Systems and Networks Architecture: Naming, Layering, and Communication

COS 518 *Advanced Computer Systems*

Lecture 2

Kyle Jamieson

[Credits: Selected content from Brad Karp, Scott Shenker]

Today

- We’ll cover three topics:

1. **Naming and the Domain Name System**

2. Layering and the *End-to-End Argument*

3. Administrivia: Reviews and presentations

DNS hostname versus IP address

- **DNS host name** (*e.g.* cos518.cs.princeton.edu)
  - Mnemonic name appreciated by humans
  - Variable length, full alphabet of characters
  - Provide little (if any) information about location

- **IP address**
  - Numerical address appreciated by routers
  - Fixed length, binary number
  - Hierarchical, related to host location

Original design of the DNS

- Per-host file named `/etc/hosts`
  - Flat namespace: each line is an IP address and a 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

- Basically, the biggest wide-area distributed database in the world.

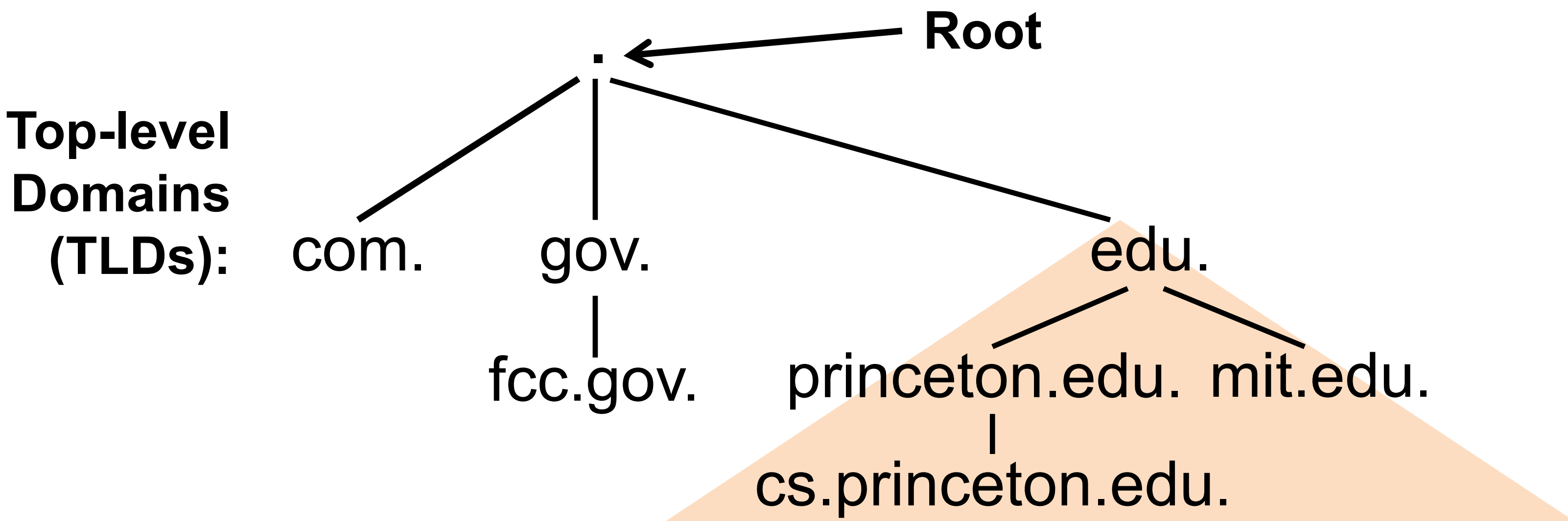
- **Goals:**
  - Scalability; decentralized maintenance
  - Robustness; global scope (names mean the same thing everywhere)
  - Distributed updates/queries
  - Good performance

- 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

The DNS namespace is hierarchical



- **Hierarchy of namespace follows hierarchy of servers**
  - **Zone:** contiguous tree/subtree in the namespace

- Set of nameservers answers queries for names within zone

- Nameservers store names and links to other servers in tree

Many uses of DNS

- Hostname to IP address translation
  - IP address to hostname translation (***reverse lookup***)

- Host name ***aliasing*** allows other names for a host
  - ***Alias*** host names point to ***canonical*** hostname

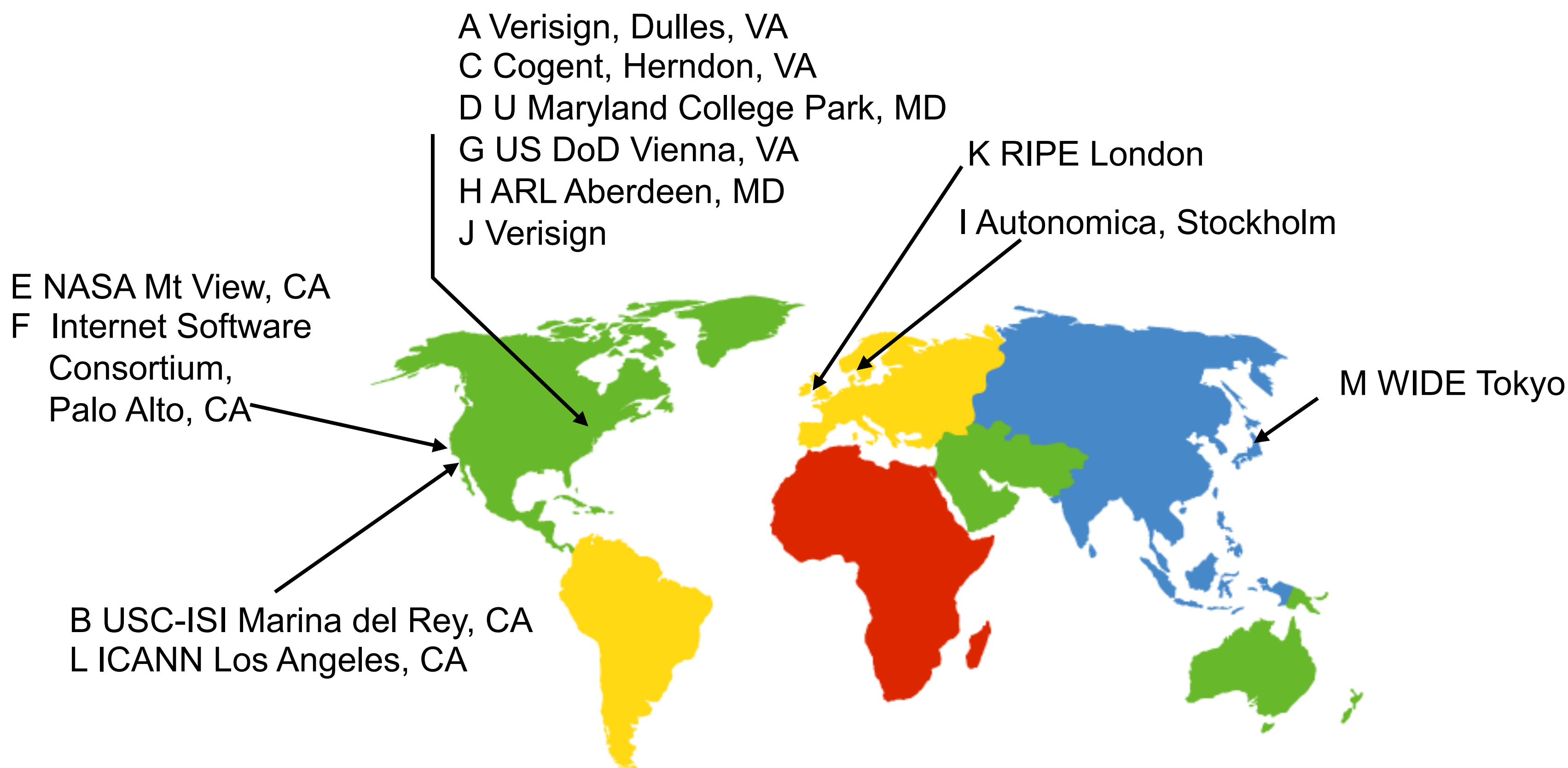
- Email: Lookup a zone's mail server based on zone name

- **Content distribution networks**
  - Load-balance between servers in different locations

  - Complex, hierarchical arrangements possible

DNS root nameservers

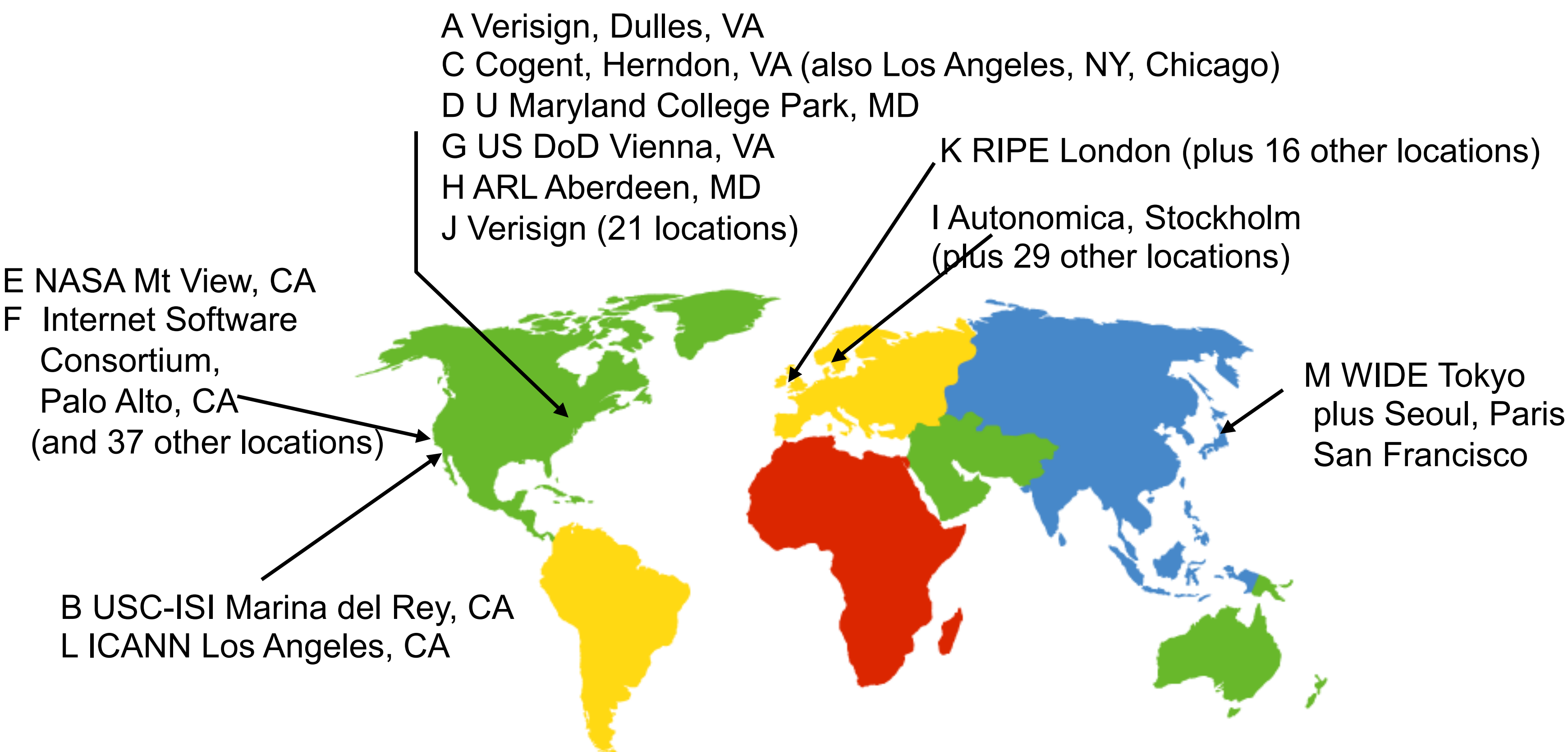
- 13 root servers. *Does this scale?*



[Slide credit: Scott Shenker]

DNS root nameservers

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



[Slide credit: Scott Shenker]

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* for edu TLD

- ***Authoritative*** DNS servers
  - An organization's DNS servers, providing authoritative information for organization's servers
  - Can be maintained by organization or ISP

Local name servers

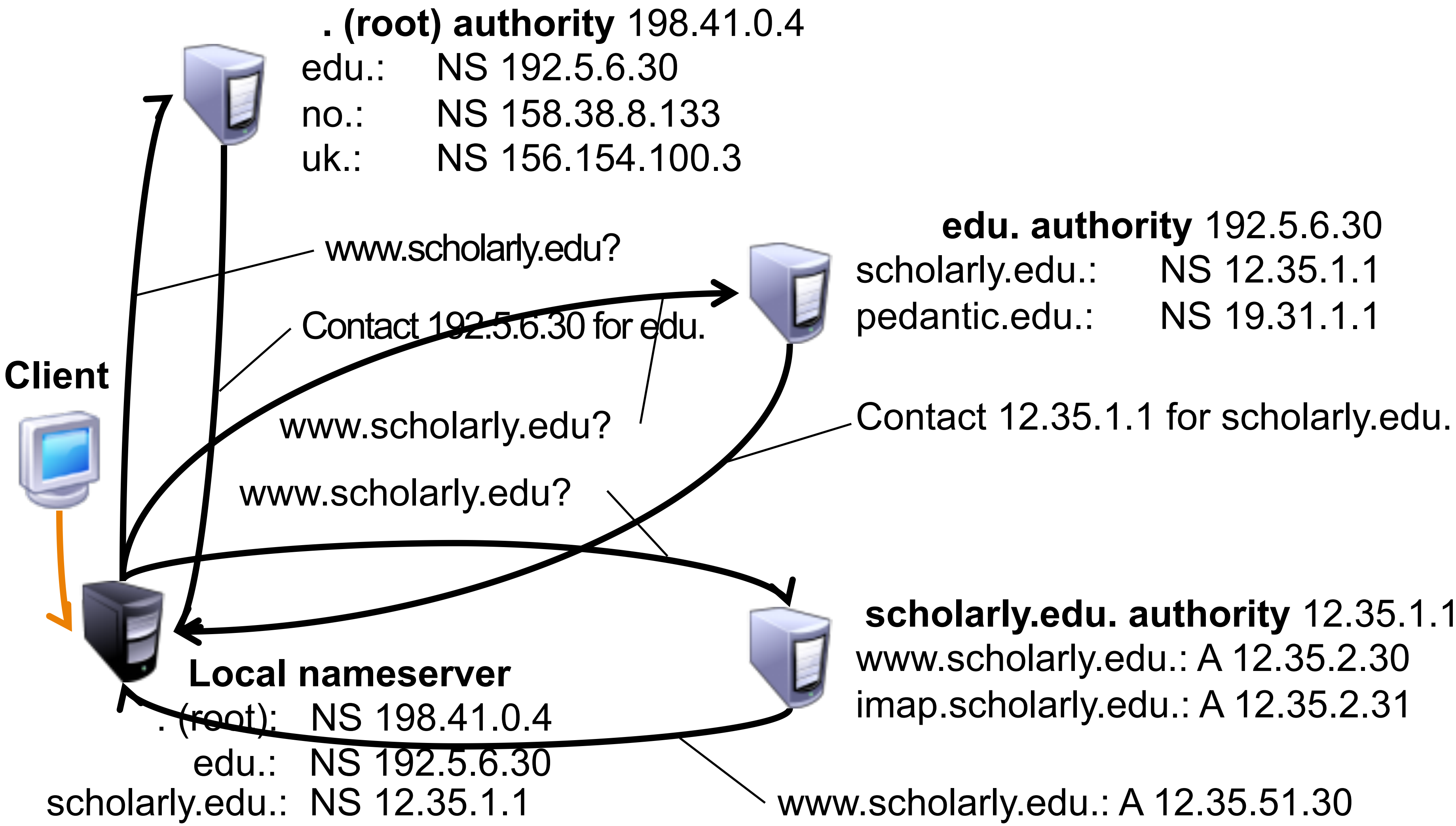
- Do not strictly belong to hierarchy
- Each ISP (or company, or university) has one
  - Also called ***default*** or ***caching*** name server
- When host makes DNS query, query is sent to its local DNS server
  - Acts as proxy, forwards query into hierarchy
  - Does work for the client

DNS in operation

- Most queries and responses are UDP datagrams
- Two types of queries:
- ***Recursive***

- ***Iterative***

A recursive DNS lookup



Recursive versus iterative queries

Recursive query

- Less burden on client**
- **More burden on nameserver** (has to return an answer to the query)
- Most root and TLD servers will **not answer** (shed load)
  - Local name server answers recursive query

Iterative query

- **More burden on client**
- Less burden on nameserver (simply refers the query to another server)

** The entity performing the query

DNS resource record (RR): Overview

DNS is a distributed database storing **resource records**

RR includes: **`(name, type, value, time-to-live)`**

- Type = **A** (address)
  - `name` is hostname
  - `value` is IP address
- Type = **NS** (name server)
  - `name` is domain (e.g. cs.ucl.ac.uk)
  - `value` is hostname of authoritative name server for this domain
- Type = **CNAME**
  - `name` is an alias for some "canonical" (real) name
  - `value` is canonical name
- Type = **MX** (mail exchange)
  - `value` is name of mail server associated with domain name
  - `pref` field discriminates between multiple MX records

Example: DNS query "*in situ*" (1/3)

```
$ dig @a.root-servers.net www.freebsd.org +norecurse
; <<>> DiG 9.4.3-P3 <<>> @a.root-servers.net
  www.freebsd.org +norecurse
; (1 server found)
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 57494
;; flags: qr; QUERY: 1, ANSWER: 0, AUTHORITY: 6,
  ADDITIONAL: 12

;; QUESTION SECTION:
;www.freebsd.org.        IN  A

;; AUTHORITY SECTION:
org.           172800 IN  NS  b0.org.afilias-nst.org.
org.           172800 IN  NS  d0.org.afilias-nst.org.

;; ADDITIONAL SECTION:
b0.org.afilias-nst.org.  172800 IN  A  199.19.54.1
d0.org.afilias-nst.org.  172800 IN  A  199.19.57.1

;; Query time: 177 msec
;; SERVER: 198.41.0.4#53(198.41.0.4)
;; WHEN: Wed Oct 28 07:32:02 2009
;; MSG SIZE  rcvd: 435
```

"Glue" record

Example: DNS query “*in situ*” (2/3)

```
$ dig @199.19.54.1 www.freebsd.org +norecurse
; <<>> DiG 9.4.3-P3 <<>> @a0.org.afilias-nst.org
  www.freebsd.org +norecurse
; (1 server found)
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id:
  39912
;; flags: qr; QUERY: 1, ANSWER: 0, AUTHORITY: 3,
  ADDITIONAL: 0

;; QUESTION SECTION:
;www.freebsd.org.       IN A

;; AUTHORITY SECTION:
freebsd.org.    86400  IN NS ns1.isc-sns.net.
freebsd.org.    86400  IN NS ns2.isc-sns.com.
freebsd.org.    86400  IN NS ns3.isc-sns.info.

;; Query time: 128 msec
;; SERVER: 199.19.56.1#53(199.19.56.1)
;; WHEN: Wed Oct 28 07:38:40 2009
;; MSG SIZE  rcvd: 121
```

- **No glue record provided** for ns1.isc-sns.net, so need to go off and resolve (**not shown here**), restart the query

Example: DNS query "*in situ*" (3/3)

```
$ dig @ns1.isc-sns.net www.freebsd.org +norecurse
; <<>> DiG 9.4.3-P3 <<>> @ns1.isc-sns.net www.freebsd.org
  +norecurse
; (1 server found)
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 17037
;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 3,
  ADDITIONAL: 5

;; QUESTION SECTION:
;www.freebsd.org.        IN A

;; ANSWER SECTION:
www.freebsd.org.   3600   IN A   69.147.83.33

;; AUTHORITY SECTION:
freebsd.org.     3600   IN NS ns2.isc-sns.com.
freebsd.org.     3600   IN NS ns1.isc-sns.net.
freebsd.org.     3600   IN NS ns3.isc-sns.info.

;; ADDITIONAL SECTION:
ns1.isc-sns.net.    3600   IN A   72.52.71.1
ns2.isc-sns.com.    3600   IN A   38.103.2.1
ns3.isc-sns.info.   3600   IN A   63.243.194.1
```

DNS Caching

- Performing all these queries takes time
  - And all this **before** actual communication takes place
  - *e.g.,* one-second latency before starting Web download
- **Caching** can greatly reduce overhead
  - The top-level servers very rarely change
  - Popular sites (e.g., www.cnn.com) visited often
  - Local DNS server often has the information cached
- How DNS caching works
  - DNS servers cache responses to queries
  - Responses include a **time-to-live** (TTL) field
  - Server deletes cached entry after TTL expires

[Slide credit: Scott Shenker]

A word on DNS security

- Implications of subverting DNS:

1. Redirect victim's web traffic to rogue servers

2. Redirect victim's email to rogue email servers (MX records in DNS)

- Does Secure Sockets Layer (SSL) provide protection?
  - **Yes**—user will get "wrong certificate" if SSL enabled
  - **No**—SSL not enabled or user ignores warnings
  - **No**—how is SSL trust established? **Often, by email!**

Security Problem #1: Coffee shop

- As you sip your latte and surf the Web, how does your laptop find `google.com`?

- Answer: it asks the local DNS nameserver
  - Which is run by the coffee shop or their contractor
  - And can return to you any answer they please
  - Including a "man in the middle" site that forwards your query to Google, gets the reply to forward back to you, yet can change anything they wish in either direction

- How can you know you're getting correct data?
  - Today, you can't. (Though if site is HTTP**S**, that helps)
  - One day, hopefully: **DNSSEC** extensions to DNS

Security Problem #2: Cache poisoning

- Suppose you are evil and **you control** the name server for **`foobar.com.`** You receive a request to resolve `www.foobar.com` and reply:

```
;; QUESTION SECTION:
;www.foobar.com.                 IN      A

;; ANSWER SECTION:
www.foobar.com.         300      IN      A       212.44.9.144

;; AUTHORITY SECTION:
foobar.com.             600      IN      NS      dns1.foobar.com.
foobar.com.             600      IN      NS      google.com.

;; ADDITIONAL SECTION:
google.com.               5      IN      A       212.44.9.155
```

Evidence of the attack disappears 5 seconds later!

A foobar.com machine, *not* google.com

DNS cache poisoning (cont'd)

- Okay, but how do you get the victim to look up `www.foobar.com` in the first place?

- Perhaps you connect to their mail server and send
  - **`HELO www.foobar.com`**
  - Which their mail server then looks up to see if it corresponds to your source address (anti-spam measure)

- Perhaps you send many spam or phishing emails containing a link to `www.foobar.com`

Bailiwick checking

- Local resolving nameserver **ignores** all RRs **not in or under the same zone** as the **question**
- Widely deployed since *ca.* 1997
- Other attacks are possible (*e.g.* Kaminsky poisoning)

```
;; QUESTION SECTION:
;www.foobar.com.                 IN      A

;; ANSWER SECTION:
www.foobar.com.          300     IN      A       212.44.9.144

;; AUTHORITY SECTION:
foobar.com.              600     IN      NS      dns1.foobar.com.
foobar.com.              600     IN      NS      google.com.

;; ADDITIONAL SECTION:
google.com.                5     IN      A       212.44.9.155
```

Today

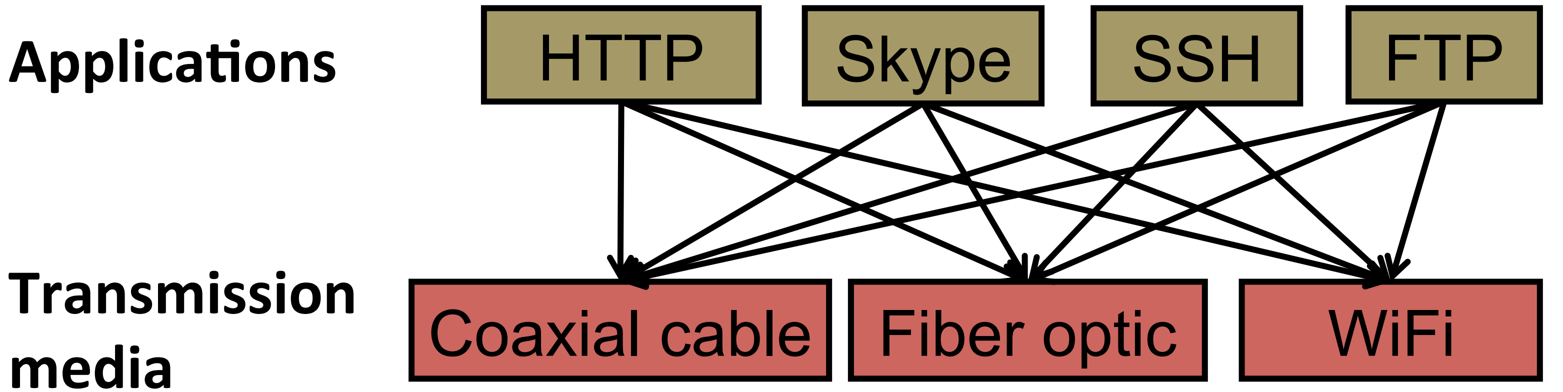
- We'll cover three topics:

1. Naming and the Domain Name System

2. **Layering and the *End-to-End Argument***

3. Administrivia: Reviews and presentations

Coping with application/link heterogeneity



- Re-implement every application for every new underlying transmission medium?
- Change every application on any change to an underlying transmission medium (and vice-versa)?

- **No!** But how does the Internet design avoid this?

(Review) Computer system modularity

- **Key idea: Partition system into modules and abstractions**

- Well-defined interfaces give flexibility and isolation
  - Hide implementation, thus, it can be freely changed
  - Extend functionality of system by adding new modules

- *e.g.*, libraries encapsulating set of functionality

- *e.g.*, a programming language and compiler abstracts away how a particular CPU and operating system work

Layering: a modular approach

- Partition protocols on the Internet into **layers**
  – Each layer solely relies on services from layer below
  – Each layer solely exports services to layer above

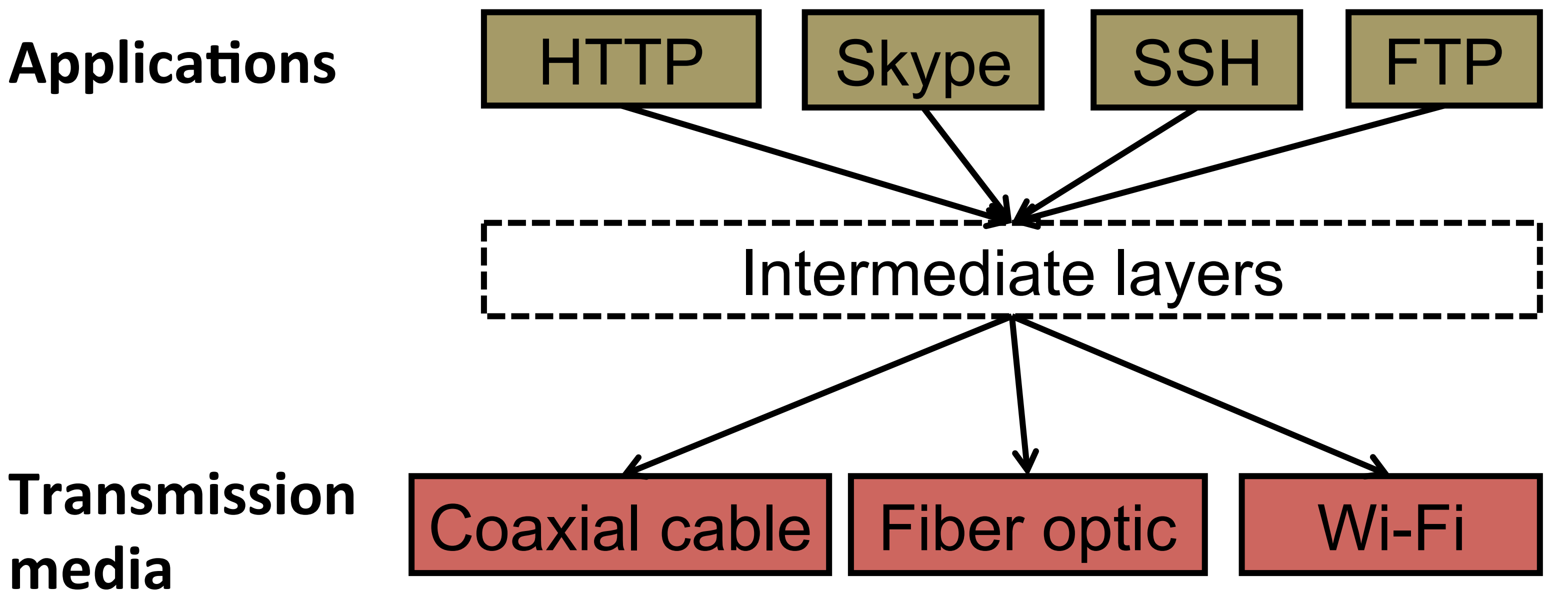
- Advantages of layering:

1. Decomposes problem of building a network into manageable pieces

2. Results in a more modular design. Additions or changes are usually isolated to one layer

3. Layer n hides complexity of layer $n-1$ to higher layers

Internet solution: Intermediate layers



- Intermediate layers provide a set of abstractions for applications and media
- New applications or media need only implement for intermediate layer's interface

Physical layer (L1)

- **Service:** move bits between two systems connected by a single physical link
- **Interface:** specifies **how to send, receive bits**
  - *e.g.*, require quantities and timing
- **Protocols:** coding scheme used to represent bits, voltage levels, duration of a bit

Data link layer (L2)

- **Service:** enables **end hosts** to exchange **atomic** messages with one another
  - Using **abstract addresses** (*i.e.*, not just direct physical connections)
  - Perhaps over **multiple physical links**, but using the same framing (headers/trailers)
  - Possibly **arbitrates access** to common physical media
  - Possibly implements **reliable transmission**, flow control
- **Interface:** send messages (frames) to other end hosts; receive messages addressed to end host
- **Protocols:** addressing, routing, Media Access Control (MAC) (*e.g.*, CSMA/CD: Carrier Sense Multiple Access / Collision Detection)

Network layer (L3)

- **Service:** Deliver packets to destinations on **other networks** (inter-network, across multiple L2 networks)
  - Works **across** networking technologies (*e.g.*, Ethernet, 802.11, frame relay, ATM …)
  - Possibly includes packet **scheduling/priority**
  - Possibly includes **buffer management**

- **Interface:** send packets to specified inter-network destinations; receive packets destined for end host

- **Protocols:** define inter-network addresses (globally unique); construct routing tables

- **Examples:** ***IP***, the Internet Protocol

Transport layer (L4)

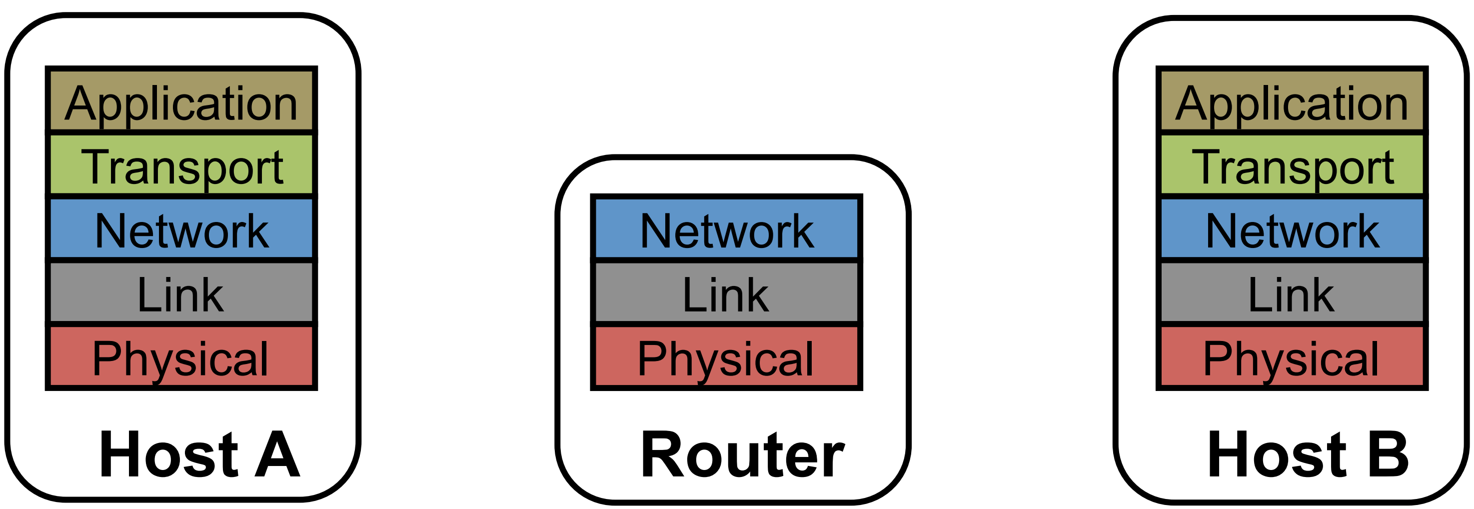
- **Service:** Provides **end-to-end** communication between **processes** on **different hosts**
  - **Demultiplexing** of communication between hosts
  - Possibly **reliability** in the presence of errors
  - **Rate adaption** (flow-control, congestion control)
- **Interface:** send message to specific process at given destination; local process receives messages sent to it
- **Protocol:** Perhaps implement reliability, flow control, packetization of large messages, framing
- **Examples:** Transport Control Protocol (TCP), User Datagram Protocol (UDP)

Application layer (L7)

- **Service:** Any service provided to the end user
- **Interface:** Depends on the application
- **Protocol:** Depends on the application
- **Examples:** File Transfer Protocol (FTP), Skype, Simple Mail Transfer Protocol (SMTP), Hypertext Transport Protocol (HTTP), BitTorrent, many others…
- What happened to layers 5 & 6?
  - "Session" and "Presentation" layers
  - Part of OSI architecture, but not Internet architecture

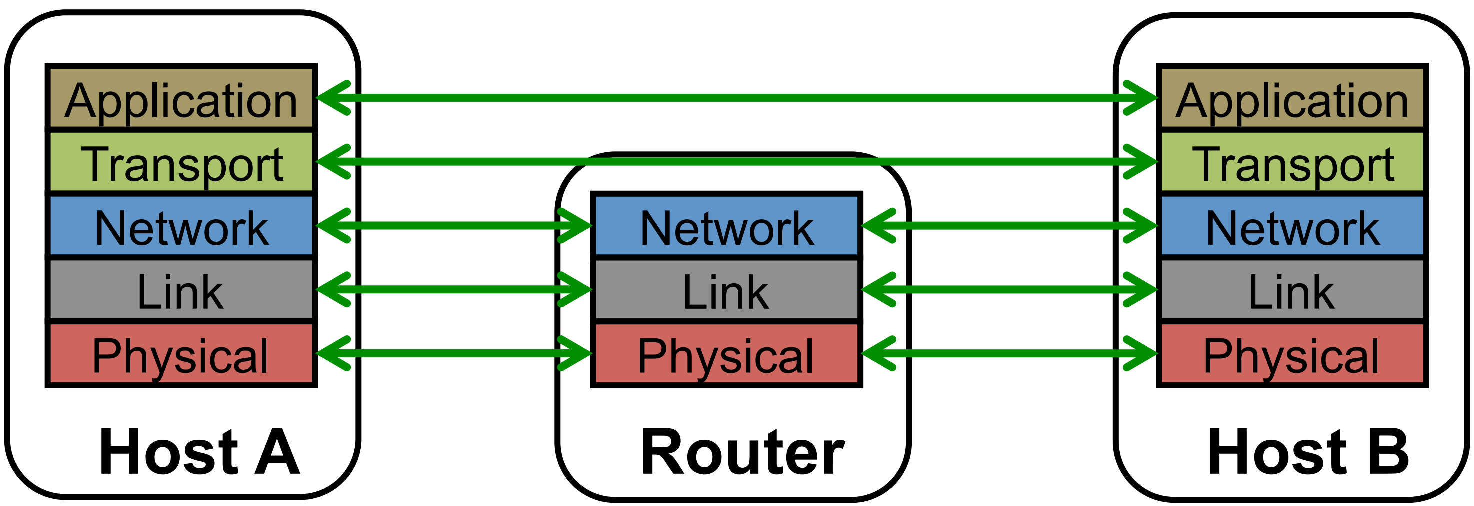
Who does what?

- Five "Internet architecture" layers
  - Lower three layers are implemented **everywhere**
  - Top two layers are implemented **only at end hosts**



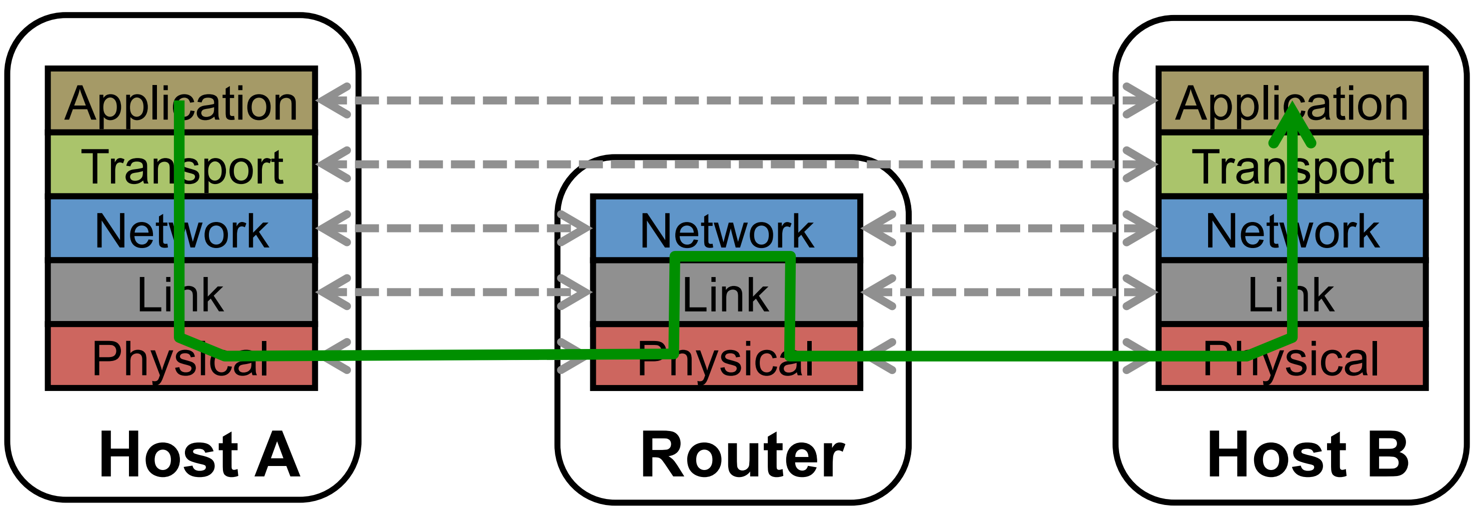
Logical communication

- Each layer on a host interacts with its **peer** host's **corresponding layer** via the **protocol interface**



Physical communication

- Communication goes down to physical network
- Then from network peer to peer
- Then up to the relevant layer



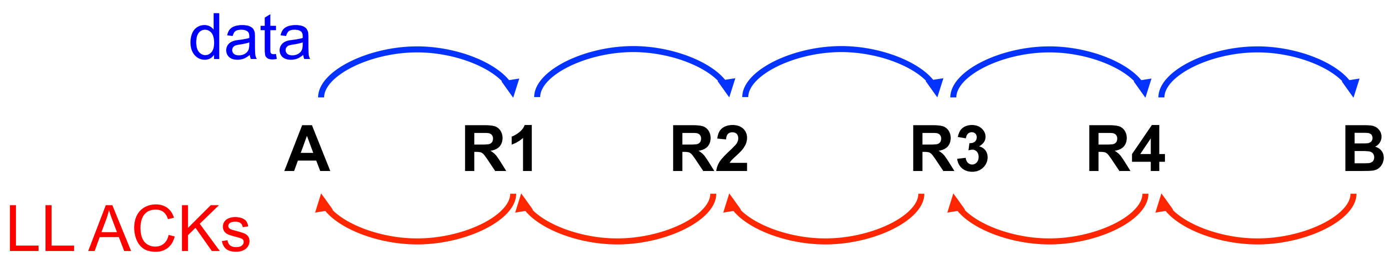
Motivation: End-to-End Argument

- Five layers in the Internet architecture model

- Five places to solve many of same problems:
  - In-order delivery
  - Duplicate-free delivery
  - Reliable delivery after corruption, loss
  - Encryption
  - Authentication

- ***In which layer(s) should a particular function be implemented?***

Example: Careful file transfer from A to B



- **Goal: Accurately copy file on A's disk to B's disk**

- Straw man design:
  - Read file from **A**'s disk
  - **A** sends stream of packets containing file data to **B**
    - L2 retransmission of lost or corrupted packets at each hop
  - **B** writes file data to disk

- ***Does this system meet the design goal?***
  - Bit errors on links not a problem

Where can errors happen?

- On **A**’s or **B**’s disk
- In **A**’s or **B**’s RAM or CPU
- In **A**’s or **B**’s software
- In the RAM, CPU, or **software** of **any router** that forwards packet

- **Why** might errors be likely?
  – Drive for CPU speed and storage density: pushes hardware to EE limits, engineered to tight tolerances
    - *e.g.*, today’s disks return data that are the output of an maximum-likelihood estimation!

  – Bugs abound!

Solution: End-to-End verification

1. **A** keeps a **checksum** with the on-disk data
   - *Why not compute checksum at start of transfer?*
2. **B** computes checksum over received data, sends to **A**
3. **A** compares the two checksums and resends if not equal

- Can we eliminate hop-by-hop error detection?
  - Suppose there's a router with bad RAM; how to find it?

- Is a whole-file checksum enough?
  - Poor performance: must resend whole file each time one packet (bit) corrupted!

End-to-End principle

- Only the **application at communication endpoints** can completely and correctly implement a function

- Processing in **middle alone cannot** provide function
  - Processing in middle **may**, however, be an important **performance optimization**

- Engineering middle hops to provide guaranteed functionality is often **wasteful of effort, inefficient**

Perils of low-layer implementation

- **Entangles** application behavior with network internals

- **Suppose** each IP router **reliably transmitted** to next hop
  - Result: Lossless delivery, but **variable delay**
  - ftp: **Okay,** move huge file reliably (just end-to-end TCP works fine, too, though)
  - Skype: **Terrible,** jitter packets when a few corruptions or drops not a problem anyway

- **Complicates deployment** of innovative applications
  - Example: Phone network *v.* the Internet

Advantages of low-layer implementation

- Each application author **needn't recode a shared function**

- Overlapping error checks (e.g., checksums) at all layers invaluable in **debugging and fault diagnosis**

- If end systems not cooperative (increasingly the case), **only way to enforce resource allocation!**

Challenge: End-to-end authentication + encryption

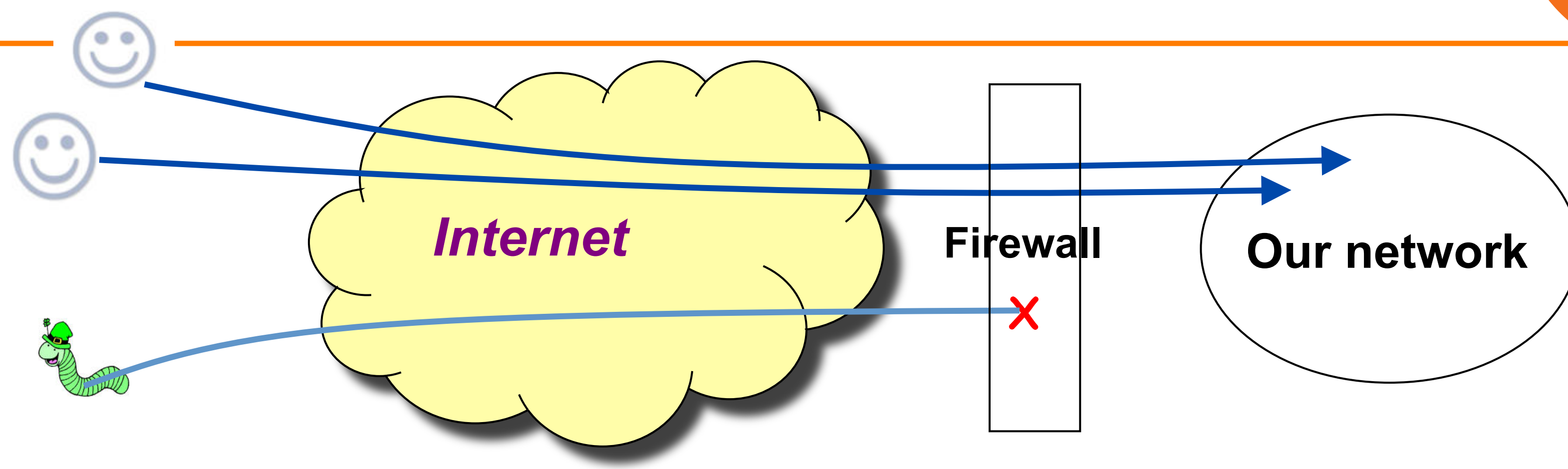
- Use a public PC to check your email using IMAP/SSL
  - **Authenticates** server to you and you to server robustly
  - **Encrypts** between you robustly for confidentiality

- Key security consideration: ***threat model***, *i.e.*
  - Which attacks are you explicitly defending against?
  - Which are you ignoring?
  - What does it **cost** your adversary to mount an attack?

- What are you trusting?
  - Mail reader application (could be trojaned)
  - OS (could also be trojaned)
  - Hardware (*e.g.*, "fake ATM" cases)

- E2E notion of security must consider **integrity of software and hardware at endpoints, possibly even of users!**

End-to-end violation: Firewalls



- Box in middle of network that blocks "malicious" traffic
  - End-host software often vulnerable to remote-exploit malware
  - Users are **naive,** don't keep systems patched and up-to-date
- Firewalls clearly **violate the e2e principle**
  - **Endpoints** are capable of deciding what traffic to ignore
  - Firewall **entangled** with design of network and higher protocol layers and apps, and vice-versa
    - Example: New ECN bit to improve TCP congestion control; many firewalls filtered all such packets!
- **Yet, we probably do need firewalls**

Summary: End-to-End principle

- Many functions **must** be implemented at application endpoints to provide desired behavior
  - Even if implemented in “middle” of network

- End-to-end approach **decouples design** of components in network interior from design of applications at edges
  - Some functions still **benefit** from implementation in **network interior** at the cost of entangling interior and edges

- End-to-end principle is **not sacred;** it’s just a way to think critically about design choices in communication systems

Today

- We'll cover three topics:

1. Naming and the Domain Name System

2. Layering and the *End-to-End Argument*

3. **Administrivia: Reviews and presentations**

Streamlined presentation/review process

- **Presentation signup** on Piazza has already begun
  - Signup today if you have not already done so

- Signup on a Piazza poll to **review** one of two papers ~48 hours before class (FCFS)
  - **Don't signup** for paper if > ~15 students (half the class) signed up

- At least **two hours** before class: Submit review and read others' reviews

How to critically read a paper (1/2)

- **Read once for perspective, twice for details**
  - Large systems have many "moving parts" (Lect. 1)
  - Analogous to "build one to throw one away", you may need to **revisit the paper** in order to know which design details to focus on
- **Take notes** as you read
  - Question assumptions, importance of problem, important effects not mentioned by authors
  - Write questions to **track** what you don't understand

How to critically read a paper (2/2)

- **Don't pass by** ideas/design details until you **understand**
  - May need to re-read a paragraph, many times, or even discuss with peers
  - You can't fully understand if the design is good unless you understand all the details: be vigilant!

- **Don't presume** authors' assumptions or design choices **correct** simply because paper was published!

How to evaluate a research paper?

- Important, relevant **problem**? Clever **idea**? These are **orthogonal**!
- Reasonable assumptions and models?
- **Longer ago published,** more you can judge **impact:**
  - Does everyone now use systems **derived** from it?
  - Has the **idea** shown up in many different contexts?
- **Recent papers:** more on cleverness, promise
- Other contributions possible
  - **Thorough investigation** of complex phenomenon
  - Comparison that **brings sense to an area**

Presentation guidelines

- Slides for a talk 12 to 15 minutes in length
- Come prepared to lead class discussion after talk

Content of a presentation

- Motivation and problem statement
- State main contributions of work (core ideas)
- Description of central design
- Experimental evaluation
- Related work
- Future work

Also applies to reviews

- "Opinion part"

Description of central design

- You won't have time/space to discuss **every** detail, so present those that are **most important**…
  – To understanding **how and why the system**, design, or algorithm works
  – To **understanding results** in the experimental evaluation

- Clarity is very important here
  – Usually describe in a **"top-down" fashion**
  – Start with the overall problem
  – Identify parts of the solution, then identifying the sub-parts of those parts, *& c.*

Experimental evaluation

- **What questions** do the authors ask in their evaluation?
- What is the authors' hypothesis for each question and why?
- You won’t have time to present all results, so present most important results
- For any **graph** you show or refer to:
  - First, **explain the axes**
  - Explain **overall trend:** why system behaves as it does
  - Justify explanation by **referring to relevant details** of the system's **design** and experiment's design
  - Does anything in the graph seem **anomalous**? Note and try to explain

Related and future work

- What are the **most closely related** other systems/results?
  - How are they **similar**? How are they **different**?
  - Is the difference between the work you are presenting and the related work **significant**?
- Should read citations enough to understand differences
- Should search for related work published after/with the paper
- **No need to claim** the work you are presenting is **"better" or "worse"** than a particular piece of related work
  - Often it is simply that the two pieces of work are different
- But, should **articulate the precise difference** (*e.g.*, "this work solves a slightly different problem…")

"Opinion part"

- Offer your final critical assessment:
  - What are the strengths of the work?
  - What are the weaknesses/limitations?
  - What important questions are left unanswered?

Advice on giving a good talk

- **Rehearse your talk** many times
  – Pay attention to length

- **Help one another** present clearly

- Use examples to explain difficult ideas
  – **Animations and pictures** help tremendously
  – There is utility in **creating your own**

- Be **constructively critical** throughout