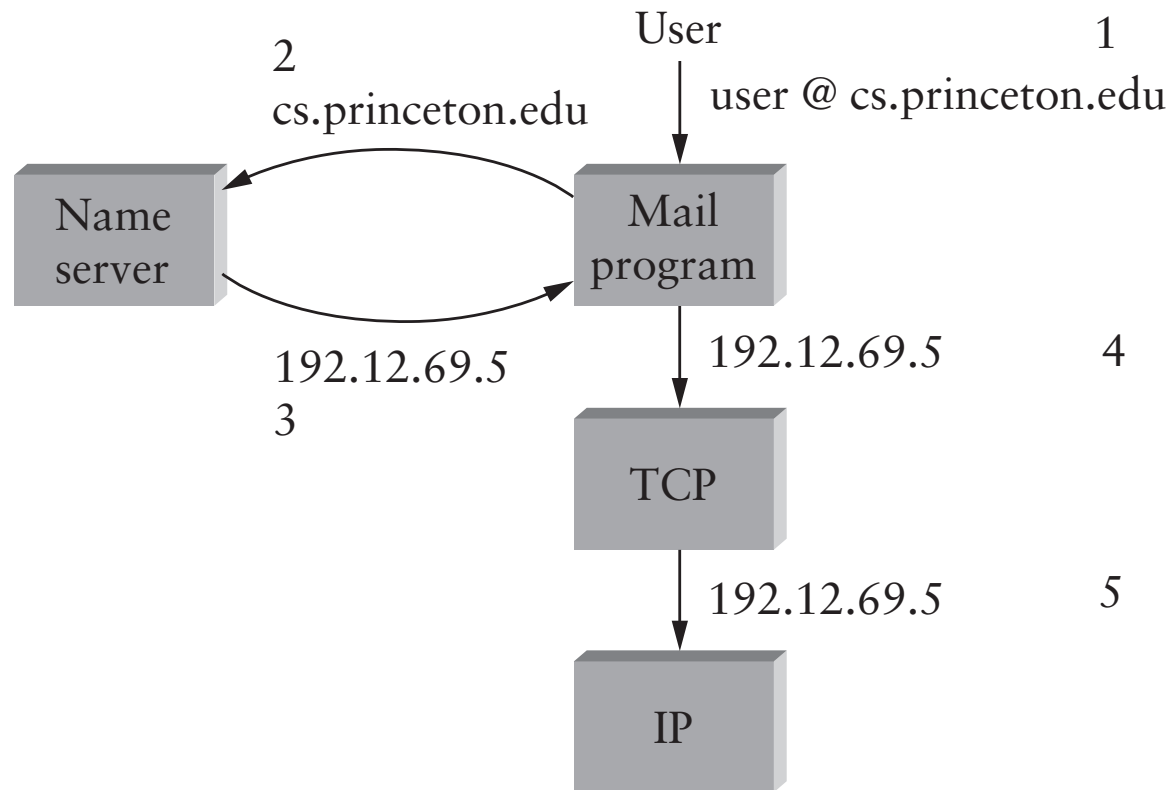


# Motivation



- **Users can't (shouldn't) remember IP addresses**
  - Need to map names (`www.brown.edu` → `128.148.128.180`)
- **Implemented by library functions & servers**
  - `gethostbyname()` talks to servers over UDP

## hosts.txt system

- **Originally, hosts were listed in a file, hosts.txt**
  - Email global network administrator when you add a host
  - Administrator mails out new hosts.txt file every few days
- **Would be completely impractical today**
  - hosts.txt today would be huge (gigabytes)
  - What if two people wanted to add same name?
  - Who is authorized to change address of a name?
  - People need to change name mappings more often than every few days (*e.g.*, Dynamic IP addresses)

# Goals of an Internet-scale naming system

- **Scalability**

- Must handle huge number of records
- Potentially *exponential* in name size—because custom software may synthesize names on-the-fly

- **Distributed control**

- Let people control their own names

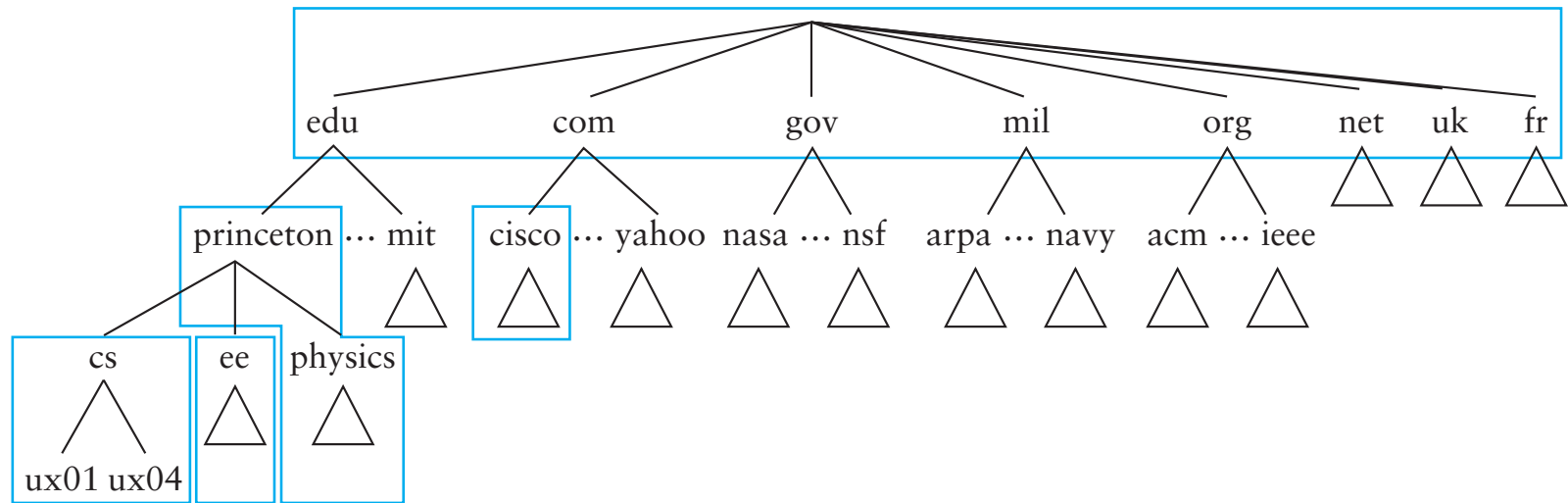
- **Fault-tolerance**

- Old software assumed `hosts.txt` always there
- Bad potential failure modes when name lookups fail
- Minimize lookup failures in the face of other network problems

# The good news

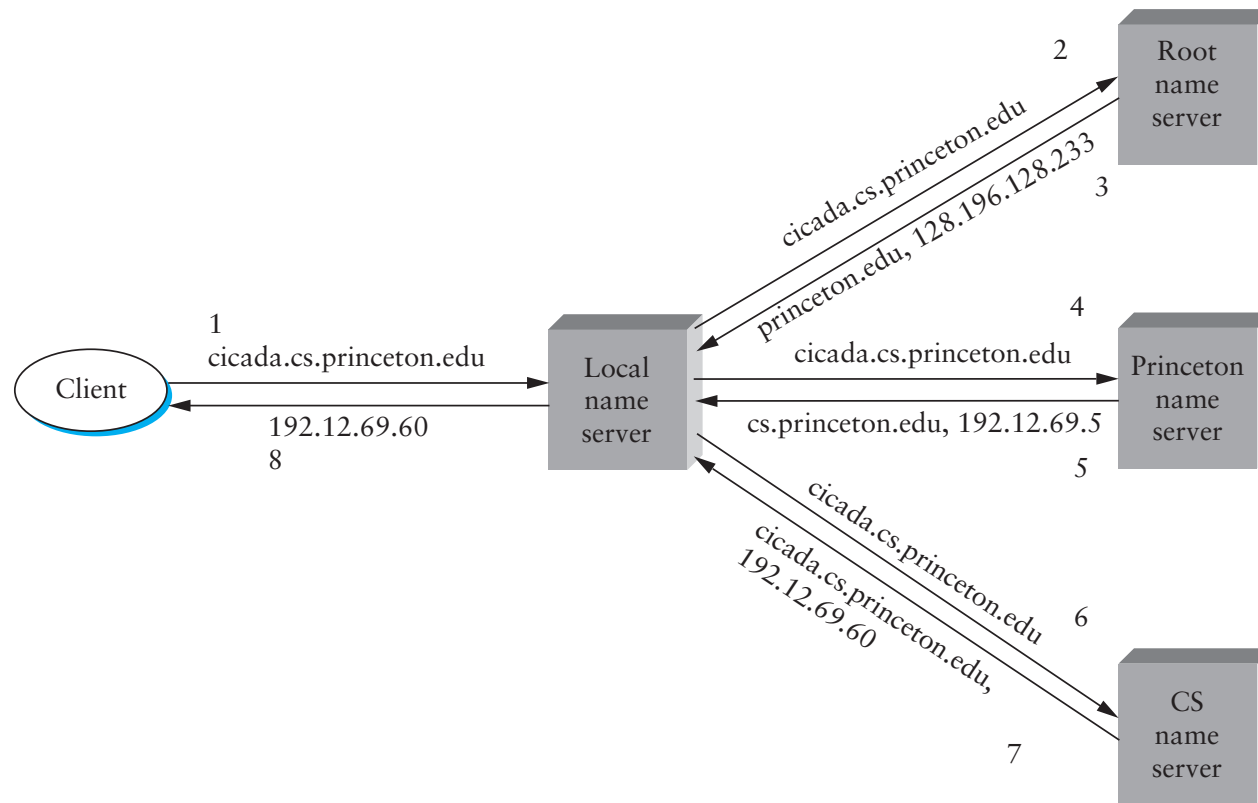
- **Properties that make these goals easier to achieve:**
  1. **Read-only or read-mostly database**
    - People typically look up hostnames much more often than they are updated
  2. **Loose consistency**
    - When adding a machine, may be okay if info takes minutes or hours to propagate
- **These suggest aggressive caching**
  - Once you have looked up hostname, remember result
  - Don't need to look it up again in near future

# Domain Name System (DNS)



- **Break namespace into a bunch of zones**
  - root (.), edu., brown.edu., cs.brown.edu.,  
www.cs.brown.edu., ...
  - Zones separately administered  $\implies$  **delegation**
  - Parent zones tell you how to find servers for subdomains.
- **Each zone served from multiple replicated servers**

# DNS software architecture



- Apps make **recursive** queries to local DNS server
- Local server queries remote servers **iteratively**
  - Aggressively caches result
  - *E.g.*, only contact root on first query ending `.edu`

# DNS protocol

- **TCP/UDP port 53**
- **Most traffic uses UDP**
  - Lightweight protocol has 512 byte UDP message limit
  - retry using TCP if UDP fails (*e.g.*, reply truncated)
- **TCP requires message boundaries**
  - Prefix all messages with 16-bit length
- **Bit in query determines if query is recursive**

# Resource records

- All DNS info represented as resource records (RR):

*name* [TTL] [*class*] *type* *rdata*

- *name* – domain name (e.g., `www.brown.edu`)
  - **TTL** – time to live (seconds)
  - *class* – for extensibility, usually IN (1) “Internet”
  - *type* – type of the record
  - *rdata* – resource data dependent on the *type*
- Two important DNS RR types:
    - **A** – Internet address (IPv4)
    - **NS** – name server
  - Example resource records (`dig www.cs.brown.edu`):

```
www.brown.edu.    600   IN    A     128.148.128.180
brown.edu.        21600 IN    NS    knot.brown.edu.
brown.edu.        21600 IN    NS    ns1.ucsb.edu.
```

# Some implementation details

- **How do local name servers know root servers?**

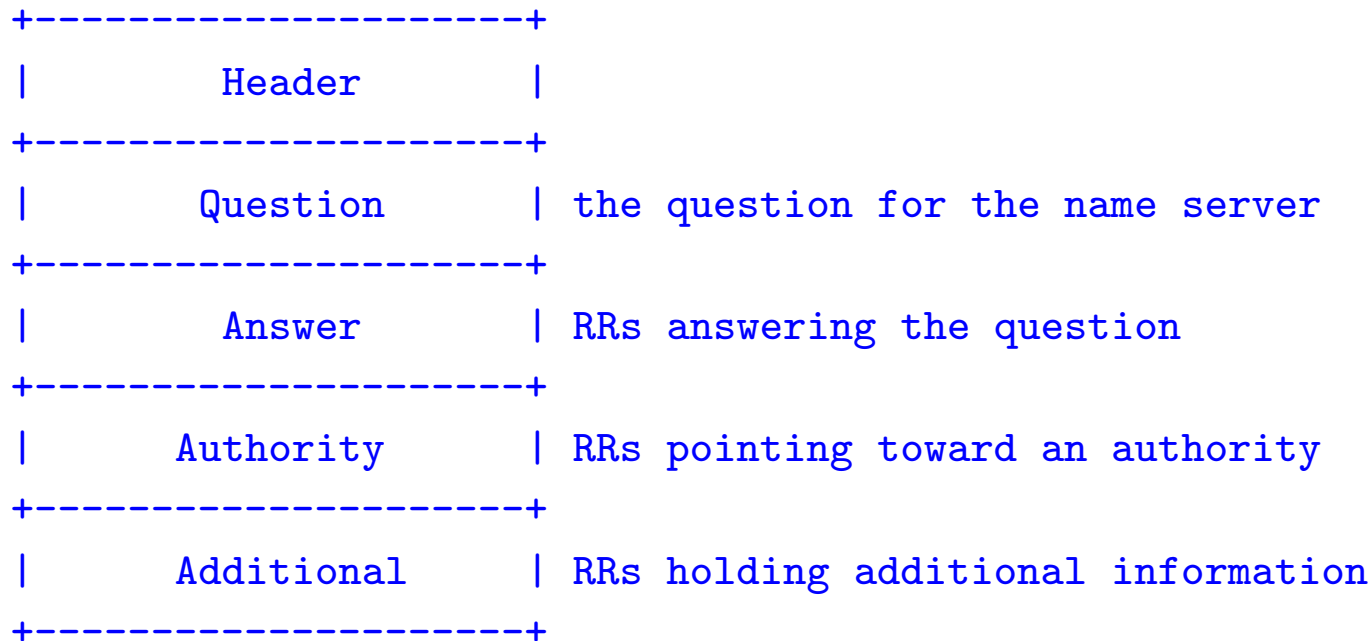
- Need to configure name server with *root cache* file
- Contains root name servers and their addresses

```
.                3600000  NS  A.ROOT-SERVERS.NET.  
A.ROOT-SERVERS.NET. 3600000  A   198.41.0.4  
.                3600000  NS  B.ROOT-SERVERS.NET.  
B.ROOT-SERVERS.NET. 3600000  A   128.9.0.107  
...
```

- **How do you get addresses of other name servers**

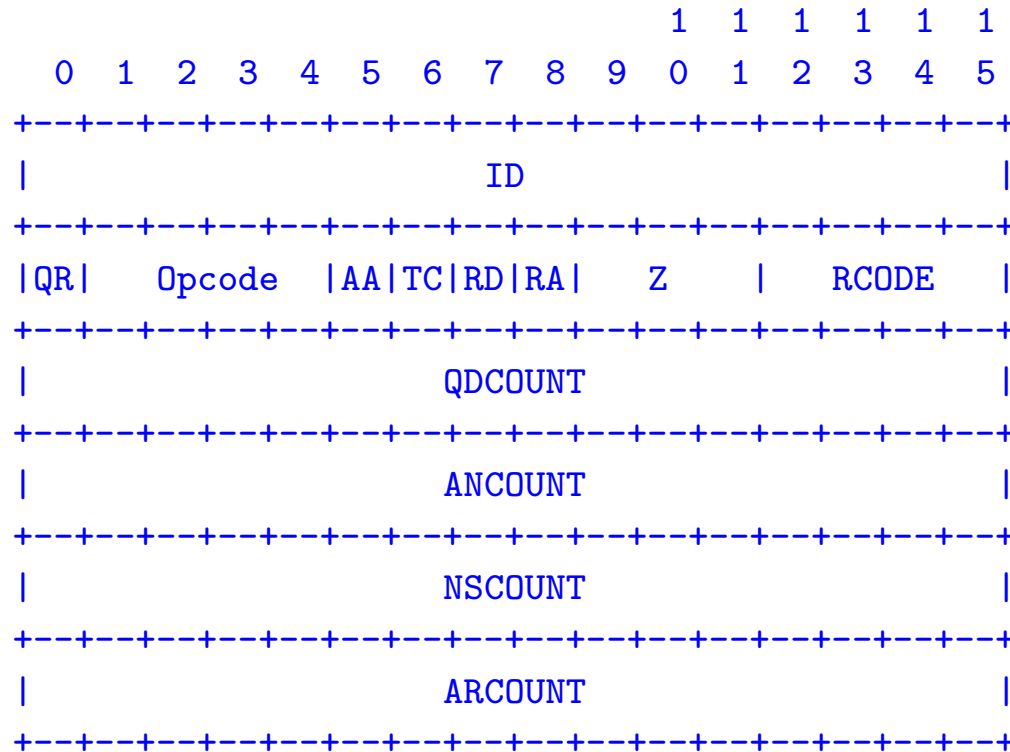
- To lookup names ending `.brown.edu.`, ask `knot.brown.edu.`
- But how to get `knot.brown.edu.`'s address?
- Solution: **glue** records – A records in parent zone
- Name servers for `edu.` have A record of `knot.brown.edu.`
- Check using `dig +norec www.brown.edu @A3.NSTLD.COM`

# Structure of a DNS message



- **Same message format for queries and replies**
  - Query has zero RRs in Answer/Authority/Additional sections
  - Reply includes question, plus has RRs
- **Authority allows for delegation**
- **Additional for glue + other RRs client might need**

# Header format



- **QR** – 0=query, 1=response
- **RCODE** – error code
- **AA**=authoritative answer, **TC**=truncated,  
**RD**=recursion desired, **RA**=recursion available



# Encoding of domain names

- **A DNS name consists of a series of labels**
  - `www.brown.edu` has three labels, `www`, `brown`, & `edu`
  - Labels can contain letters, digits, and “-”, but should not start or end with “-”
  - Maximum length 63 characters
  - Encoded as length byte followed by label
  - Last label always empty label
  - `www.cs.brown.edu` → `[3]www[2]cs[5]brown[3]edu[0]`
- **Names are case insensitive**
  - But server must preserve case of question in replies
  - `dig wWw.br0wn.eDu`, look at answer & authority

# Name compression

- **Observation: many common suffixes in DNS messages**
  - Particularly because of case preservation rule
- **Allow pointer labels to re-use suffixes**
  - Recall label starts with length byte (0-63)
  - If value  $\geq 0xc0$  (192), subtract  $0xc000$  from first *two* bytes, and treat as pointer into message
  - edge.cs.brown.edu  $\rightarrow$  [4]edge[192][5]
- **Example: Using root-servers.net allows more root NS records to fit in one UDP message**

# Secondary servers

- **Availability requires geographically disparate replicas**
  - *e.g.*, Brown & UCSB serve each others' names.
- **Typical setup: One master many slave servers**
- **How often to sync up servers? Trade-off**
  - All the time  $\implies$  high overhead
  - Rarely  $\implies$  stale data
- **Put trade-off under domain owner's control**
  - Fields in SOA record control secondary's behavior
  - Primary can change SOA without asking human operator of secondary

# The SOA record

- Every delegated zone has one SOA record

*name [TTL] [IN] SOA mname rname*

*serial refresh retry expire minimum*

- *name* – Name of zone (*e.g.*, brown.edu)
- *mname* – DNS name of main name server
- *rname* – E-mail address of contact (@→.)
- *serial* – Increases each time zone changes
- *refresh* – How often secondary servers should sync
- *retry* – How soon to re-try sync after a failure
- *expire* – When to discard data after repeated failures
- *minimum* – How long to cache negative results

# MX records

- For historical reasons, mail does not have to use A records directly

- Example: You can send mail to `jj@edge.cs.brown.edu`
- Edge does *not* run a mail server (and it's firewalled).

- Mail is using MX (Mail eXchanger) records:

*name [TTL] [IN] MX prio server*

- *name* is part after "@" in email addr
- *prio* is numeric priority (lower # = higher priority)
- *server* is server hostname, for which client needs A record(s)
- More additional records: return A records with MX records

- Examples:

```
csail.mit.edu.      14400   IN      MX      1  incoming.csail.mit.edu.
csail.mit.edu.      14400   IN      MX      2  fedex.ai.mit.edu.
```

# MX record details

- **Client contacts servers in order of priority**
  - If connection fails, try next server
  - Two MXes can have same priority, sort consistently to avoid mail loops
- **Can be used to collect mail while server is down**
  - Secondary server spools mail while primary is down
  - Flush queue on secondary once primary is back up
  - Historically, any mail server would act as backup for any other—major cause of spam
- **Can also be used for extreme load balancing**
  - AOL has more servers than fit in 512-byte UDP packet
  - Solution: Many MX records, w. many A records each

# CNAME records

- CNAME record specifies an alias:

*name* [TTL] [IN] CNAME *canonical-name*

- As if any RR's associated w. *canonical-name* also for *name*
- Reflected in h\_aliases field of hostent

- Examples, to save typing:

```
wb.scs.cs.nyu.edu. CNAME williamsburg-bridge.scs.cs.nyu.edu.
```

```
mb.scs.cs.nyu.edu. CNAME manhattan-bridge.scs.cs.nyu.edu.
```

- CNAME precludes any other RRs for name

```
jannotti.com. CNAME jj.cs.brown.edu.
```

```
jannotti.com. NS ILLEGAL!
```

- Note answer section can have CNAME for query name + other RR(s) for *canonical-name*

# Using DNS for load-balancing

- **Can have multiple RR of most types for one name**
  - Required for NS records (for availability)
  - Useful for A records
  - (Not legal for CNAME records)
- **Servers rotate order in which records returned**
  - Most apps just use first address returned  
(`#define h_addr h_addr_list[0]`)
  - Even if name servers cache results, clients will be spread among many servers
- **Example: dig cnn.com several times**

# TXT records

- Can place arbitrary text in DNS
  - name* [TTL] [IN] TXT "*text*" ...
  - *text* – whatever you want it to mean

- Great for prototyping new services
  - Don't need to change DNS infrastructure

- Example: `dig aol.com txt`

```
aol.com. 300 IN TXT "v=spf1 ip4:152.163.225.0/24 ip4:205.188.139.0/24 ip4:205.188.144.0/24 ip4:205.188.156.0/23 ip4:205.188.159.0/24 ip4:64.12.136.0/23 ip4:64.12.138.0/24 ptr:mx.aol.com ?all"
```

- SPF = "sender permitted from", protection from forged email
- SPF specifies IP addresses allowed to send mail from @aol.com
- Can have incremental deployment
- Only mail servers must change, DNS can stay the same

# SRV records

- **Service location records**

*service.proto.name [...] SRV prio weight port target*

- **service** – e.g., `_sfs` for NYU's SFS file system
- **proto** – `_tcp` or `_udp`
- **name** – domain name record applies to
- **prio** – as with MX records, lower # → higher priority
- **weight** – within priority, affects randomization of order
- **port** – TCP or UDP port number
- **target** – Server name, for which client needs A record

- **Generalization of MX records for arbitrary services**

# Uses of SRV records

- **Relatively new standard**
  - Not yet supported by many applications
  - Would be very useful if it were
- **Useful for both load balancing and backup servers**
- **Also useful for port number conservation**
  - Most services use well-known ports below 1,024
  - Will someday run out of port numbers
  - But unlikely one machine running 1,024 services
- **And for IP address conservation**
  - May have multiple machines with only one IP address
  - Can map services to correct machine using ports

# Mapping addresses to names

- Sometimes want to find DNS name given address

- PTR records specify names

*name* [TTL] [IN] PTR "*ptrdname*"

- *name* – somehow encode address...how?

- *ptrdname* – domain name for this address

- IPv4 addrs stored under in-addr.arpa domain

- Reverse name, append in-addr.arpa

- To look up 216.165.108.10 → 10.108.165.216.in-addr.arpa.

- Why reversed? Delegation!

- IPv6 under ip6.arpa

- Historical note: ARPA funded original Internet

# Classless in-addr delegation

- How to delegate on non-byte boundary?
- Solution: Use CNAME records
  - So-called *classless* in-addr delegation
- Example:

```
1.108.165.216.in-addr.arpa. CNAME 1.ptr.your-domain.com.
```

```
2.108.165.216.in-addr.arpa. CNAME 2.ptr.your-domain.com.
```

```
3.108.165.216.in-addr.arpa. CNAME 3.ptr.your-domain.com.
```

# IPv6 addresses

- **Next generation IP addresses**
  - 16-bytes long, alleviates address shortage
- **Originally mapped with AAAA records**
  - Like A records, only 4 times as long

## Coming up

- Today, 4:30pm: VMWare talk, recruiting [Lubrano]
- Today, 7:30pm: Startup Dinner [Lubrano]
- Tue, Oct 16th: **Midterm**
- Tue, Oct 16th 5:30pm: Startups by CS Grads [CIT 165]
- Tue, Oct 17th 6p: Sun/Solaris talk, recruiting [Lubrano]