# Discovery

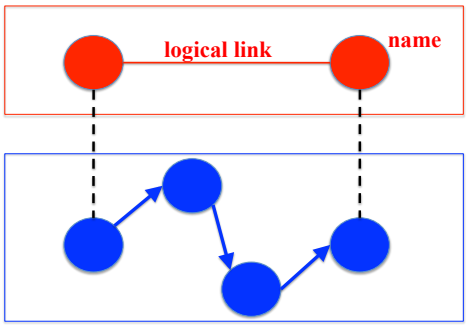
Mike Freedman

COS 461: Computer Networks

Lectures: MW 10-10:50am in Architecture N101

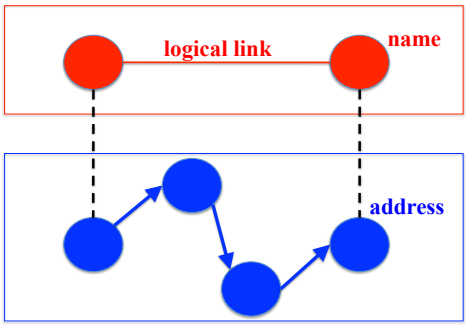
http://www.cs.princeton.edu/courses/archive/spr13/cos461/

## Relationship Between Layers



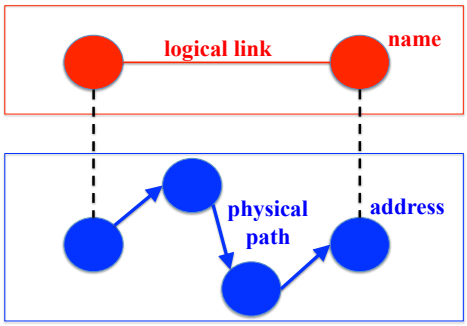

2

## Discovery: Mapping Name to Address




3

## Routing: Mapping Link to Path




4

## Naming

5

## What's in a Name?

- Human readable?
  - If users interact with the names
- Fixed length?
  - If equipment processes at high speed
- Large name space?
  - If many nodes need unique names
- Hierarchical names?
  - If the system is very large and/or federated
- Self-certifying?
  - If preventing "spoofing" is important

6

## Different Kinds of Names

- **Host name** (e.g., www.cs.princeton.edu)
  - Mnemonic, variable-length, appreciated *by humans*
  - Hierarchical, based on organizations
- **IP address** (e.g., 128.112.7.156)
  - Numerical 32-bit address appreciated *by routers*
  - Hierarchical, based on organizations and topology
- **MAC address** (e.g., 00-15-C5-49-04-A9)
  - Numerical 48-bit address appreciated *by adapters*
  - Non-hierarchical, unrelated to network topology

7

## Hierarchical Assignment Processes

- **Host name:** www.cs.princeton.edu
  - Domain: registrar for each top-level domain (eg, .edu)
  - Host name: local administrator assigns to each host
- **IP addresses:** 128.112.7.156
  - Prefixes: ICANN, regional Internet registries, and ISPs
  - Hosts: static configuration, or dynamic using DHCP
- **MAC addresses:** 00-15-C5-49-04-A9
  - Blocks: assigned to vendors by the IEEE
  - Adapters: assigned by the vendor from its block

8

## Host Names vs. IP Addresses

- Names easier (for us!) to remember
- IP addresses can change underneath
  - E.g., renumbering when changing providers
- Name could map to multiple IP addresses
  - www.cnn.com to multiple replicas of the Web site
- Map to different addresses in different places
  - E.g., to reduce latency, or return different content
- Multiple names for the same address
  - E.g., aliases like ee.mit.edu and cs.mit.edu

9

## IP vs. MAC Addresses

- LANs designed for arbitrary network protocols
  - Not just for IPv4 (e.g., IPvX, Appletalk, X.25, …)
  - Different LANs may have different addressing schemes
- A host may move to a new location
  - So, cannot simply assign a static IP address
  - Instead, must reconfigure the adapter
- Must identify the adapter during bootstrap
  - Need to talk to the adapter to assign it an IP address

10

## Questions

- Which allocations follow network topology?
- Which allocations follow organizational structure?

(A) Domain names
(B) IPs
(C) MACs
(D) Domains and IPs
(E) All of above

11

## Discovery

12

## Directories

- A key-value store
  - Key: name; value: address(es)
  - Answer queries: given name, return address(es)
- Caching the response
  - Reuse the response, for a period of time
  - Better performance and lower overhead
- Allow entries to change
  - Updating the address(es) associated with a name
  - Invalidating or expiring cached responses

13

## Directory Design: Three Extremes

- Flood the query (e.g., ARP)
  - The named node responds with its address
  - But, high overhead in large networks
- Push data to all clients (/etc/hosts)
  - All nodes store a full copy of the directory
  - But, high overhead for many names and updates
- Central directory server
  - All data and queries handled by one machine
  - But, poor performance, scalability, and reliability

14

## Directory Design: Distributed Solutions

- Hierarchical directory (e.g., DNS)
  - Follow the hierarchy in the name space
  - Distribute the directory, distribute the queries
  - Enable decentralized updates to the directory
- Distributed Hash Table (e.g. P2P applications)
  - Directory as a hash table with flat names
  - Each directory node handles range of hash outputs
  - Use hash to direct query to the directory node

15

## Domain Name System (DNS)

Computer science concepts underlying DNS

- Indirection: names in place of addresses
- Hierarchy: in names, addresses, and servers
- Caching: of mappings from names to/from addresses

16

## Strawman Solution #1: Local File

- Original name to address mapping
  - Flat namespace
  - /etc/hosts
  - SRI kept main copy
  - Downloaded regularly
- Count of hosts was increasing: moving from a machine per domain to machine per user
  - Many more downloads
  - Many more updates

17

## Strawman Solution #2: Central Server

- Central server
  - One place where all mappings are stored
  - All queries go to the central server
- Many practical problems
  - Single point of failure
  - High traffic volume
  - Distant centralized database
  - Single point of update
  - Does not scale

**Need a distributed, hierarchical collection of servers**

18

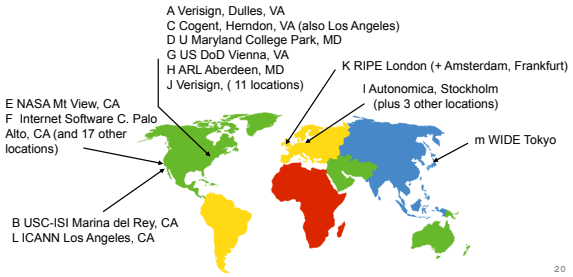
## Domain Name System (DNS)

- Properties of DNS
  - Hierarchical name space divided into zones
  - Distributed over a collection of DNS servers
- Hierarchy of DNS servers
  - Root servers
  - Top-level domain (TLD) servers
  - Authoritative DNS servers
- Performing the translations
  - Local DNS servers and client resolvers



## DNS Root Servers

- 13 root servers (see http://www.root-servers.org/)
- Labeled A through M






## Reliability

- DNS servers are replicated
  - Name service available if at least one replica is up
  - Queries can be load balanced between replicas
- UDP used for queries
  - Need reliability: must implement this on top of UDP
- Try alternate servers on timeout
  - Exponential backoff when retrying same server
- Same identifier for all queries
  - Don't care which server responds

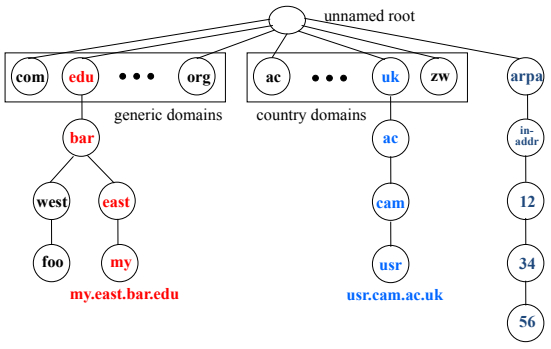


## TLD and Authoritative DNS Servers

- Global Top-level domain (gTLD) servers
  - Generic domains (e.g., .com, .org, .edu)
  - Country domains (e.g., .uk, .fr, .ca, .jp)
  - Managed professionally (e.g., Verisign for .com .net)
- Authoritative DNS servers
  - Provide public records for hosts at an organization
  - For the organization's servers (e.g., Web and mail)
  - Can be maintained locally or by a service provider



## Distributed Hierarchical Database






## DNS Queries and Caching

## Using DNS

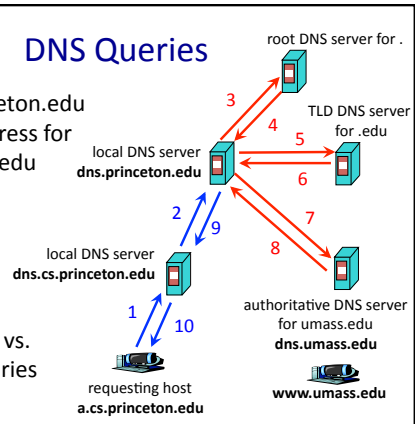
- Local DNS server ("default name server")
  - Usually near the end hosts who use it
  - Local hosts configured with local server (e.g., /etc/resolv.conf) or learn the server via DHCP
- Client application
  - Extract server name (e.g., from the URL)
  - Do *gethostbyname()* or *getaddrinfo()* to get address
- Server application
  - Extract client IP address from socket
  - Optional *gethostbyaddr()* to translate into name



## DNS Queries

Host a.cs.princeton.edu wants IP address for www.umass.edu

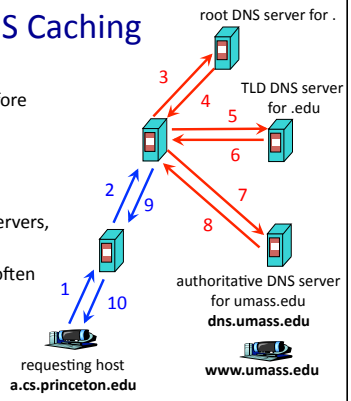
Note Recursive vs. Iterative Queries






## DNS Caching

- DNS query latency
  - E.g., 1 sec latency before starting a download
- Caching to reduce overhead and delay
  - Small # of top-level servers, that change rarely
  - Popular sites visited often
- Where to cache?
  - Local DNS server
  - Browser






## DNS Cache Consistency

- Goal: Ensuring cached data is up to date
- DNS design considerations
  - Cached data is "read only"
  - Explicit invalidation would be expensive
    - Server would need to keep track of all resolvers caching
- Avoiding stale information
  - Responses include a "time to live" (TTL) field
  - Delete the cached entry after TTL expires
- Perform negative caching (for dead links, misspellings)
  - So failures quick and don't overload gTLD servers



## Setting the Time To Live (TTL)

- TTL trade-offs
  - Small TTL: fast response to change
  - Large TTL: higher cache hit rate
- Following the hierarchy
  - Top of the hierarchy: days or weeks
  - Bottom of the hierarchy: seconds to hours
- Tension in practice
  - CDNs set low TTLs for load balancing and failover
  - Browsers cache for 15-60 seconds



## Questions

- Tension:
  - DNS operators want high TTL for low load on DNS servers,
  - Domains want low TTL for faster failover b/w IP addrs

  (A) True (B) False
- By returning IP addresses in "round robin" fashion, DNS operators can ensure equal load better servers

  (A) True (B) False
- Most applications obey TTLs on DNS records

  (A) True (B) False

## Questions

- Tension:
  - DNS operators want high TTL for low load on DNS servers,
  - Domains want low TTL for faster failover b/w IP addrs

  (A) True (B) False
- By returning IP addresses in "round robin" fashion, DNS operators can ensure equal load better servers

  (A) True (B) False
- Most applications obey TTLs on DNS records

  (A) True (B) False

31

## DNS Resource Records

RR format: `(name, value, type, ttl)`

- Type=A
  - **Name**: hostname
  - **Value**: IP address
- Type=NS
  - **Name**: domain
  - **Value**: hostname of name server for domain
- Type=CNAME
  - **Name**: alias for some "canonical" (the real) name: www.ibm.com is really srveast.backup2.ibm.com
  - **Value**: canonical name
- Type=MX
  - **Value**: name of mailserver associated with **name**

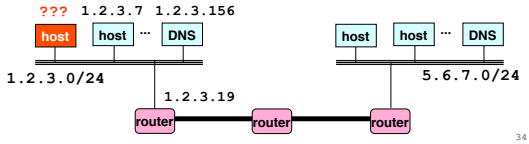
32

## Learning Your Local DNS Server

33

## How To Bootstrap an End Host?

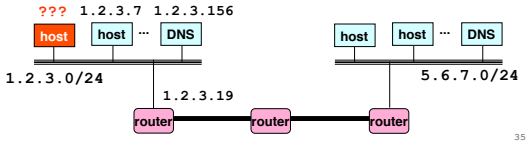
- What local DNS server to use?
- What IP address the host should use?
- How to send packets to remote destinations?
- How to ensure incoming packets arrive?




34

## Avoiding Manual Configuration

- Dynamic Host Configuration Protocol (DHCP)
  - End host learns how to send packets
  - Learn IP address, DNS servers, and gateway
- Address Resolution Protocol (ARP)
  - Others learn how to send packets to the end host
  - Learn mapping between IP address & interface address




35

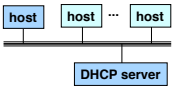
## Key Ideas in Both Protocols

- **Broadcasting**: when in doubt, shout!
  - Broadcast query to all hosts in local-area-network
- **Caching**: remember the past for a while
  - Store the information you learn to reduce overhead
  - Remember your address & other host's addresses
- **Soft state**: … but eventually forget the past
  - Associate a time-to-live field with the information
  - … and either refresh or discard the information
  - Key for robustness in face of unpredictable change

36

## Bootstrapping Problem

- Host doesn't have an IP address yet
  - So, host doesn't know what source to use
- Host doesn't know who to ask for an IP address
  - So, host doesn't know what destination to use
- Solution: discover a server who can help
  - Broadcast a DHCP server-discovery message
  - Server sends a DHCP "offer" offering an address




37

## Dynamic Host Configuration Protocol

arriving client

DHCP discover (broadcast)

DHCP server 192.168.1.1

DHCP offer

One or more servers return:
- Config params (proposed IP addr, net mask, gateway, DNS server, …)
- Lease time (validity interval)

- Client echoes selected parameters

DHCP request (broadcast)

DHCP ACK

- Chosen DHCP server confirms
- Other servers see not chosen

38

## Deciding What IP Address to Offer

- Static allocation
  - Servers have dedicated IP for each MAC address
  - Makes it easy to track a host over time
- Dynamic allocation
  - Servers maintain address pool and assign on demand
  - More efficient use of (limited) set of addresses
- Soft-state assignments
  - Client can release explicitly or leave/crash
  - Tradeoff: inactive addresses vs. frequent renewals

39

## Questions

- When should client start using allocated address?
  - (A) After it receives the first DHCP Offer
  - (B) After it selects one to use following one or more Offers
  - (C) After it receives a DHCP ACK from the server
- DHCP servers require a special coordination protocol to maintain their address pool's consistency

  (A) True (B) False

40

## Questions

- When should client start using allocated address?
  - (A) After it receives the first DHCP Offer
  - (B) After it selects one to use following one or more Offers
  - (C) After it receives a DHCP ACK from the server
- DHCP servers require a special coordination protocol to maintain their address pool's consistency

  (A) True (B) False

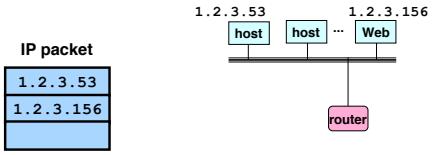
41

## So, Now the Host Knows Things

- IP address
- Mask
- Gateway router
- DNS server
- And can send packets to other IP addresses
  - How to learn the MAC address of the destination?

42

## Sending Packets Over a Link




- Adapters only understand MAC addresses
  - Translate the destination IP address to MAC address
  - Encapsulate the IP packet inside a link-level frame

43

## Address Resolution Protocol Table

- Every node maintains an ARP table
  - (IP address, MAC address) pair
- Consult the table when sending a packet
  - Map destination IP to destination MAC address
  - Encapsulate and transmit the data packet
- But, what if the IP address is not in the table?
  - Sender broadcasts: "Who has IP address 1.2.3.156?"
  - Receiver responds: "MAC address 58-23-D7-FA-20-B0"
  - Sender caches the result in its ARP table

44

## Conclusion

- Discovery
  - Mapping a name at the upper layer
  - … to an address at the lower layer
- Domain Name System (DNS)
  - Hierarchical names, hierarchical directory
  - Query-response protocol with caching
  - Time-To-Live to expire stale cached responses
- Next time: routing

45

## Backup Slides

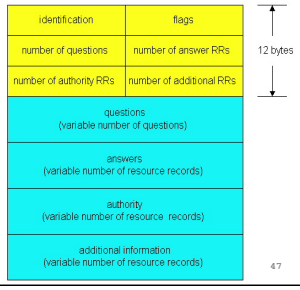
46

## DNS Protocol

DNS protocol : *query* and *reply* msg, both with same *msg format*

Message header

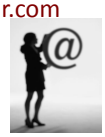
- Identification: 16 bit # for query, reply to query uses same #
- Flags:
  - Query or reply
  - Recursion desired
  - Recursion available
  - Reply is authoritative




47

## Inserting Resource Records into DNS

- Example: just created startup "FooBar"
- Register foobar.com at Network Solutions
  - Provide registrar with names and IP addresses of your authoritative name server (primary and secondary)
  - Registrar inserts two RRs into the com TLD server:
    - (foobar.com, dns1.foobar.com, NS)
    - (dns1.foobar.com, 212.212.212.1, A)
- Put in authoritative server dns1.foobar.com
  - Type A record for www.foobar.com
  - Type MX record for foobar.com
- Play with "dig" on UNIX



```
$ dig nytimes.com ANY
; QUESTION SECTION:
;nytimes.com.           IN  ANY

;; ANSWER SECTION:
nytimes.com.        267 IN  MX  100 NYTIMES.COM.S7A1.PSMTP.com.
nytimes.com.        267 IN  MX  200 NYTIMES.COM.S7A2.PSMTP.com.
nytimes.com.        267 IN  A   199.239.137.200
nytimes.com.        267 IN  A   199.239.136.200
nytimes.com.        267 IN  TXT "v=spf1 mx ptr ip4:199.239.138.0/24
   include:alerts.wallst.com include:authsmtp.com ~all"
nytimes.com.        267 IN  SOA nslt.nytimes.com. root.nslt.nytimes.com.
   2009070102 1800 3600 604800 3600
nytimes.com.        267 IN  NS  nydns2.about.com.
nytimes.com.        267 IN  NS  nslt.nytimes.com.
nytimes.com.        267 IN  NS  nydns1.about.com.

;; AUTHORITY SECTION:
nytimes.com.        267 IN  NS  nydns1.about.com.
nytimes.com.        267 IN  NS  nslt.nytimes.com.
nytimes.com.        267 IN  NS  nydns2.about.com.

;; ADDITIONAL SECTION:
nydns1.about.com.   86207   IN  A   207.241.145.24
nydns2.about.com.   86207   IN  A   207.241.145.25
```



```
$ dig nytimes.com +norec @a.root-servers.net

;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 53675
;; flags: qr; QUERY: 1, ANSWER: 0, AUTHORITY: 13, ADDITIONAL: 14

;; QUESTION SECTION:
;nytimes.com.                   IN      A

;; AUTHORITY SECTION:
com.                    172800  IN      NS      K.GTLD-SERVERS.NET.
com.                    172800  IN      NS      E.GTLD-SERVERS.NET.
com.                    172800  IN      NS      D.GTLD-SERVERS.NET.
com.                    172800  IN      NS      I.GTLD-SERVERS.NET.
com.                    172800  IN      NS      C.GTLD-SERVERS.NET.

;; ADDITIONAL SECTION:
A.GTLD-SERVERS.NET.     172800  IN      A       192.5.6.30
A.GTLD-SERVERS.NET.     172800  IN      AAAA    2001:503:a83e::2:30
B.GTLD-SERVERS.NET.     172800  IN      A       192.33.14.30
B.GTLD-SERVERS.NET.     172800  IN      AAAA    2001:503:231d::2:30
C.GTLD-SERVERS.NET.     172800  IN      A       192.26.92.30
D.GTLD-SERVERS.NET.     172800  IN      A       192.31.80.30
E.GTLD-SERVERS.NET.     172800  IN      A       192.12.94.30

;; Query time: 76 msec
;; SERVER: 198.41.0.4#53(198.41.0.4)
;; WHEN: Mon Feb 23 11:24:06 2009
;; MSG SIZE  rcvd: 501
```



```
$ dig nytimes.com +norec @k.gtld-servers.net

;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 38385
;; flags: qr; QUERY: 1, ANSWER: 0, AUTHORITY: 3, ADDITIONAL: 3

;; QUESTION SECTION:
;nytimes.com.                   IN      A

;; AUTHORITY SECTION:
nytimes.com.            172800  IN      NS      nslt.nytimes.com.
nytimes.com.            172800  IN      NS      nydns1.about.com.
nytimes.com.            172800  IN      NS      nydns2.about.com.

;; ADDITIONAL SECTION:
nslt.nytimes.com.       172800  IN      A       199.239.137.15
nydns1.about.com.       172800  IN      A       207.241.145.24
nydns2.about.com.       172800  IN      A       207.241.145.25

;; Query time: 103 msec
;; SERVER: 192.52.178.30#53(192.52.178.30)
;; WHEN: Mon Feb 23 11:24:59 2009
;; MSG SIZE  rcvd: 144
```



```
$ dig nytimes.com ANY +norec @nslt.nytimes.com

;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 39107
;; flags: qr aa; QUERY: 1, ANSWER: 13, AUTHORITY: 0, ADDITIONAL: 1

;; QUESTION SECTION:
;nytimes.com.                   IN      ANY

;; ANSWER SECTION:
nytimes.com.        300     IN      SOA     nslt.nytimes.com.
        root.nslt.nytimes.com. 2009070102 1800 3600 604800 3600
nytimes.com.        300     IN      MX      200 NYTIMES.COM.S7A2.PSMTP.com.
nytimes.com.        300     IN      MX      100 NYTIMES.COM.S7A1.PSMTP.com.
nytimes.com.        300     IN      NS      nslt.nytimes.com.
nytimes.com.        300     IN      NS      nydns1.about.com.
nytimes.com.        300     IN      NS      nydns2.about.com.
nytimes.com.        300     IN      A       199.239.137.245
nytimes.com.        300     IN      A       199.239.136.200
nytimes.com.        300     IN      A       199.239.136.245
nytimes.com.        300     IN      TXT     "v=spf1 mx ptr ip4:199.239.138.0/24
       include:alerts.wallst.com include:authsmtp.com ~all"

;; ADDITIONAL SECTION:
nslt.nytimes.com.       300     IN      A       199.239.137.15

;; Query time: 10 msec
;; SERVER: 199.239.137.15#53(199.239.137.15)
;; WHEN: Mon Feb 23 11:25:20 2009
;; MSG SIZE  rcvd: 454
```



# DNS security

- DNS cache poisoning
  - Ask for www.evil.com
  - Additional section for (www.cnn.com, 1.2.3.4, A)
  - Thanks! I won't bother check what I asked for
- DNS hijacking
  - Let's remember the domain. And the UDP ID.
  - 16 bits: 65K possible IDs
    - What rate to enumerate all in 1 sec? 64B/packet
    - 64*65536*8 / 1024 / 1024 = 32 Mbps
  - Prevention: Also randomize the DNS source port
    - E.g., Windows DNS alloc's 2500 DNS ports: ~164M possible IDs
    - Would require 80 Gbps
    - Kaminsky attack: this source port…wasn't random after all