Chair for Network Architectures and Services – Prof. Carle
Department of Computer Science
TU München

Master Course
Computer Networks
IN2097

Prof. Dr.-Ing. Georg Carle
Christian Grothoff, Ph.D.

Chair for Network Architectures and Services
Institut für Informatik
Technische Universität München
http://www.net.in.tum.de

Chapter 2: Application layer

- Principles of network applications
- Web and HTTP
- DNS
- P2P applications
- Summary

Chapter 2: Application Layer

Our goals:

- conceptual, implementation aspects of network application protocols
    - transport-layer service models
    - client-server paradigm
    - peer-to-peer paradigm
- learn about protocols by examining popular application-level protocols
    - HTTP
    - DNS
- programming network applications
    - socket API

Some network applications

- e-mail
- web
- instant messaging
- remote login
- P2P file sharing
- multi-user network games
- streaming stored video clips
- voice over IP
- real-time video conferencing
- grid computing

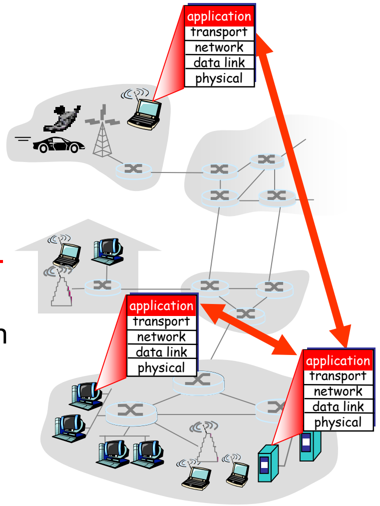
Creating a network application

write programs that

- run on (different) *end systems*
- communicate over network
- e.g., web server software communicates with browser software

No need to write software for network-core devices

- Network-core devices do not run user applications
- applications on end systems allows for rapid application development, propagation



⇨ think whether a counter-example exists:
what would be benefits if you could program your router?

Chapter 2: Application layer

- ❑ **Principles of network applications**
- ❑ Web and HTTP
- ❑ DNS
- ❑ P2P applications
- ❑ Summary

Application architectures

- ❑ Client-server
- ❑ Peer-to-peer (P2P)
- ❑ Hybrid of client-server and P2P

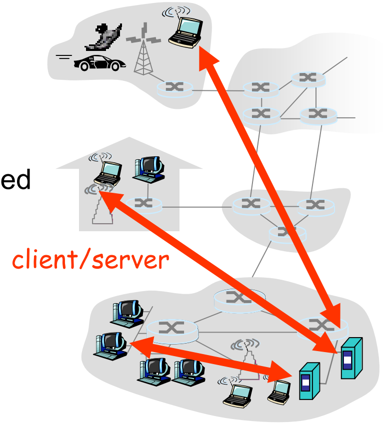
Client-server architecture

server:

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

clients:

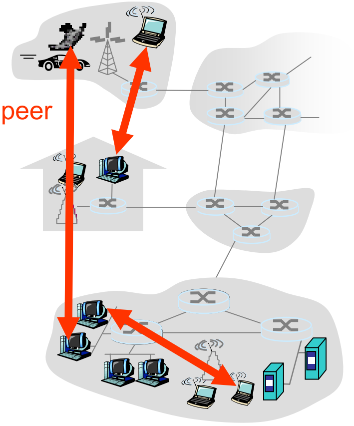
- communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do not communicate directly with each other



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



Hybrid of client-server and P2P

Skype
- voice-over-IP P2P application
- centralized server: authenticates user, finds 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

---

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

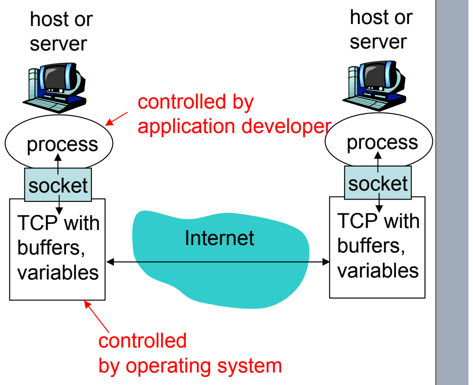
Server process: process that waits to be contacted

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

---

Sockets

- 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



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

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

Application-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

What transport service does an application need?

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

Timing
- some applications (e.g., Internet telephony, interactive games) require low delay to be “effective”
- frequently the applications also need timestamps (e.g. specifying playout time)

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

Security
- Encryption, data integrity, …

Transport service requirements of common apps

Application	Data loss	Throughput	Time Sensitive
file transfer	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5kbps-1Mbps video:10kbps-5Mbps	yes, 100’s msec
stored audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	few kbps up	yes, 100’s msec
instant messaging	no loss	elastic	yes and no

Internet transport protocols services

TCP service:

- *connection-oriented:* setup required between client and server processes
- *reliable transport* between sending and receiving process
- *flow control:* sender won’t overwhelm receiver
- *congestion control:* throttle sender when network overloaded
- *does not provide:* timing, minimum throughput guarantees, security

UDP service:

- unreliable data transfer between sending and receiving process
- does not provide: connection setup, reliability, flow control, congestion control, timing, throughput guarantee, or security

Q: why bother? Why is there a UDP?

Internet apps: application, transport protocols

Application	Application layer protocol	Underlying transport protocol
e-mail	SMTP [RFC 2821]	TCP
remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
file transfer	FTP [RFC 959]	TCP
streaming multimedia	HTTP (e.g., Youtube), RTP [RFC 1889]	TCP or UDP
Internet telephony	SIP, RTP, proprietary (e.g., Skype)	typically UDP

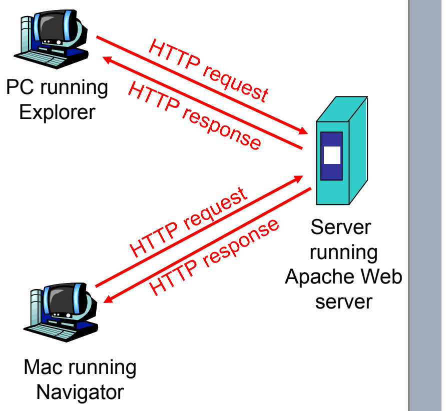
Chapter 2: Application layer

- Principles of network applications
- **Web and HTTP**
- DNS
- P2P applications
- Summary

HTTP overview

HTTP: hypertext transfer protocol

- Web’s application layer protocol
- client/server model
  - *client:* browser that requests, receives, “displays” Web objects
  - *server:* Web server sends objects in response to requests



HTTP overview (continued)

HTTP uses TCP:

- client initiates TCP connection (creates socket) to server at port 80
- server accepts TCP connection from client
- HTTP messages (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server)
- http1.0: TCP connection closed after HTTP response

HTTP is “stateless”

- server maintains no information about past client requests

aside

Protocols that maintain “state” are complex!

- past history (state) must be maintained
- if server/client crashes, their views of “state” may be inconsistent, must be reconciled

---

HTTP connections

Nonpersistent HTTP (v1.0)

- At most one object is sent over a TCP connection.

Persistent HTTP (v1.1)

- Multiple objects can be sent over single TCP connection between client and server.

Nonpersistent HTTP

Suppose user enters URL

`www.someSchool.edu/someDepartment/home.index` (contains text, references to 10 jpeg images)

1a. **HTTP** client initiates TCP connection to HTTP server (process) at www.someSchool.edu on port 80

1b. HTTP server at host www.someSchool.edu waiting for TCP connection at port 80. “accepts” connection, notifying client

2. HTTP client sends HTTP *request message* (containing URL) into TCP connection socket. Message indicates that client wants object someDepartment/home.index

3. HTTP server receives request message, forms *response message* containing requested object, and sends message into its socket

time

Nonpersistent HTTP (cont.)

4. HTTP server closes TCP connection.

5. HTTP client receives response message containing html file, displays html. Parsing html file, finds 10 referenced jpeg objects

6. Steps 1-5 repeated for each of 10 jpeg objects

time

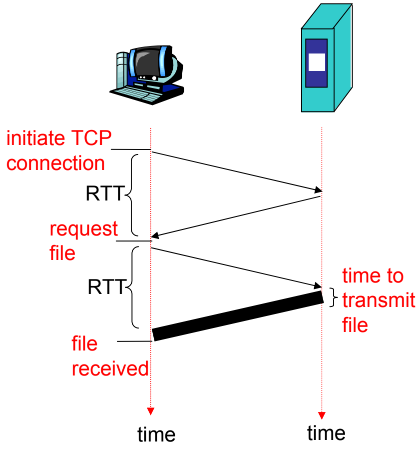
Non-Persistent HTTP: Response time

Definition of RTT: time for a small packet to travel from client to server and back.

Response time:

- one RTT to initiate TCP connection
- one RTT for HTTP request and first few bytes of HTTP response to return
- file transmission time

total = 2RTT+ transmit time



Persistent HTTP

Nonpersistent HTTP issues:

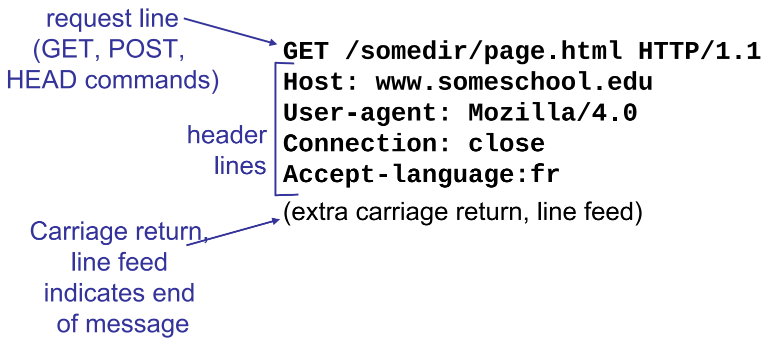
- requires 2 RTTs per object
- OS overhead for *each* TCP connection
- browsers often open parallel TCP connections to fetch referenced objects

Persistent HTTP

- server leaves connection open after sending response
- subsequent HTTP messages between same client/server sent over open connection
- client sends requests as soon as it encounters a referenced object
- as little as one RTT for all the referenced objects

HTTP request message

- two types of HTTP messages: *request, response*
- HTTP request message:
  - ASCII (human-readable format)



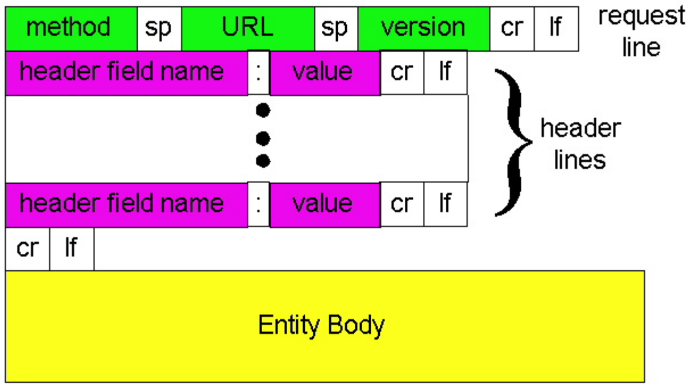
```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language:fr
```

(extra carriage return, line feed)

HTTP request message: general format

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language:fr
```

(extra carriage return, line feed)



Uploading form input

Post method:

- Web page often includes form input
- Input is uploaded to server in entity body

URL method:

- Uses GET method
- Input is uploaded in URL field of request line:

`www.somesite.com/animalsearch?monkeys&banana`

Method types

HTTP/1.0

- GET
- POST
- HEAD
  - asks server to leave requested object out of response

HTTP/1.1

- GET, POST, HEAD
- PUT
  - uploads file in entity body to path specified in URL field
- DELETE
  - deletes file specified in the URL field

HTTP response message

status line (protocol status code status phrase)

```
HTTP/1.1 200 OK
```

header lines

```
Connection: close
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 …...
Content-Length: 6821
Content-Type: text/html
```

data, e.g., requested HTML file

```
data data data data data ...
```

HTTP response status codes

- In first line in server: client response message
- A few sample codes:

`200 OK`
  - request succeeded, requested object later in this message

`301 Moved Permanently`
  - requested object moved, new location specified later in this message (Location:)

`400 Bad Request`
  - request message not understood by server

`404 Not Found`
  - requested document not found on this server

`505 HTTP Version Not Supported`

Trying out HTTP (client side) for yourself

1. Telnet to your favorite Web server:

telnet cis.poly.edu 80

Opens TCP connection to port 80
(default HTTP server port) at cis.poly.edu.
Anything typed in sent
to port 80 at cis.poly.edu

2. Type in a GET HTTP request:

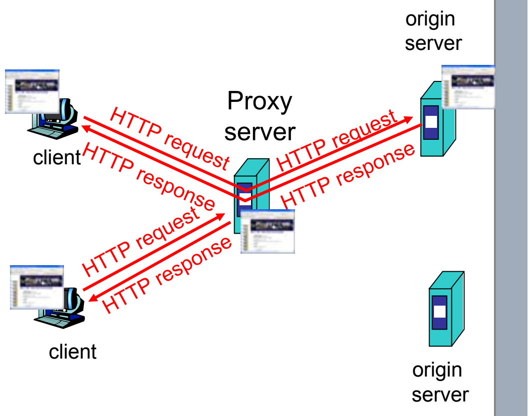
GET /~ross/ HTTP/1.1
Host: cis.poly.edu

By typing this in (hit carriage
return twice), you send
this minimal (but complete)
GET request to HTTP server

3. Look at response message sent by HTTP server!

Web caches (proxy server)

- Goal: satisfy client request without involving origin server
- user sets browser: Web accesses via cache
- browser sends all HTTP requests to cache
  - object in cache: cache returns object
  - else cache requests object from origin server, then returns object to client



More about Web caching

- cache acts as both client and server
- typically cache is installed by ISP (university, company, residential ISP)

Why Web caching?

- reduce response time for client request
- reduce traffic on an institution's access link.
- Internet dense with caches: enables "poor" content providers to effectively deliver content (but so does P2P file sharing)

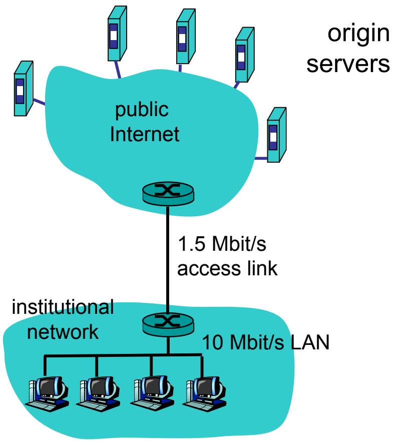
Example

Assumptions

- average object size = 100.000 bits
- avg. request rate from institution's browsers to origin servers = 15/sec
- delay from institutional router to any origin server and back to router = 2 sec

Consequences

- traffic intensity (utilization) on LAN = 15%
- traffic intensity (utilization) on access link = 100%
- total delay
  = Internet delay + access delay + LAN delay
  = 2 sec + minutes + milliseconds



Example (cont)

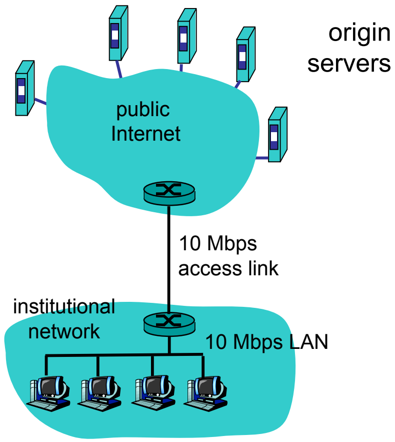
possible solution

- increase bandwidth of access link to, say, 10 Mbps

consequence

- utilization on LAN = 15%
- utilization on access link = 15%
- Total delay = Internet delay + access delay + LAN delay

  = 2 sec + msecs + msecs
- often a costly upgrade



Example (cont)

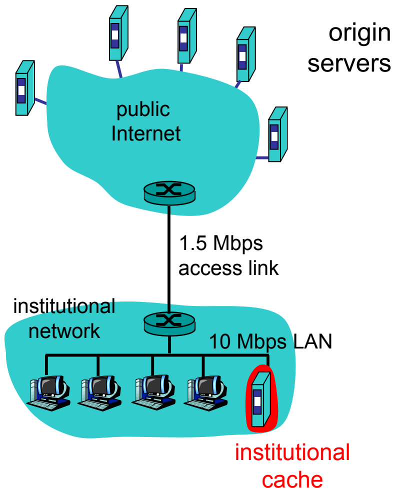
possible solution: install cache

- suppose hit rate is 0.4

consequence

- 40% requests will be satisfied almost immediately
- 60% requests satisfied by origin server
- utilization of access link reduced to 60%, resulting in negligible delays (say 10 msec)
- total average delay

  = 60%*{ Internet delay
  + access delay
  + LAN delay}
  + 40% * milliseconds
  = 0.6*(2.01) sec
  + 0.4*milliseconds
  ≈ 1.2 secs

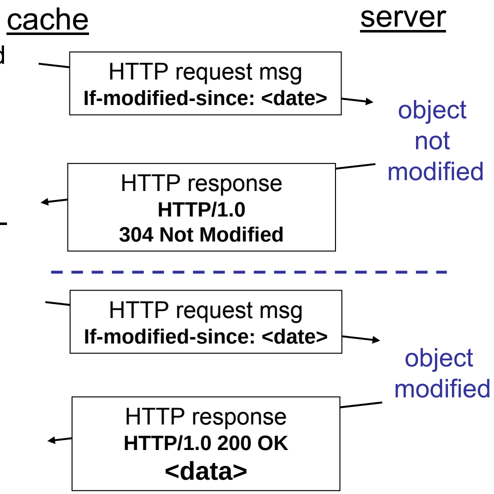


Conditional GET

- Goal: don't send object if cache has up-to-date cached version
- cache: specify date of cached copy in HTTP request

  `If-modified-since: <date>`
- server: response contains no object if cached copy is up-to-date:

  `HTTP/1.0 304 Not Modified`



cache

server

HTTP request msg
If-modified-since: <date>

object not modified

HTTP response
HTTP/1.0
304 Not Modified

HTTP request msg
If-modified-since: <date>

object modified

HTTP response
HTTP/1.0 200 OK
<data>

Chapter 2: Application layer

- Principles of network applications
- Web and HTTP
- **DNS**
- P2P applications
- Summary

DNS: Domain Name System

People: many identifiers:

- Social Secuity Number, name, passport #

Internet hosts, routers:

- IP address (32 bit) - used for addressing datagrams
- “name”, e.g., ww.yahoo.com - used by humans

Q: map between IP addresses and name ?

Domain Name System:

- *distributed database* implemented in hierarchy of many *name servers*
- *application-layer protocol* host, routers, name servers to communicate to *resolve* names (address/name translation)
  - note: core Internet function, implemented as application-layer protocol
  - complexity at network’s “edge”

DNS

Why not centralize DNS?

- single point of failure
- traffic volume
- distant centralized database
- maintenance

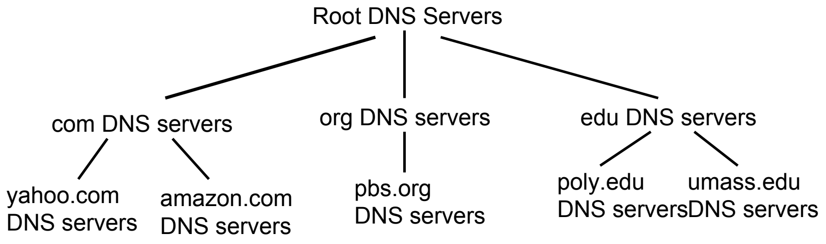
doesn’t *scale!*

DNS services

- hostname to IP address translation
- host aliasing
  - canonical name
  - alias names
- mail server aliasing
  - mnemonic host name desired
  - MX record allows mnemonic host name reused for mail server
- load distribution
  - replicated Web servers: set of IP addresses for one canonical name

---

Distributed, Hierarchical Database



Root DNS Servers

com DNS servers

org DNS servers

edu DNS servers

yahoo.com DNS servers

amazon.com DNS servers

pbs.org DNS servers

poly.edu DNS servers

umass.edu DNS servers

Client wants IP for www.amazon.com; 1st approx:

- client queries a root server to find com DNS server
- client queries com DNS server to get amazon.com DNS server
- client queries amazon.com DNS server (authorative DNS server – configured by original source) to get IP address for www.amazon.com

---

DNS: Root name servers

- contacted by local name server that can not resolve name
- root name server:
  - contacts authoritative name server if name mapping not known
  - gets mapping
  - returns mapping to local name server

13 root name servers worldwide

TLD and Authoritative Servers

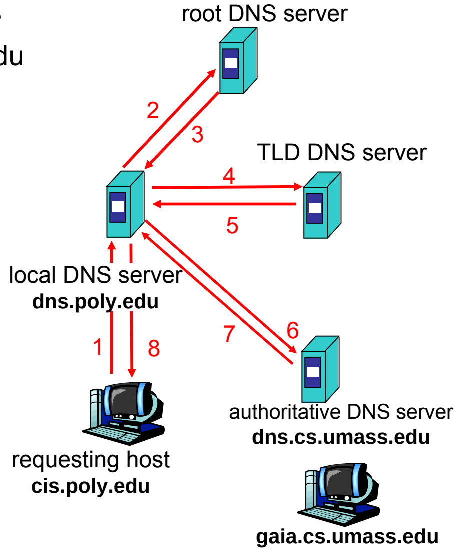
- Top-level domain (TLD) servers:
  - responsible for com, org, net, edu, etc, and all top-level country domains uk, fr, ca, jp.
  - organisations hosting TLD servers:
    - Network Solutions maintains servers for com TLD
    - Educause for edu TLD
- Authoritative DNS servers:
  - organization’s DNS servers, providing authoritative hostname to IP mappings for organization’s servers (e.g., Web, mail).
  - can be maintained by organization or service provider

Local Name Server

- does not strictly belong to hierarchy
- each ISP (residential ISP, company, university) has one.
  - also called “default name server”
- when host makes DNS query, query is sent to its local DNS server
  - acts as proxy, forwards query into hierarchy

DNS name resolution example

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



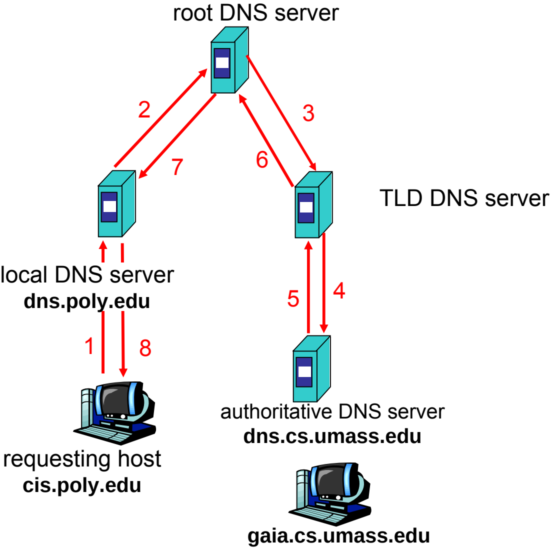
iterated query:

- contacted server replies with name of server to contact
- “I don’t know this name, but ask this server”

DNS name resolution example

recursive query:

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



DNS: caching and updating records

- once (any) name server learns mapping, it *caches* mapping
  - cache entries timeout (disappear) after some time
  - TLD servers typically cached in local name servers
    - Thus root name servers not often visited
- update/notify mechanisms
  - RFC 2136
  - http://www.ietf.org/html.charters/dnsind-charter.html
  - „notify“ mechanism: primary sends a message to known secondaries. for fast convergence of servers

---

DNS records

DNS: distributed db storing resource records (RR)

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

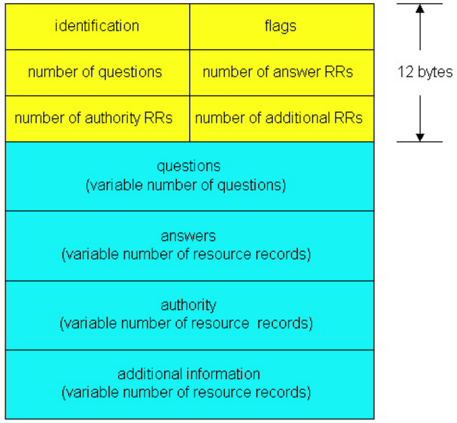
- Type=A
  - name is hostname
  - value is IP address
- Type=NS
  - `name` is domain (e.g. foo.com)
  - `value` is hostname of authoritative name server for this domain
- Type=CNAME
  - name is alias name for some "canonical" (the real) name
  - e.g.: www.ibm.com is really servereast.backup2.ibm.com (canonical name)
- Type=MX
  - value is name of mailserver associated with name

DNS protocol, messages

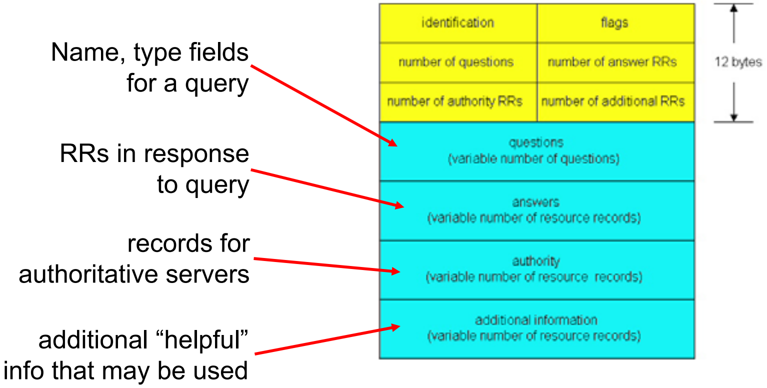
DNS protocol : *query* and *reply* messages, both with same *message format*

message header

- identification: 16 bit # for query, reply to query uses same #
- flags:
  - query or reply
  - recursion desired
  - recursion available
  - reply is authoritative



DNS protocol, messages


Name, type fields for a query

RRs in response to query

records for authoritative servers

additional "helpful" info that may be used

Inserting records into DNS

- example: new startup “Network Utopia”
- register name networkuptopia.com at *DNS* *registrar* (e.g., Network Solutions)
  - provide names, IP addresses of authoritative name server (primary and secondary)
  - registrar inserts two RRs into com TLD server:

```
(networkutopia.com, dns1.networkutopia.com,
  NS)
(dns1.networkutopia.com, 212.212.212.1, A)
```

- create authoritative server
  Type A record for www.networkuptopia.com
  Type MX record for networkutopia.com

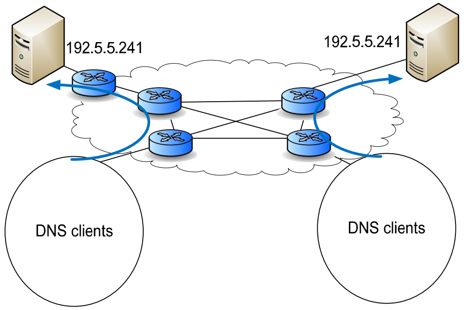
DNS Root Servers

- 13 root servers (A to M)
- But number of physical servers in total is higher
- and increasing:
  - 191 by Oct. 2009
  - 229 by Oct. 2010

Source: http://root-servers.org/

DNS and IP Anycast

- Multiple servers can be made reachable under the same IP address
- Via *IP anycast*
- E.g. F-root server (IPv4: 192.5.5.241; IPv6: 2001:500:2f::f)



- IP anycast used for DNS since 2002 for root servers and many TLDs

➔ High robustness

➔ New servers can be easily added without updating the DNS clients.

DNS Caching

- TTL not specified in the standard (RFC 1034-1035)
- But in practice TTLs often up to 24 hours
- Records for TLDs are provided by root servers and typically stored even for 48 hours

- Caching typically improves lookup performance
- Caching relieves upper nodes in the hierarchy (root + TLDs)

- Massive caching makes it difficult to:
  - Dynamically react to current load
  - Migrate services

  ➔ TTLs of 60 s are typical today (e.g. amazon.com)

Example: DNS with Low TTLs

- e.g. amazon.com

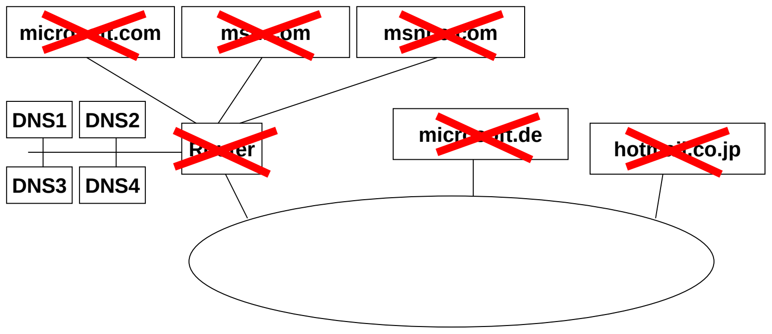
```
user@host:~$ dig amazon.com
; <<>> DiG 9.6.1-P2 <<>> amazon.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 42197
;; flags: qr rd ra; QUERY: 1, ANSWER: 3, AUTHORITY: 7,
   ADDITIONAL: 9

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

;; ANSWER SECTION:
amazon.com.         60      IN      A       72.21.210.250
amazon.com.         60      IN      A       207.171.166.252
amazon.com.         60      IN      A       72.21.207.65
```

Dependency on DNS

- DoS-Attack targeting Microsoft in January 2001
  - First: router problem ➔ Microsoft's websites and services were down on January 23rd 2001
  - The damage was surprisingly large



Dependency on DNS

- Web servers are be running
- But DNS failure leads to service failure

➔ Need to deploy multiple DNS authorative servers

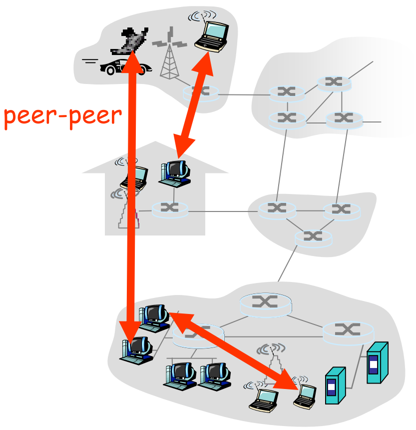
➔ In different networks

Chapter 2: Application layer

- Principles of network applications
- Web and HTTP
- DNS
- **P2P applications**
- Summary

Pure P2P architecture

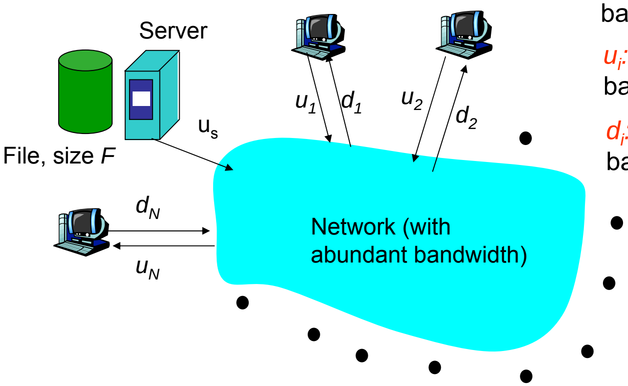
- *no* always-on server
- arbitrary end systems directly communicate
- peers are intermittently connected and change IP addresses



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

File Distribution: Server-Client vs P2P

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



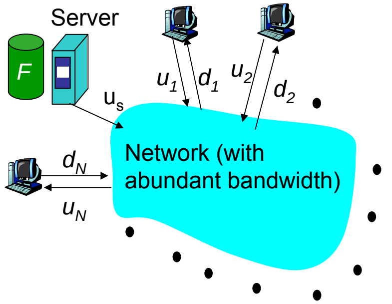
u_s: server upload bandwidth

u_i: peer i upload bandwidth

d_i: peer i download bandwidth

File distribution time: server-client

- server sequentially sends N copies. distribution time is at least: NF/u_s time
- client i takes F/d_i time to download minimum download time: F/d_{min}



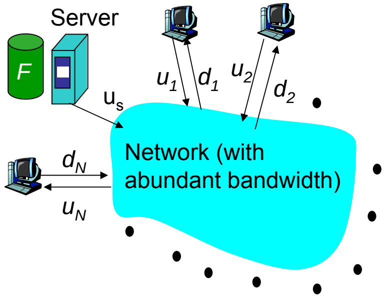
Time to distribute *F* to *N* clients using client/server approach $= d_{cs} = \max \{ NF/u_s, F/d_{min} \}$

increases linearly in N (for large N)

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)

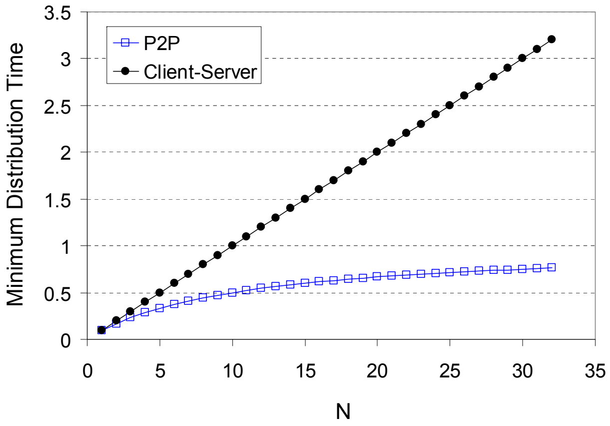
fastest possible upload rate: $u_s + \Sigma u_i$



$$d_{P2P} = \max \{ F/u_s, F/d_{min}, NF/(u_s + \Sigma u_i) \}$$

Server-client vs. P2P: example

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

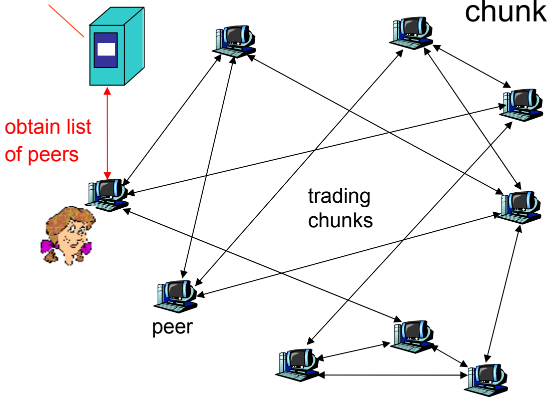


File distribution: BitTorrent

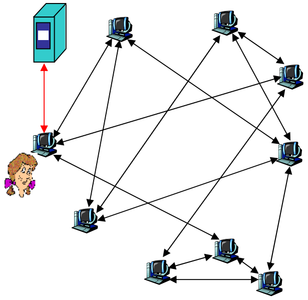
- P2P file distribution

tracker: tracks peers participating in torrent

torrent: group of peers exchanging chunks of a file



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

---

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

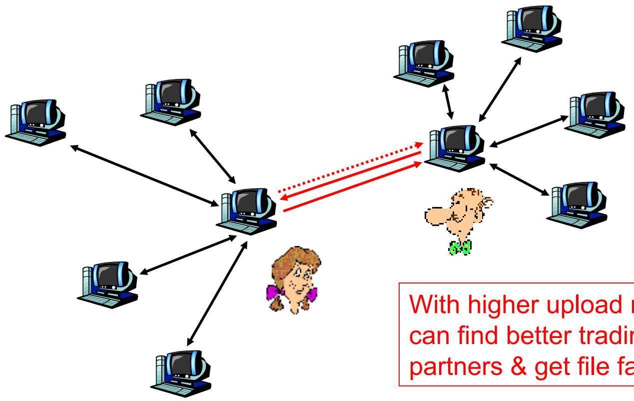
- Alice sends chunks to four neighbors currently sending her chunks *at the highest rate*
  - re-evaluate top 4 every 10 secs
- every 30 secs: randomly select another peer, starts sending chunks
  - newly chosen peer may join top 4
  - "optimistically unchoke"

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



With higher upload rate, can find better trading partners & get file faster!

Distributed Hash Table (DHT)

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

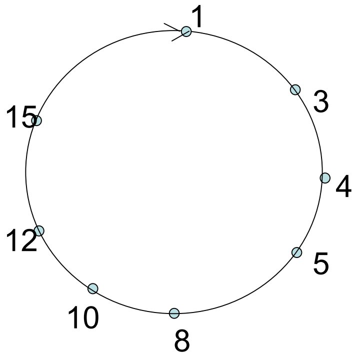
DHT Identifiers

- Assign integer identifier to each peer in range $[0,2^n-1]$.
  - Each identifier can be represented by n bits.
- Require each key to be an integer in same range.
- To get integer keys, hash original key.
  - eg, key = h(“Led Zeppelin IV”)
  - This is why they call it a distributed “hash” table

How to assign keys to peers?

- Central issue:
  - Assigning (key, value) pairs to peers.
- Rule: assign key to the peer that has the closest ID.
- Convention in lecture: closest is the immediate successor of the key.
- Example: n=4; peers: 1,3,4,5,8,10,12,14;
  - key = 13, then successor peer = 14
  - key = 15, then successor peer = 1

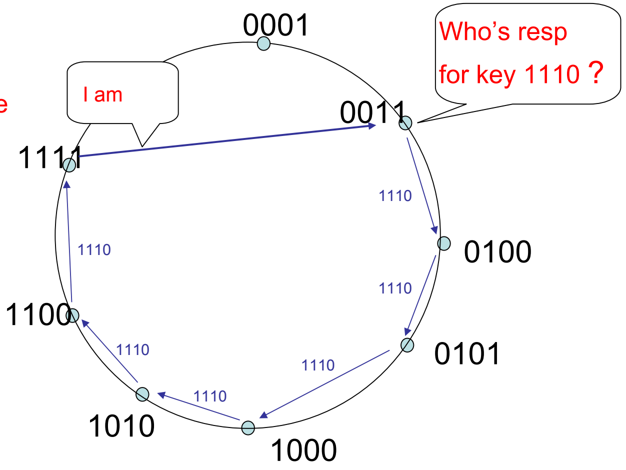
Circular DHT (1)



- Each peer *only* aware of immediate successor and predecessor.
- “Overlay network”

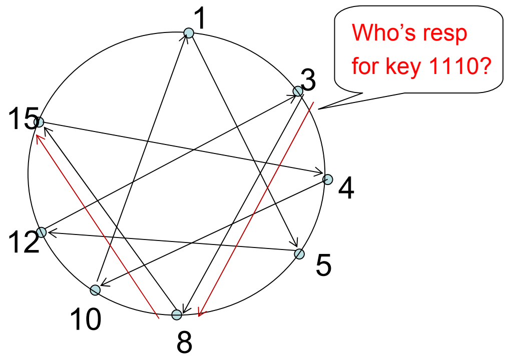
Circle DHT (2)

O(N) messages on avg to resolve query, when there are N peers



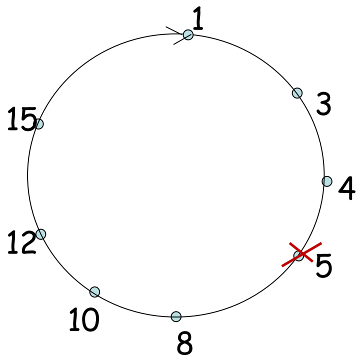
Define closest as closest successor

Circular DHT with Shortcuts



- Each peer keeps track of IP addresses of predecessor, successor, short cuts.
- Shortcuts reduce required number of query messages (e.g. from 6 to 2).
- Possible to design shortcuts so O(log N) neighbors, O(log N) messages in query

Peer Churn

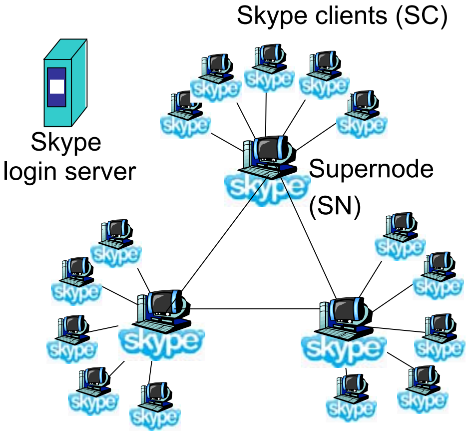


- To handle peer churn, require each peer to know the IP address of its two successors.
- Each peer periodically pings its two successors to see if they are still alive.

- Peer 5 abruptly leaves
- Peer 4 detects; makes 8 its immediate successor; asks 8 who its immediate successor is; makes 8’s immediate successor its second successor.
- What if peer 13 wants to join?

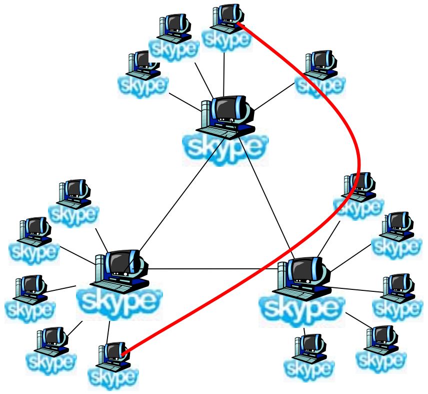
P2P Case study: Skype

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



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 Supernodes, Relay is chosen
  - Each peer initiates session with relay.
  - Peers can now communicate through NATs via relay



Chapter 2: Application layer

- Principles of network applications
- Web and HTTP
- DNS
- P2P applications
- **Summary**

Chapter 2: Summary

network application level issues

- application architectures
  - client-server
  - P2P
  - hybrid
- application service requirements:
  - reliability, bandwidth, delay
- Internet transport service model
  - connection-oriented, reliable: TCP
  - unreliable, datagrams: UDP
- specific protocols:
  - HTTP
  - DNS
  - P2P: BitTorrent, Skype
- socket programming

Chapter 2: Summary

Most importantly: learned about *protocols*

- typical request/reply message exchange:
  - client requests info or service
  - server responds with data, status code
- message formats:
  - headers: fields giving info about data
  - data: info being communicated
- *Important themes:*
- control vs. data messages
  - in-band, out-of-band
- centralized vs. decentralized
- stateless vs. stateful
- reliable vs. unreliable message transfer
- “complexity at network edge”