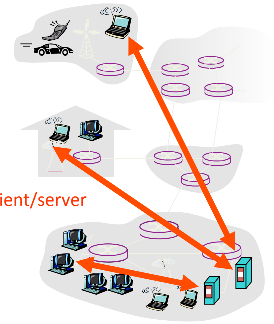
Topic 6 – Applications

- Traditional Applications (web)
- Infrastructure Services (DNS)
- Multimedia Applications (SIP)
- P2P Networks

2

Client-server architecture



server:
- always-on host
- permanent IP address
- server farms for scaling

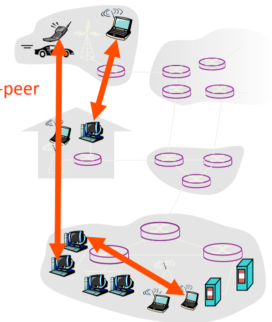
clients:
- communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do not communicate directly with each other

3

Pure P2P architecture

- *no* always-on server
- arbitrary end systems directly communicate
- peers are intermittently connected and change IP addresses

Highly scalable but difficult to manage



4

Hybrid of client-server and P2P

Skype
- voice-over-IP P2P application
- centralized server: finding address of remote party:
- client-client connection: direct (not through server)

Instant messaging
- chatting between two users is P2P
- centralized service: client presence detection/ location
  - user registers its IP address with central server when it comes online
  - user contacts central server to find IP addresses of buddies

5

Processes communicating

Process: program running within a host.

- within same host, two processes communicate using inter-process communication (defined by OS).
- processes in different hosts communicate by exchanging messages

Client process: process that initiates communication

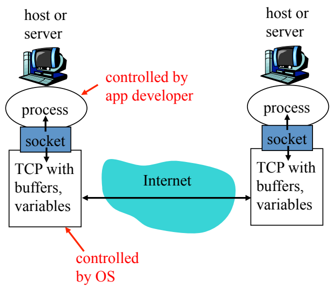
Server process: process that waits to be contacted

❒ Note: applications with P2P architectures have client processes & server processes

6

Sockets – an abstraction hiding layers

- process sends/receives messages to/from its socket
- socket analogous to door
  - sending process shoves message out door
  - sending process relies on transport infrastructure on other side of door which brings message to socket at receiving process



❒ Socket API: (1) choice of transport protocol; (2) ability to fix a few parameters

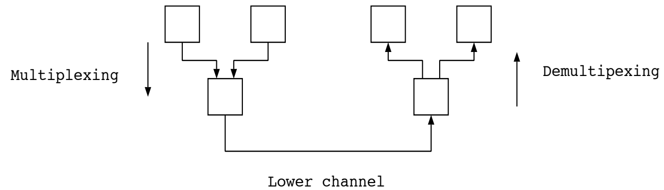
7

Addressing processes

- to receive messages, process must have *identifier*
- host device has unique 32-bit IP address
- Q: does IP address of host on which process runs suffice for identifying the process?
  - A: No, *many* processes can be running on same host
- *identifier* includes both IP address and port numbers associated with process on host.
- Example port numbers:
  - HTTP server: 80
  - Mail server: 25
- to send HTTP message to gaia.cs.umass.edu web server:
  - IP address: 128.119.245.12
  - Port number: 80
- more shortly…

8

Recall: Multiplexing is a service provided by (each) layer too!



Application: one web-server multiple sets of content
Host: one machine multiple services
Network: one physical box multiple addresses (like vns.cl.cam.ac.uk)
….

UNIX: /etc/protocols = examples of different transport-protocols on top of IP

UNIX: /etc/services = examples of different (TCP/UDP) services – by port

(THESE FILES ARE EXAMPLES OF NAME SERVICES)

App-layer protocol defines

- Types of messages exchanged,
  - e.g., request, response
- Message syntax:
  - what fields in messages & how fields are delineated
- Message semantics
  - meaning of information in fields
- Rules for when and how processes send & respond to messages

Public-domain protocols:

- defined in RFCs
- allows for interoperability
- e.g., HTTP, SMTP

Proprietary protocols:

- e.g., Skype

10

What transport service does an app need?

Data loss

- some apps (e.g., audio) can tolerate some loss
- other apps (e.g., file transfer, telnet) require 100% reliable data transfer

Timing

- some apps (e.g., Internet telephony, interactive games) require low delay to be "effective"

Throughput

- ❒ some apps (e.g., multimedia) require minimum amount of throughput to be "effective"
- ❒ other apps ("elastic apps") make use of whatever throughput they get

Security

- ❒ Encryption, data integrity, …

Mysterious secret of *Transport*

- There is more than sort of *transport* layer

Shocked?

I seriously doubt it…

We call the two most common TCP and UDP

11

Naming

- Internet has one global system of addressing: IP
  - By explicit design
- And one global system of naming: DNS
  - Almost by accident
- At the time, only items worth naming were hosts
  - A mistake that causes many painful workarounds
- Everything is now named relative to a host
  - Content is most notable example (URL structure)

12

Logical Steps in Using Internet

- Human has name of entity she wants to access
  - Content, host, etc.
- Invokes an application to perform relevant task
  - Using that name
- App invokes DNS to translate name to address
- App invokes transport protocol to contact host
  - Using address as destination

13

Addresses vs Names

- Scope of relevance:
  - App/user is primarily concerned with names
  - Network is primarily concerned with addresses
- Timescales:
  - Name lookup once (or get from cache)
  - Address lookup on each packet
- When moving a host to a different subnet:
  - The address changes
  - The name does not change
- When moving content to a differently named host
  - Name and address both change!

14

Relationship Betw'n Names/ Addresses

- Addresses can change underneath
  - Move www.cnn.com to 4.125.91.21
  - Humans/Apps should be unaffected
- Name could map to multiple IP addresses
  - www.cnn.com to multiple replicas of the Web site
  - Enables
    - Load-balancing
    - Reducing latency by picking nearby servers
- Multiple names for the same address
  - E.g., aliases like www.cnn.com and cnn.com
  - Mnemonic stable name, and dynamic canonical name
    - Canonical name = actual name of host

15

Mapping from Names to Addresses

- Originally: per-host file /etc/hosts
  - SRI (Menlo Park) kept master copy
  - Downloaded regularly
  - Flat namespace
- Single server not resilient, doesn't scale
  - Adopted a distributed hierarchical system
- Two intertwined hierarchies:
  - Infrastructure: hierarchy of DNS servers
  - Naming structure: www.cnn.com

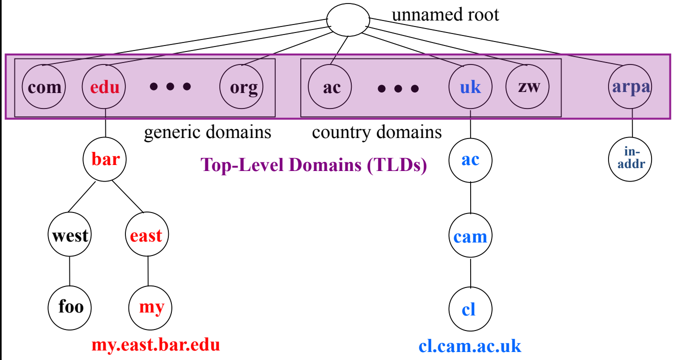
16

Domain Name System (DNS)

- Top of hierarchy: Root
  - Location hardwired into other servers
- Next Level: Top-level domain (TLD) servers
  - .com, .edu, etc.
  - Managed professionally
- Bottom Level: Authoritative DNS servers
  - Actually do the mapping
  - Can be maintained locally or by a service provider

17

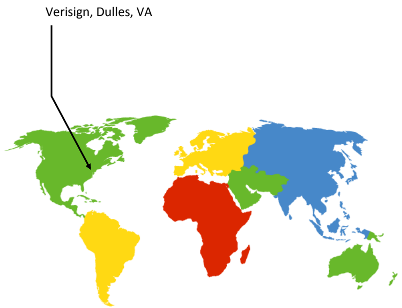
Distributed Hierarchical Database



18

DNS Root

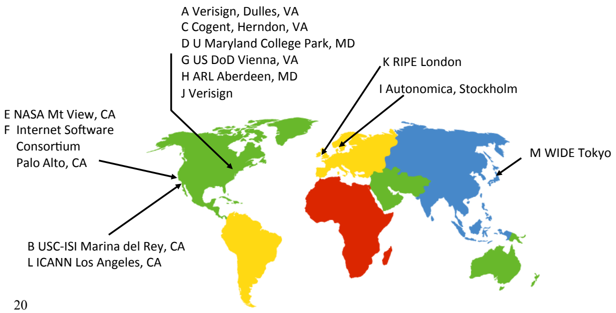
- Located in Virginia, USA
- How do we make the root scale?



19

DNS Root Servers

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



20

DNS Root Servers

- 13 root servers (see http://www.root-servers.org/)
  - Labeled A through M
- Replication via any-casting (localized routing for addresses)

21

Using DNS

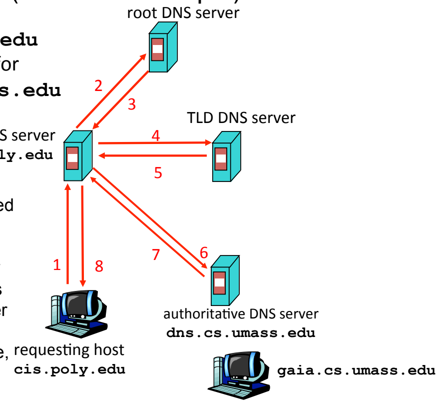
- Two components
  - Local DNS servers
  - Resolver software on hosts
- Local DNS server ("default name server")
  - Usually near the endhosts that use it
  - Local hosts configured with local server (e.g., /etc/resolv.conf) or learn server via DHCP
- Client application
  - Extract server name (e.g., from the URL)
  - Do gethostbyname() to trigger resolver code

22

How Does Resolution Happen?

(**Iterative** example)

Host at **`cis.poly.edu`** wants IP address for **`gaia.cs.umass.edu`**



iterated query:

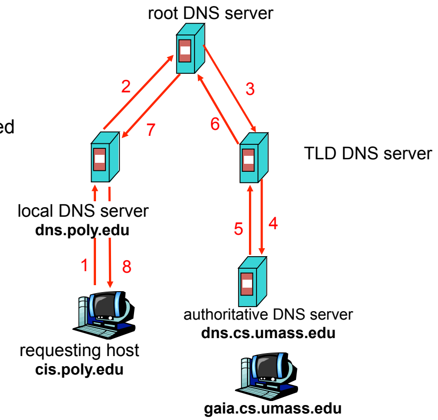
- ❒ Host enquiry is delegated to local DNS server
- ❒ Consider transactions 2 – 7 only
- ❒ contacted server replies with name of next server to contact
- ❒ "I don't know this name, but ask this server"

23

DNS name resolution **recursive** example

recursive query:

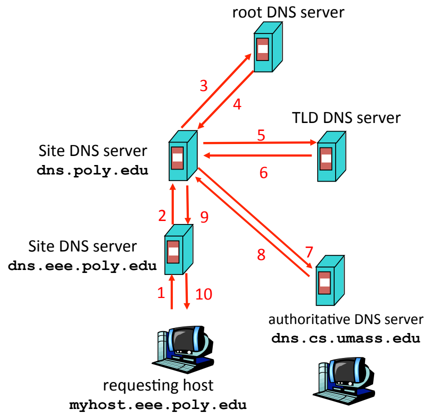
- ❒ puts burden of name resolution on contacted name server
- ❒ heavy load?



24

Recursive and Iterative Queries - **Hybrid** case

- Recursive query
  - Ask server to get answer for you
  - E.g., requests 1,2 and responses 9,10
- Iterative query
  - Ask server who to ask next
  - E.g., all other request-response pairs



25

DNS Caching

- Performing all these queries takes time
  - And all this before actual communication takes place
  - E.g., 1-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

26

Negative Caching

- Remember things that don't work
  - Misspellings like *www.cnn.comm* and *www.cnnn.com*
  - These can take a long time to fail the first time
  - Good to remember that they don't work
  - … so the failure takes less time the next time around
- But: negative caching is optional
  - And not widely implemented

27

Reliability

- DNS servers are replicated (primary/secondary)
  - Name service available if at least one replica is up
  - Queries can be load-balanced between replicas
- Usually, UDP used for queries
  - Need reliability: must implement this on top of UDP
  - Spec supports TCP too, but not always implemented
- Try alternate servers on timeout
  - Exponential backoff when retrying same server
- Same identifier for all queries
  - Don't care which server responds

28

DNS Measurements (MIT data from 2000)

- What is being looked up?
  - ~60% requests for A records
  - ~25% for PTR records
  - ~5% for MX records
  - ~6% for ANY records
- How long does it take?
  - Median ~100msec (but 90th percentile ~500msec)
  - 80% have no referrals; 99.9% have fewer than four
- Query packets per lookup: ~2.4
  - But this is misleading….

29

DNS Measurements (MIT data from 2000)

- Does DNS give answers?
  - ~23% of lookups fail to elicit an answer!
  - ~13% of lookups result in NXDOMAIN (or similar)
    - Mostly reverse lookups
  - Only ~64% of queries are successful!
    - *How come the web seems to work so well?*
- ~ 63% of DNS packets in unanswered queries!
  - Failing queries are frequently retransmitted
  - 99.9% successful queries have ≤2 retransmissions

30

DNS Measurements (MIT data from 2000)

- Top 10% of names accounted for ~70% of lookups
  - Caching should really help!
- 9% of lookups are unique
  - Cache hit rate can never exceed 91%
- Cache hit rates ~ 75%
  - But caching for more than 10 hosts doesn't add much

31

A Common Pattern…..

- Distributions of various metrics (file lengths, access patterns, etc.) often have two properties:
  - Large fraction of total metric in the top 10%
  - Sizable fraction (~10%) of total fraction in low values
- Not an exponential distribution
  - Large fraction is in top 10%
  - But low values have very little of overall total
- Lesson: have to pay attention to both ends of dist.
- Here: caching helps, but not a panacea

32

Moral of the Story

- If you design a highly resilient system, many things can be going wrong without you noticing it!

and this is a **good** thing

33

DNS and Security

- No way to verify answers
  - Opens up DNS to many potential attacks
  - DNSSEC fixes this
- Most obvious vulnerability: recursive resolution
  - Using recursive resolution, host must trust DNS server
  - When at Starbucks, server is under their control
  - And can return whatever values it wants
- More subtle attack: Cache poisoning
  - Those "additional" records can be anything!

34

Cache Poisoning

- Suppose you are a Bad Guy and you control the name server for foobar.com. You receive a request to resolve www.foobar.com and

```
;; 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

35

The Web – Precursor

Ted Nelson

- **1967**, Ted Nelson, Xanadu:
  - A world-wide publishing network that would allow information to be stored not as separate files but as connected literature
  - Owners of documents would be automatically paid via electronic means for the virtual copying of their documents
- Coined the term "Hypertext"
  - Influenced research community
    - Who then missed the web.....

36

The Web – History

Tim Berners-Lee

- CS grad turned physicist trying to solve real problem
  - Distributed access to data
- World Wide Web (WWW): a distributed database of "pages" linked through Hypertext Transport Protocol (HTTP)
  - First HTTP implementation - 1990
    - Tim Berners-Lee at CERN
  - HTTP/0.9 – 1991
    - Simple GET command for the Web
  - HTTP/1.0 –1992
    - Client/Server information, simple caching
  - HTTP/1.1 - 1996

37

Why Didn't CS Research Invent Web?

HTML is precisely what we were trying to PREVENT— ever-breaking links, links going outward only, quotes you can't follow to their origins, no version management, no rights management.

– Ted Nelson

Academics get paid for being clever, not for being right.

–Don Norman

38

Why So Successful?

- What do the web, youtube, fb have in common?
  - The ability to self-publish
- Self-publishing that is easy, independent, free
- No interest in collaborative and idealistic endeavor
  - People aren't looking for Nirvana (or even Xanadu)
  - People also aren't looking for technical perfection
- Want to make their mark, and find something neat
  - Two sides of the same coin, creates synergy
  - "Performance" more important than dialogue....

39

Web Components

- Infrastructure:
  - Clients
  - Servers
  - Proxies
- Content:
  - Individual objects (files, etc.)
  - Web sites (coherent collection of objects)
- Implementation
  - HTML: formatting content
  - URL: naming content
  - HTTP: protocol for exchanging content

    Any content not just HTML!

40

HTML: HyperText Markup Language

- A *Web page* has:
  - Base HTML file
  - Referenced objects (*e.g.*, images)
- HTML has several functions:
  - Format text
  - Reference images
  - Embed *hyperlinks* (HREF)

41

URL Syntax

protocol* : //*hostname*[: *port*] /*directorypath* /*resource

protocol	http, ftp, https, smtp, rtsp, *etc.*
hostname	DNS name, IP address
port	Defaults to protocol's standard port *e.g.* http: 80 https: 443
directory path	Hierarchical, reflecting file system
resource	Identifies the desired resource Can also extend to program executions: `http://us.f413.mail.yahoo.com/ym/ShowLetter?box=%40B%40Bulk&MsgId=2604_1744106_29699_1123_1261_0_28917_3552_1289957100&Search=&Nhead=f&YY=31454&order=down&sort=date&pos=0&view=a&head=b`

42

HyperText Transfer Protocol (HTTP)

- Request-response protocol
- Reliance on a global namespace
- Resource *metadata*
- *Stateless*
- ASCII format

% telnet www.icir.org 80
GET /jdoe/ HTTP/1.0
<blank line, i.e., CRLF>

43

Steps in HTTP Request

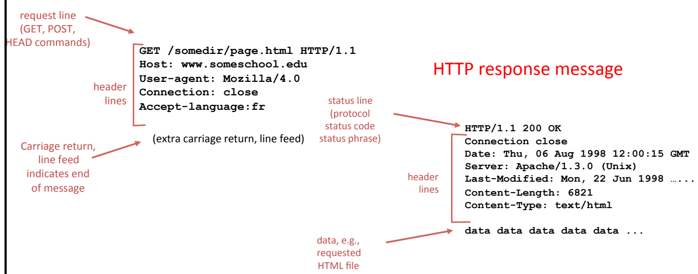
- HTTP Client initiates TCP connection to server
  - SYN
  - SYNACK
  - ACK
- Client sends HTTP request to server
  - Can be piggybacked on TCP's ACK
- HTTP Server responds to request
- Client receives the request, terminates connection
- TCP connection termination exchange

How many RTTs for a single request?

44

Client-Server Communication

- two types of HTTP messages: *request, response*
- HTTP request message: (GET POST HEAD ....)



45

Different Forms of Server Response

- Return a file
  - URL matches a file (*e.g.,* `/www/index.html`)
  - Server returns file as the response
  - Server generates appropriate response header
- Generate response dynamically
  - URL triggers a program on the server
  - Server runs program and sends output to client
- Return meta-data with no body

46

HTTP Resource Meta-Data

- Meta-data
  - Info *about* a resource, stored as a separate entity
- Examples:
  - Size of resource, last modification time, type of content
- Usage example: Conditional GET Request
  - Client requests object "**`If-modified-since`**"
  - If unchanged, "**`HTTP/1.1 304 Not Modified`**"
  - No body in the server's response, only a header

47

HTTP is *Stateless*

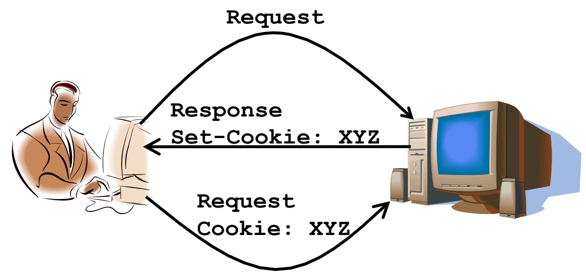
- Each request-response treated independently
  - Servers *not* required to retain state
- **Good**: Improves scalability on the server-side
  - Failure handling is easier
  - Can handle higher rate of requests
  - Order of requests doesn't matter
- **Bad**: Some applications need persistent state
  - Need to uniquely identify user or store temporary info
  - *e.g.,* Shopping cart, user profiles, usage tracking, …

48

State in a Stateless Protocol:

Cookies

- *Client-side* state maintenance
  - Client stores small(?) state on behalf of server
  - Client sends state in future requests to the server
- Can provide authentication



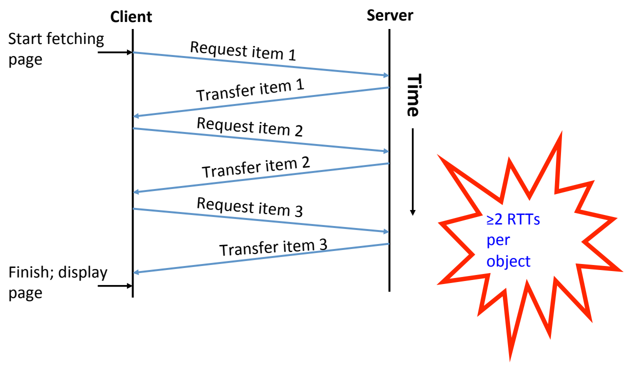
49

HTTP Performance

- Most Web pages have multiple objects
  - *e.g.,* HTML file and a bunch of embedded images
- How do you retrieve those objects (naively)?
  - *One item at a time*

50

Fetch HTTP Items: Stop & Wait



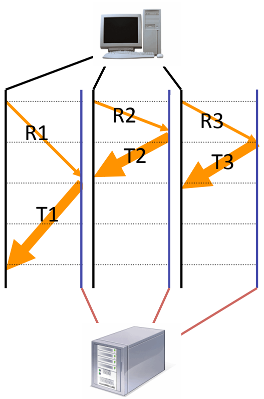
51

Improving HTTP Performance:

Concurrent Requests & Responses

- Use multiple connections *in parallel*
- Does not necessarily maintain order of responses

- Client = ☺
- Server = ☺
- Network = ☹ Why?

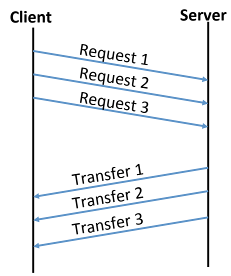


52

Improving HTTP Performance:

Pipelined Requests & Responses

- *Batch* requests and responses
  - Reduce connection overhead
  - Multiple requests sent in a single batch
  - Maintains order of responses
  - Item 1 always arrives before item 2
- How is this different from concurrent requests/responses?
  - Single TCP connection



53

Improving HTTP Performance:
Persistent Connections

- Enables multiple transfers per connection
  - Maintain TCP connection across multiple requests
  - Including transfers subsequent to current page
  - Client or server can tear down connection
- Performance advantages:
  - Avoid overhead of connection set-up and tear-down
  - Allow TCP to learn more accurate RTT estimate
  - Allow TCP congestion window to increase
  - i.e., leverage previously discovered bandwidth
- Default in HTTP/1.1

54

Scorecard: Getting n Small Objects

Time dominated by latency

- One-at-a-time: ~2n RTT
- Persistent: ~ (n+1)RTT
- M concurrent: ~2[n/m] RTT
- Pipelined: ~2 RTT
- Pipelined/Persistent: ~2 RTT first time, RTT later

55

Scorecard: Getting n Large Objects

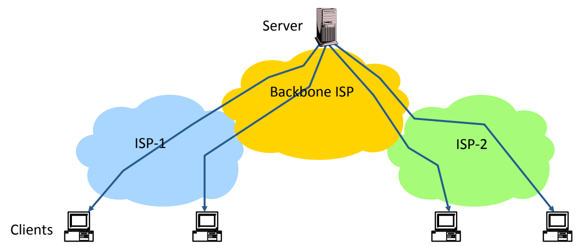
Time dominated by bandwidth

- One-at-a-time: ~ nF/B
- M concurrent: ~ [n/m] F/B
  - assuming shared with large population of users
- Pipelined and/or persistent: ~ nF/B
  - The only thing that helps is getting more bandwidth..

56

Improving HTTP Performance:
Caching

- Many clients transfer same information
  - Generates redundant server and network load
  - Clients experience unnecessary latency



57

Improving HTTP Performance:

Caching: How

- Modifier to GET requests:
  - `If-modified-since` – returns "not modified" if resource not modified since specified time
- Response header:
  - `Expires` – how long it's safe to cache the resource
  - `No-cache` – ignore all caches; always get resource directly from server

58

Improving HTTP Performance:

Caching: Why

- Motive for placing content closer to client:
  - User gets better response time
  - Content providers get happier users
    - Time is money, really!
  - Network gets reduced load
- Why does caching work?
  - Exploits *locality of reference*
- How well does caching work?
  - Very well, up to a limit
  - Large overlap in content
  - But many unique requests

59

Improving HTTP Performance:

Caching on the Client

Example: Conditional GET Request

- Return resource only if it has changed at the server
  - Save server resources!

Request from client to server:

```
GET /~ee122/fa07/ HTTP/1.1
Host: inst.eecs.berkeley.edu
User-Agent: Mozilla/4.03
If-Modified-Since: Sun, 27 Aug 2006 22:25:50 GMT
<CRLF>
```

- How?
  - Client specifies "if-modified-since" time in request
  - Server compares this against "last modified" time of desired resource
  - Server returns "304 Not Modified" if resource has not changed
  - …. or a "200 OK" with the latest version otherwise

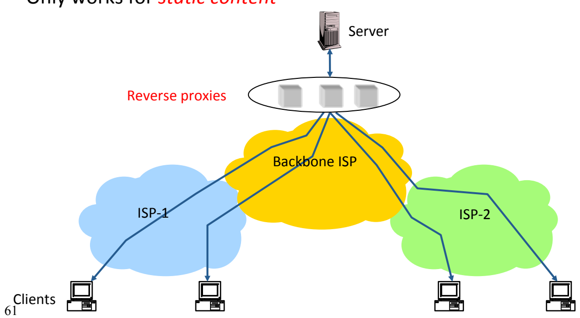
60

Improving HTTP Performance:

Caching with Reverse Proxies

Cache documents close to **server**
→ decrease server load

- Typically done by content providers
- Only works for *static content*

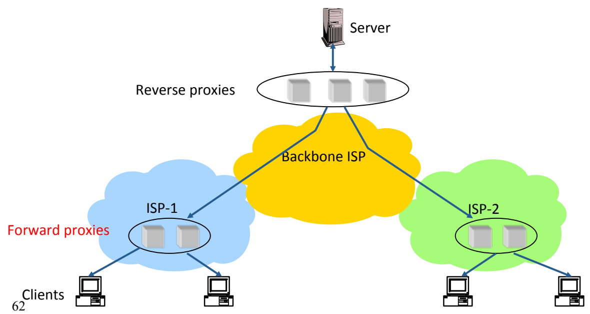


61

Improving HTTP Performance:
Caching with Forward Proxies

Cache documents close to **clients**
→ reduce network traffic and decrease latency

- Typically done by ISPs or corporate LANs



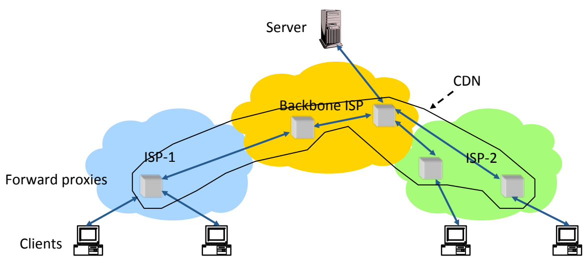
62

Improving HTTP Performance:
Caching w/ Content Distribution Networks

- Integrate forward and reverse caching functionality
  - One overlay network (usually) administered by one entity
  - *e.g.,* Akamai
- Provide document caching
  - **Pull:** Direct result of clients' requests
  - **Push:** Expectation of high access rate
- Also do some processing
  - Handle *dynamic* web pages
  - *Transcoding*

63

Improving HTTP Performance:
Caching with CDNs (cont.)



64

Improving HTTP Performance:
CDN Example – Akamai

- Akamai creates new domain names for each client content provider.
  - e.g., *a128.g.akamai.net*
- The CDN's DNS servers are authoritative for the new domains
- The client content provider modifies its content so that embedded URLs reference the new domains.
  - "Akamaize" content
  - e.g.: *http://www.cnn.com/image-of-the-day.gif* becomes *http://a128.g.akamai.net/image-of-the-day.gif*
- *Requests now sent to CDN's infrastructure…*

65

CDN examples

Hosting: Multiple Sites Per Machine

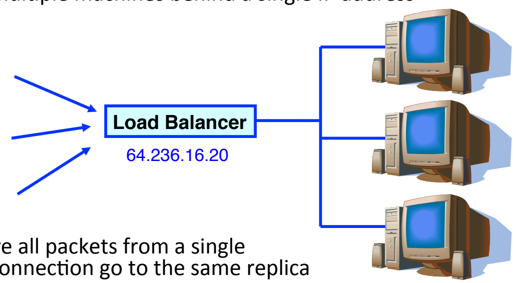
- Multiple Web sites on a single machine
  - Hosting company runs the Web server on behalf of multiple sites (*e.g.*, www.foo.com and www.bar.com)
- Problem: `GET /index.html`
  - `www.foo.com/index.html` or `www.bar.com/index.html`?
- Solutions:
  - Multiple server processes on the same machine
    - Have a separate IP address (or port) for each server
  - Include site name in HTTP request
    - Single Web server process with a single IP address
    - Client includes "Host" header (*e.g.,* `Host: www.foo.com`)
    - *Required header* with HTTP/1.1

Hosting: Multiple Machines Per Site

- Replicate popular Web site across many machines
  - Helps to handle the load
  - Places content closer to clients
- Helps when content isn't cacheable
- Problem: Want to direct client to particular replica
  - Balance load across server replicas
  - Pair clients with nearby servers

Multi-Hosting at Single Location

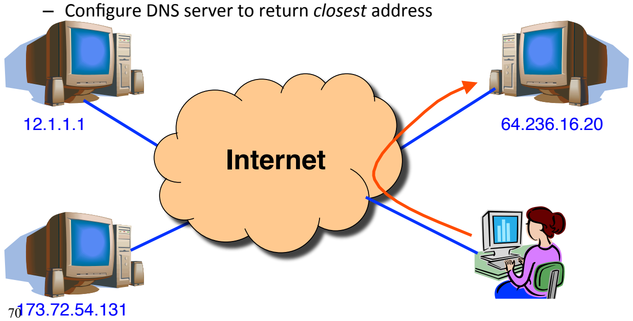
- Single IP address, multiple machines
  - Run multiple machines behind a single IP address



  - Ensure all packets from a single TCP connection go to the same replica

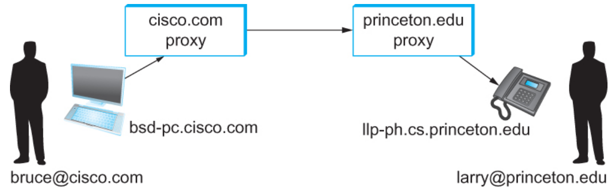
Multi-Hosting at Several Locations

- Multiple addresses, multiple machines
  - Same name but different addresses for all of the replicas
  - Configure DNS server to return *closest* address



70

SIP - VoIP



Establishing communication through SIP proxies.

71

SIP?

- SIP – bringing the fun/complexity of telephony to the Internet
  - User location
  - User availability
  - User capabilities
  - Session setup
  - Session management
    - (e.g. "call forwarding")

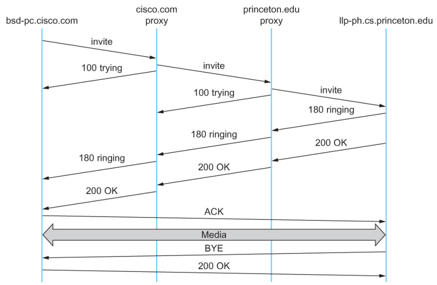
72

H.323 – ITU

- Why have one standard when there are at least two….
- The full H.323 is hundreds of pages
  - The protocol is known for its complexity – an ITU hallmark
- SIP is not much better

73

Multimedia Applications

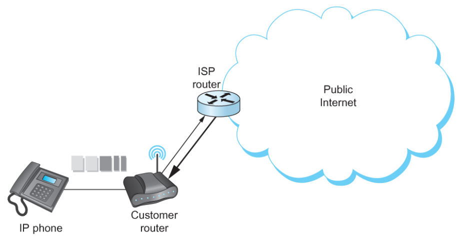


Message flow for a basic SIP session

74

The (still?) missing piece:

Resource Allocation for Multimedia Applications

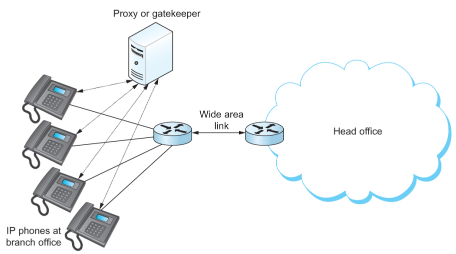


I can 'differentiate' VoIP from data but…
I can only control data going into the Internet

75

Multimedia Applications

- Resource Allocation for Multimedia Applications



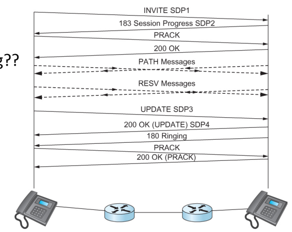
Admission control using session control protocol.

76

Resource Allocation for Multimedia Applications

Coming soon… ~~1995~~
~~2000~~
~~2010~~
who are we kidding??

Co-ordination of SIP signaling and resource reservation.



So where does it happen?
Inside single institutions or *domains of control…..*
(Universities, Hospitals, big corp…)

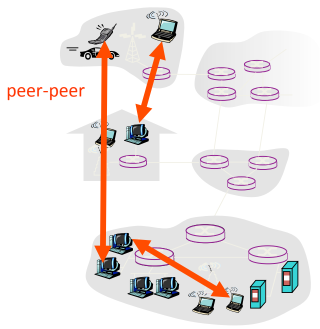
What about my aDSL/CABLE/etc it combines voice and data?
Phone company **controls** the multiplexing on the line
and throughout their own network too……

77

Pure P2P architecture

- *no* always-on server
- arbitrary end systems directly communicate
- peers are intermittently connected and change IP addresses

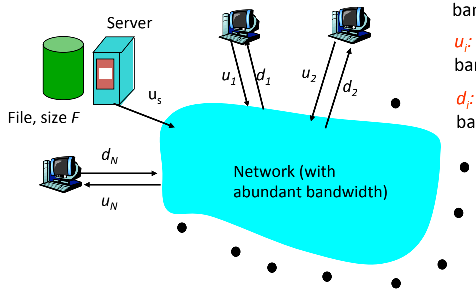
- Three topics:
  - File distribution
  - Searching for information
  - Case Study: Skype



78

File Distribution: Server-Client vs P2P

Question : How much time to distribute file from one server to *N peers*?



u_s: server upload bandwidth

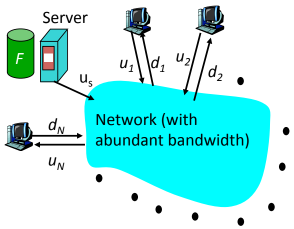
u_i: peer i upload bandwidth

d_i: peer i download bandwidth

79

File distribution time: server-client

- server sequentially sends N copies:
  - NF/u_s time
- client i takes F/d_i time to download



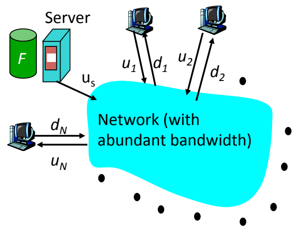
Time to distribute *F* to *N* clients using client/server approach $= d_{cs} = \max\{ NF/u_s, F/\min_i(d_i) \}$

increases linearly in N (for large N)

80

File distribution time: P2P

- server must send one copy: F/u_s time
- client i takes F/d_i time to download
- NF bits must be downloaded (aggregate)
  - fastest possible upload rate: $u_s + \sum u_i$

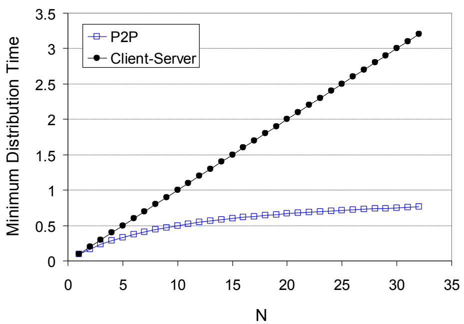


$$d_{P2P} = \max\{ F/u_s, F/\min_i(d_i), NF/(u_s + \sum_i u_i) \}$$

81

Server-client vs. P2P: example

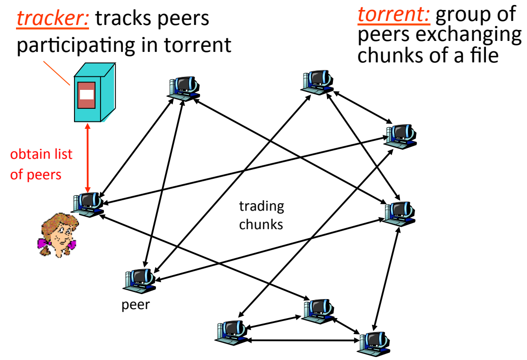
Client upload rate = u, F/u = 1 hour, u_s = 10u, $d_{min} \geq u_s$



82

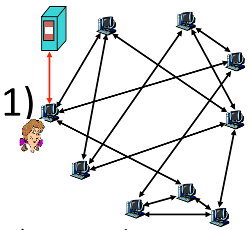
File distribution: BitTorrent

❒ P2P file distribution



83

BitTorrent (1)



- file divided into 256KB *chunks*.
- peer joining torrent:
  - has no chunks, but will accumulate them over time
  - registers with tracker to get list of peers, connects to subset of peers ("neighbors")
- while downloading, peer uploads chunks to other peers.
- peers may come and go
- once peer has entire file, it may (selfishly) leave or (altruistically) remain

84

BitTorrent (2)

Pulling Chunks

- at any given time, different peers have different subsets of file chunks
- periodically, a peer (Alice) asks each neighbor for list of chunks that they have.
- Alice sends requests for her missing chunks
  - rarest first

Sending Chunks: tit-for-tat

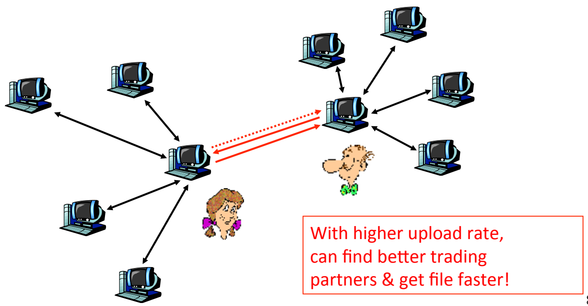
❒ Alice sends chunks to four neighbors currently sending her chunks *at the highest rate*
  - re-evaluate top 4 every 10 secs

❒ every 30 secs: randomly select another peer, starts sending chunks
  - newly chosen peer may join top 4
  - "optimistically unchoke"

85

BitTorrent: Tit-for-tat

(1) Alice "optimistically unchokes" Bob
(2) Alice becomes one of Bob's top-four providers; Bob reciprocates
(3) Bob becomes one of Alice's top-four providers



86

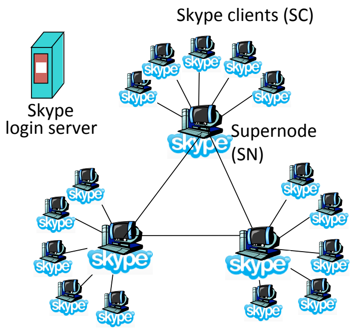
Distributed Hash Table (DHT)

- DHT = distributed P2P database
- Database has (key, value) pairs;
  - key: ss number; value: human name
  - key: content type; value: IP address
- Peers query DB with key
  - DB returns values that match the key
- Peers can also insert (key, value) peers

87

P2P Case study: Skype

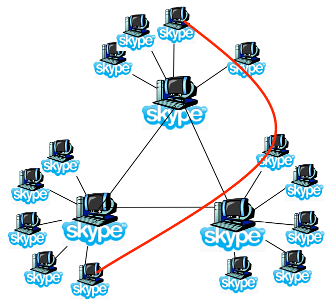
- inherently P2P: pairs of users communicate.
- proprietary application-layer protocol (inferred via reverse engineering)
- hierarchical overlay with SNs
- Index maps usernames to IP addresses; distributed over SNs



88

Peers as relays

- Problem when both Alice and Bob are behind "NATs".
  - NAT prevents an outside peer from initiating a call to insider peer
- Solution:
  - Using Alice's and Bob's SNs, Relay is chosen
  - Each peer initiates session with relay.
  - Peers can now communicate through NATs via relay



89