = Y
.de = Z

.edu
usc.edu = P
ucla.edu = Q

ucla.edu
ee.ucla.edu = J
cs.ucla.edu = K

.cs.ucla.edu
lever.cs.ucla.edu = R
www.cs.ucla.edu = S

www.cs.ucla.edu

Web Server

I know where .edu is!
Iterative DNS

. (ROOT)
edu = X
.com = Y
de = Z

.edu
usc.edu = P
ucla.edu = Q

ucla.edu
ee.ucla.edu = J
cs.ucla.edu = K

cs.ucla.edu
lever.cs.ucla.edu = R
www.cs.ucla.edu = S

www.cs.ucla.edu

AND I know where ucla.edu is!

AND I know where ucla.edu is!

Web Server

CS 118
Winter 2016
Iterative DNS

. (ROOT)
  .edu = X
  .com = Y
  .de = Z

.edu
  usc.edu = P
  ucla.edu = Q

ucla.edu
  ee.ucla.edu = J
  cs.ucla.edu = K

cs.ucla.edu
  lever.cs.ucla.edu = R
  www.cs.ucla.edu = S

www.cs.ucla.edu

I even know where cs.ucla.edu is!
Iterative DNS

. (ROOT)
.edu = X
.com = Y
.de = Z

.edu
usc.edu = P
ucla.edu = Q

cs.ucla.edu
lever.cs.ucla.edu = R
www.cs.ucla.edu = S

cs.ucla.edu

ee.ucla.edu = J
cs.ucla.edu = K

www.cs.ucla.edu

Actually, I know where www.cs.ucla.edu is!

Web Server
Iterative DNS

. (ROOT)
.edu = X
.com = Y
.de = Z

.edu
usc.edu = P
ucla.edu = Q

ucla.edu
ee.ucla.edu = J
cs.ucla.edu = K

cs.ucla.edu
lever.cs.ucla.edu = R
www.cs.ucla.edu = S

www.cs.ucla.edu

Web Server
Recursive DNS

. (ROOT)
.edu = X
.com = Y
.de = Z

.edu
usc.edu = P
ucla.edu = Q

ucla.edu
ee.ucla.edu = J
cs.ucla.edu = K

cs.ucla.edu
lever.cs.ucla.edu = R
www.cs.ucla.edu = S

www.cs.ucla.edu

Web Server
Iterative vs. recursive

• Iterative
  – Pros
    • Client retains control
    • Responses can be client-customized
    • Local caching
  – Cons
    • Many packets
    • Long delays

• Recursive
  – Pros
    • More efficient
    • Faster
    • Aggregate caching
  – Cons
    • Client hands over control
    • Responses can’t be customized
Other DNS uses

• Load distribution
  – One name ➔ many addresses
  – Respond with entire list, rotated “round robin”
  – NB: not load BALANCING; no feedback used

• Localization
  – Infer topological/geographic location
  – Use location to prioritize or mask responses

• Fault tolerance
  – List of backups/alternates
DNS pros and cons

• Pros
  – Local control over subsets of the name space
  – Similar names end up in one place

• Cons
  – Need to start at root creates a vulnerability
  – Depends on servers in many places (no fate sharing)
  – Caching and replication consistency issues
  – Similar names end up in one place
Search Approaches

• Unstructured
  – Ask neighbors or anyone else who might know
  – Some peer file sharing services used this

• Structured
  – Use information about query to start search in a good spot
  – Someone who has the translation
  – Or knows who will
  – Distributed hash tables, e.g.
Broadcast

• Undirected message

• Flooded to all
  – Or a preconfigured set, e.g., “all routers”

• The ultimate in unstructured search
ARP

• 128.9.160.161 ➔ 128.9.160.7
  – .161 shouts “WHO HAS 128.9.160.7?”
  – .7 responds “.7 is 0F:EE:…”

• 128.9.160.161 ➔ 201.35.6.2
  – .161 shouts “WHO HAS 201.35.6.2?”
  – .7 responds “.7 is 0F:EE:…”
  – .7 lies
Broadcast pros and cons

• **Pros**
  – Easy to configure
  – Fault tolerant (reaches everyone)

• **Cons**
  – Doesn’t scale
  – Can cause high load ("chatter")
Distributed Hash Table (DHT)

- Structured search
- Using a hierarchy (like the DNS)
  - Scrambled using a hash

![Diagram of Distributed Hash Table]

Data

- Fox
- The red fox runs across the ice
- The red fox walks across the ice

Hash function

Key

- DFCD3454
- 52ED879E
- 46042841

Distributed Network

Peers
DHT variants

• Many variants
  – Different structure to name space and thus server interconnections
  – Some are “regular” (ring, hypercube), others are varieties of trees
  – Most include replication and fault-tolerance

• Widely used
  – Bittorrent file sharing
  – Freenet anonymous networking
DHT pros and cons

• Pros
  – Fault tolerant
  – Scalable
  – Load distributing
  – Hash avoids locality of similar names
  • There are locality-preserving hashes, but that somewhat defeats the point of a DHT

• Cons
  – Complexity
  – Hash avoids locality of similar names
Caching

• Many name translation operations are expensive
• So avoid duplication of them
• Exploit “locality of reference”
  – Endpoint repeating its own requests
  – Endpoints repeating each others’ requests
• Keep a copy
  – At the original endpoint
  – At any relay
Where is caching used?

• In the endpoint
  – ARP, i.e., IP->ethernet
  – Web responses

• At a server, to aggregate requests
  – DNS server (home router, on a LAN)
  – Web proxy cache (home router sometimes)

• At the edge of an enterprise (organization)
  – DNS
Cache timeouts and performance

• Timeout issues
  – How long to keep a copy
  – What to do when the copy expires?
    • Ask for another one (proactive)?
    • Just delete the stale entry (reactive)?

• Performance
  – Longer can reduce load, improve response
  – Too long will ignore changes (delete/rename)
Cache invalidation

• Various events cause an entry to expire
  – Timeout
    • Time since info was first retrieved (DNS)
    • Time since last request (ARP)
  – Conflicting information
    • If requests are broadcast, responses can too
    • Check to see if responses match (ARP)
Summary

• Different names may exist for the same thing at different layers
• Names often need to be resolved from one form to another
• Resolution requires some ability to get hold of translation information
  – Which can be done in many ways
• Caching is useful to achieve good name resolution performance