



UNIVERSITY OF COPENHAGEN

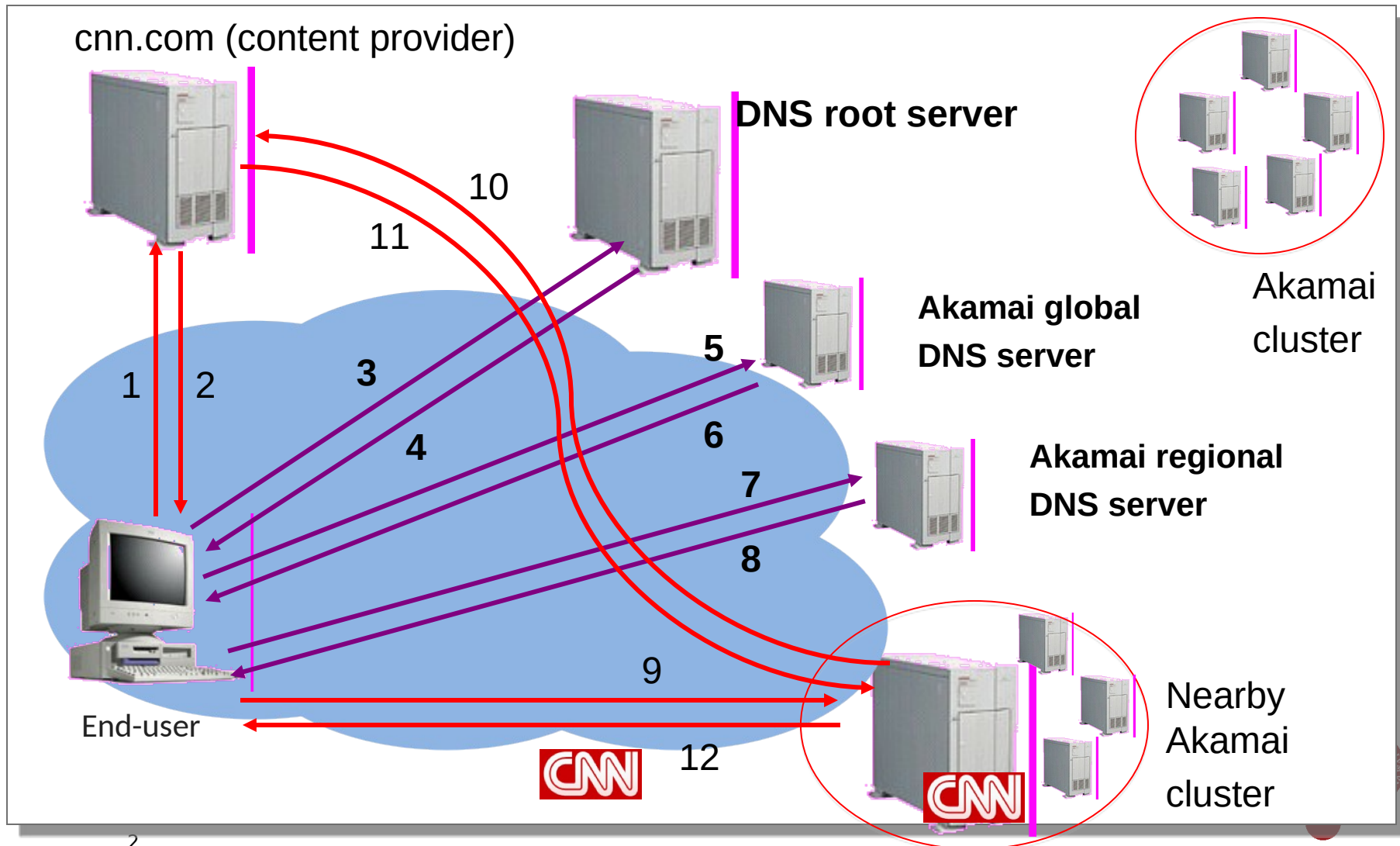
Application Layer: DNS & Peer to Peer File Sharing Transport Layer: UDP

David Marchant

Based on slides compiled by Marcos Vaz Salles, adaptations by
Vivek Shah and Michael Kirkedal Thomsen

Recap: How Akamai Works

Source: Freedman (partial)



Hierarchical Names

- **Host name:** `www.cs.princeton.edu`
 - **Domain:** registrar for each top-level domain (e.g., .edu)
 - **Host name:** local administrator assigns to each host
- **IP addresses:** `128.112.7.156`
 - **Prefixes:** ICANN, regional Internet registries, and ISPs
 - **Hosts:** static configuration, or dynamic using DHCP (more on DHCP later in the course 😊)



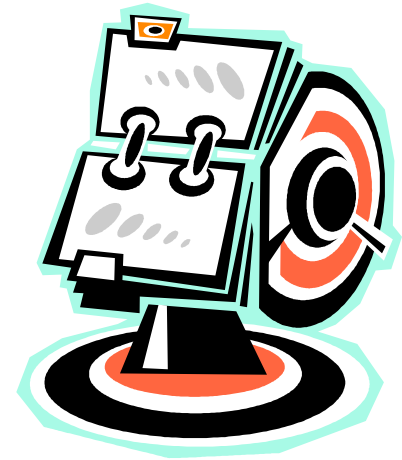
Separating Names and IP Addresses

- Names are easier (for us!) to remember
 - `www.cnn.com` vs. `64.236.16.20`
- IP addresses can change underneath
 - Move `www.cnn.com` to `173.15.201.39`
 - E.g., renumbering when changing providers
- Name could map to multiple IP addresses
 - `www.cnn.com` to multiple replicas of the Web site
- Map to different addresses in different places
 - Address of a nearby copy of the Web site
 - E.g., to reduce latency, or return different content
- Multiple names for the same address
 - E.g., aliases like `ee.mit.edu` and `cs.mit.edu`



Outline: Domain Name System

- Computer science concepts underlying DNS
 - **Indirection:** names in place of addresses
 - **Hierarchy:** in names, addresses, and servers
 - **Caching:** of mappings from names to/from addresses
- DNS software components
 - DNS resolvers
 - DNS servers
- DNS queries
 - Iterative queries
 - Recursive queries
- DNS caching based on time-to-live (TTL)



Strawman Solution: Central Server

- All you need is to map names!
- Central server
 - One place where all mappings are stored
 - All queries go to the central server

- Is this a good solution?
- What would be the potential drawbacks?



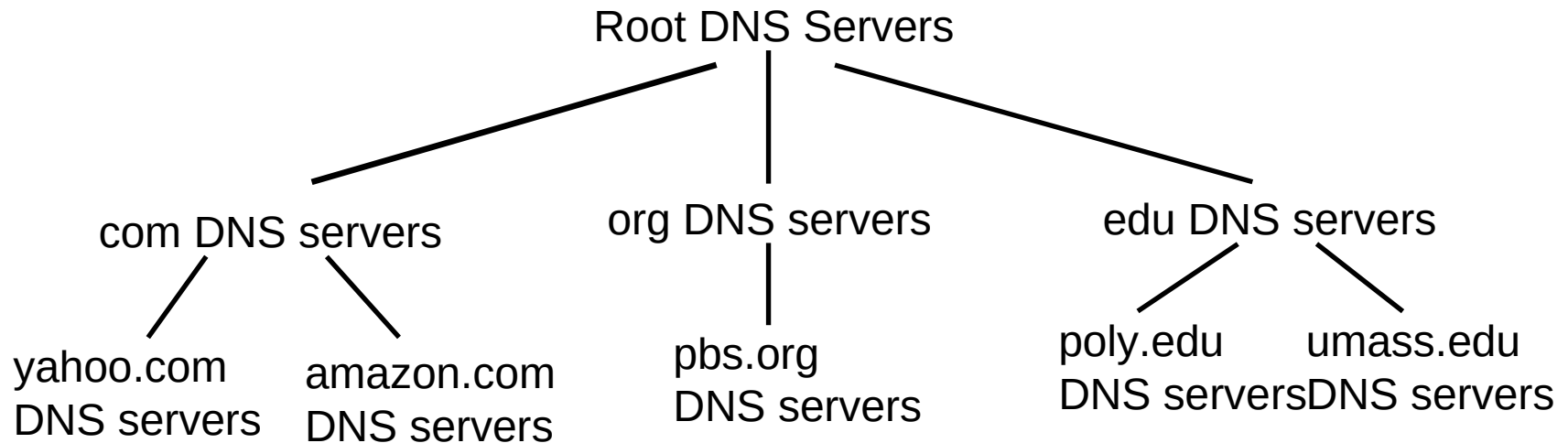
Source: Freedman (partial)

Domain Name System (DNS)

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



Distributed, Hierarchical Database



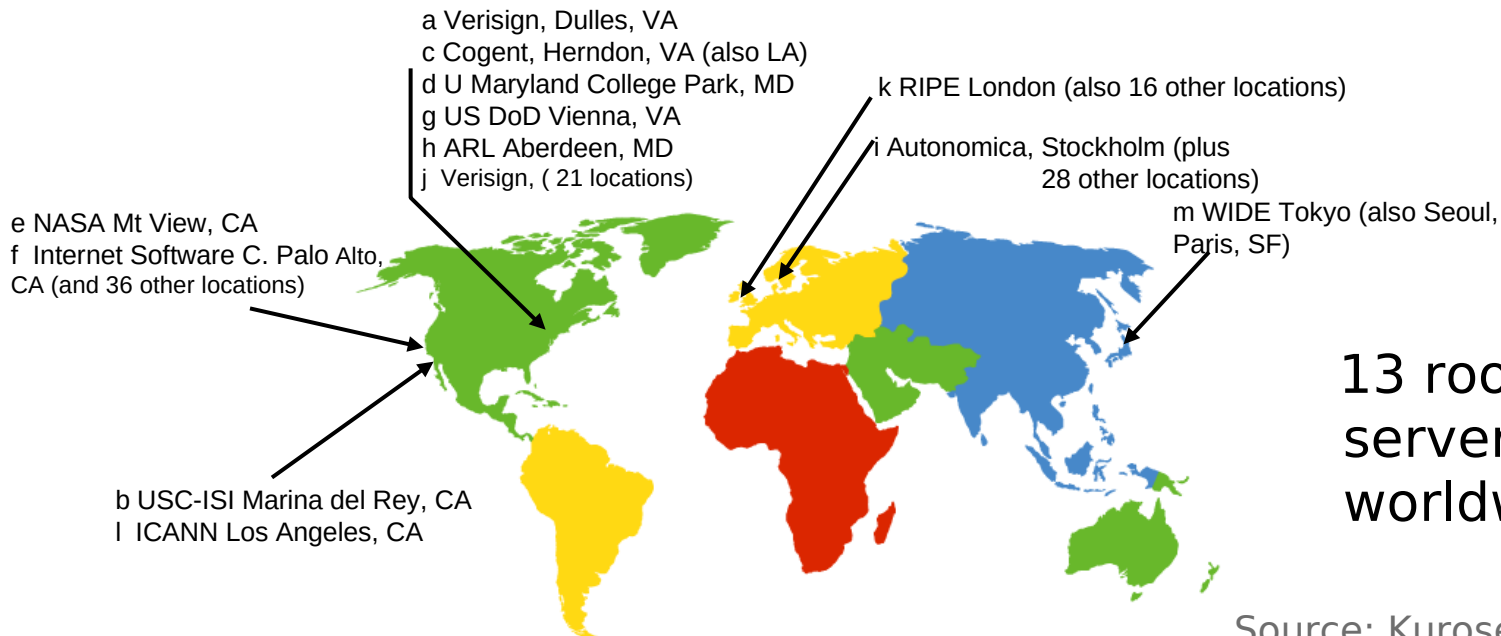
client wants IP for www.amazon.com; 1st approx:

- 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 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



13 root name
server organizers
worldwide

Source: Kurose & Ross



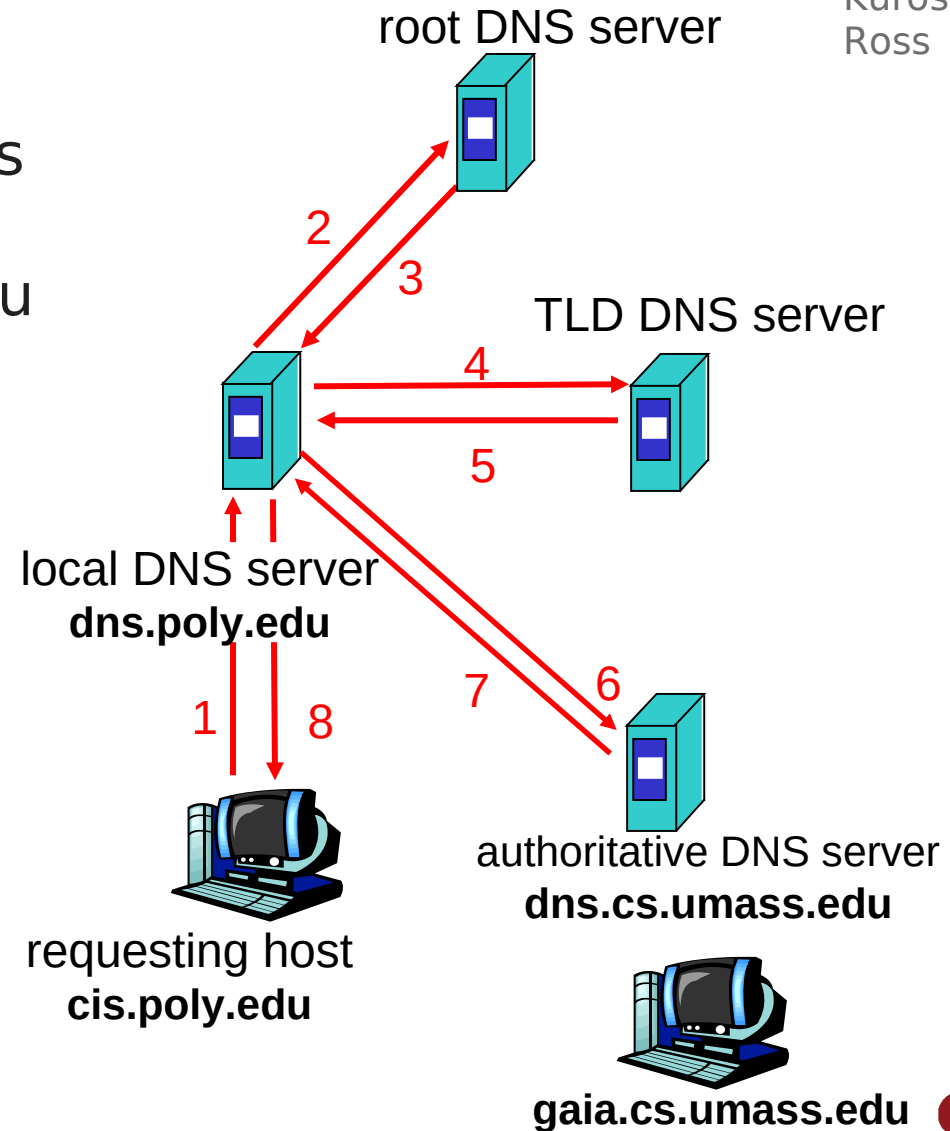
DNS Name Resolution Example

Source:
Kurose &
Ross

- 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”

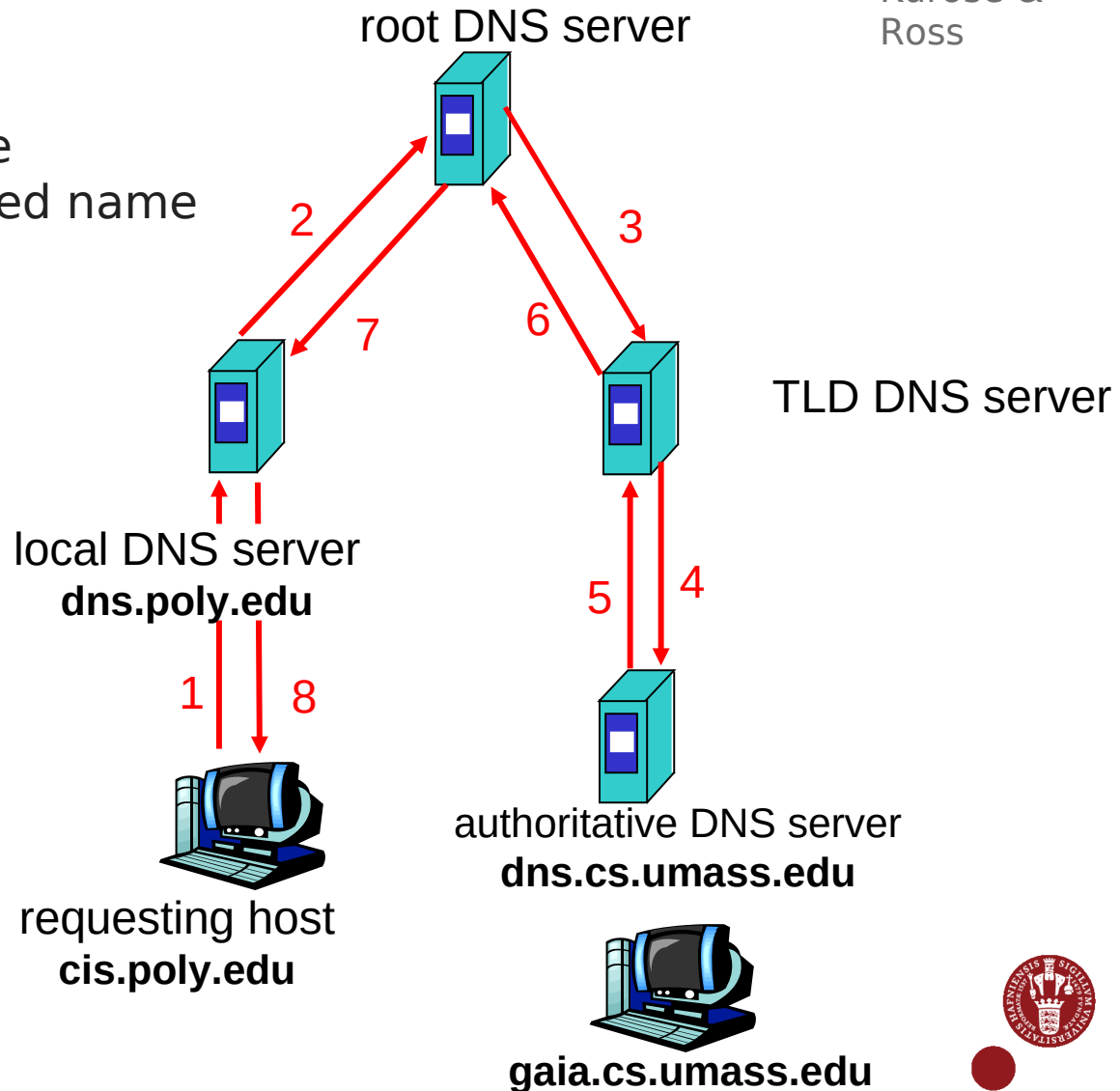


DNS Name Resolution Example

Source:
Kurose &
Ross

Recursive query

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



DNS Caching

- Performing all these queries take time
 - And all this before the actual communication takes place
 - E.g., 1-second latency before starting Web download
- Caching can substantially 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



Time to Live & Negative Caching

- How DNS caching works
 - DNS servers cache responses to queries
 - Responses include a “ **time to live**” (TTL) field
 - Server deletes the cached entry after TTL expires
- Negative Caching: Remember things that do not 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



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
- www.ibm.com is really servereast.backup2.ibm.com
- value is canonical name

Type=MX

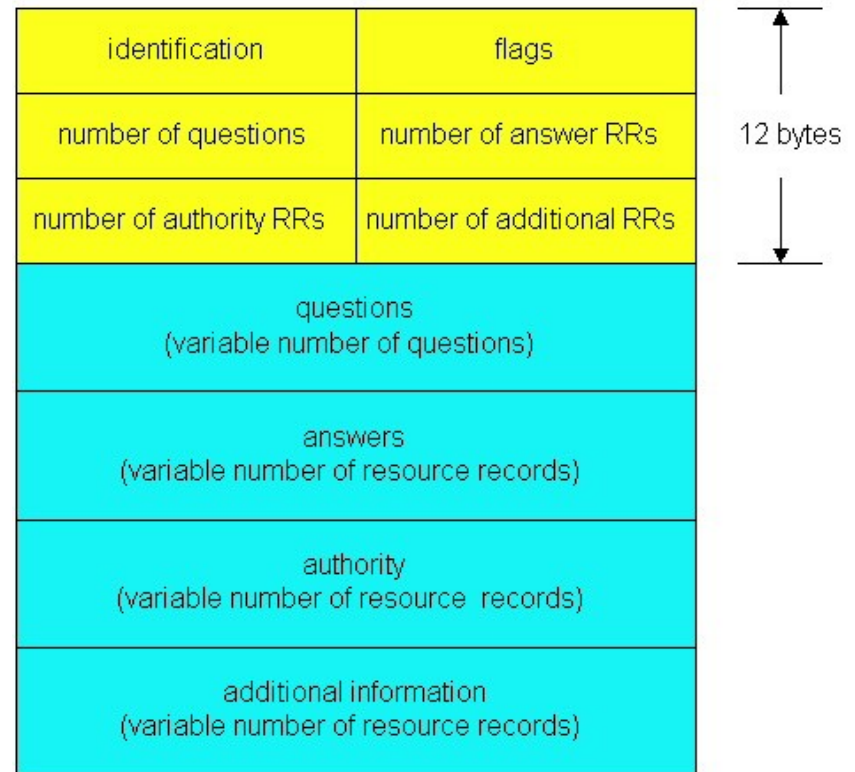
- value** is name of mailserver associated with **name**



DNS protocol, messages

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

- msg header
 - **identification**: 16 bit # for query, reply to query uses same #
 - **Flags**:
 - query or reply
 - recursion desired
 - recursion available
 - reply is authoritative



Inserting records into DNS

- example: new startup “ Network Utopia”
- register name networkutopia.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.networkutopia.com; Type MX record for networkutopia.com
- How do people get IP address of your Web site?



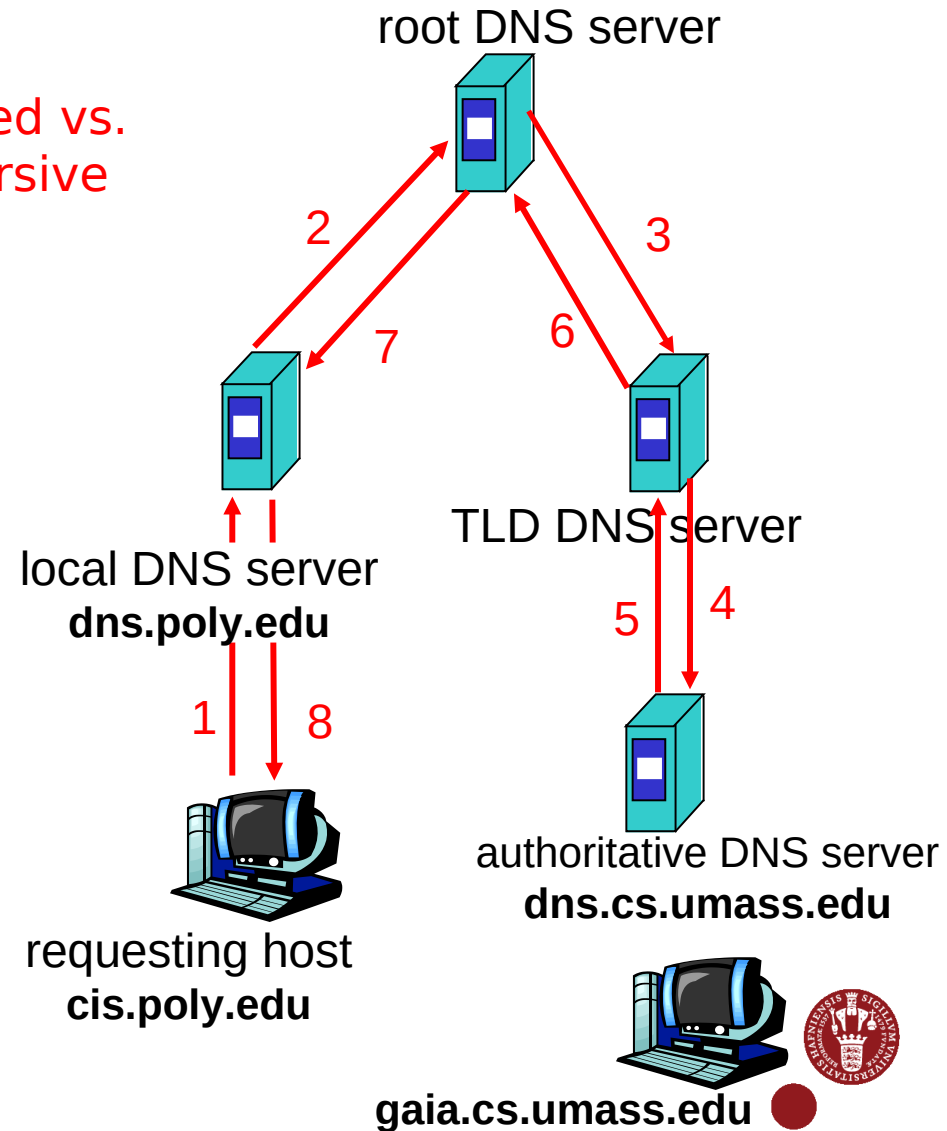
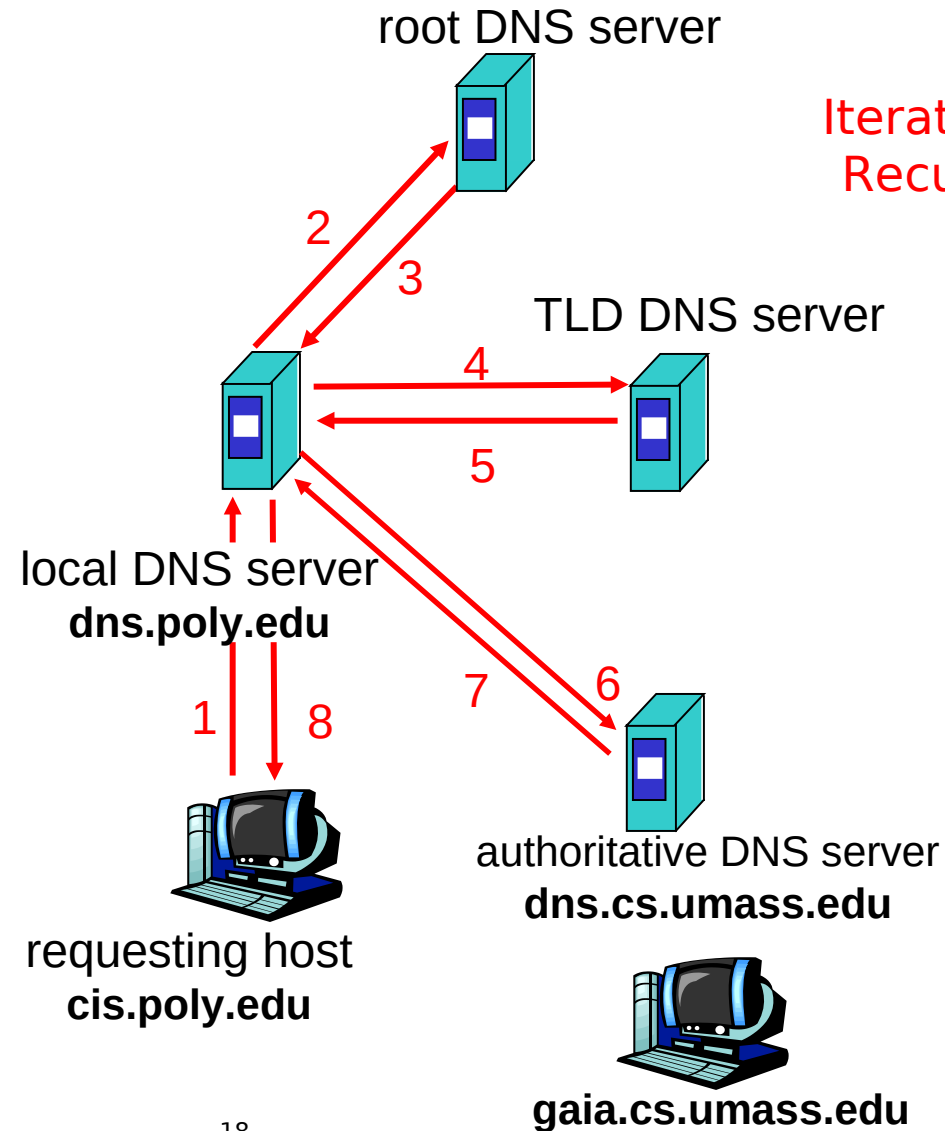
DNS security

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



How does caching affect each variant? In which one do you expect caching to work best?

Iterated vs.
Recursive



Use dig and nslookup with what you learned today

bonii@Bigbang@14:21:50 ~ \$nslookup www.diku.dk

Server: 192.38.118.220

Address: 192.38.118.220#53

Non-authoritative answer:

www.diku.dk canonical name = web-aggregator.diku.dk.

Name: web-aggregator.diku.dk

Address: 130.226.14.83

bonii@Bigbang@14:25:04 ~ \$dig +trace www.diku.dk

; <<>> DiG 9.10.4-P3 <<>> +trace www.diku.dk

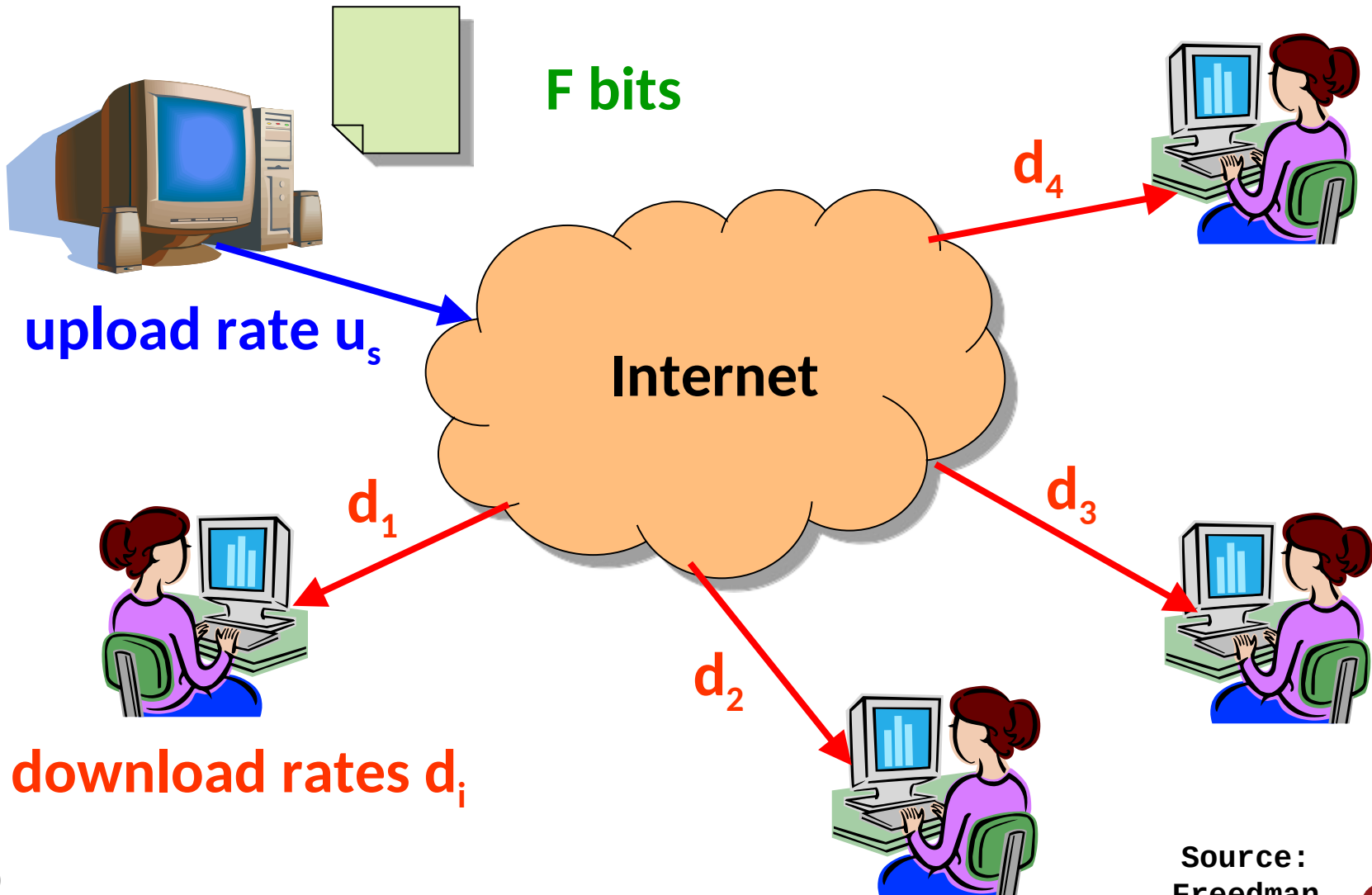
:: global options: +cmd

.	185772	IN	NS	g.root-servers.net.
.	185772	IN	NS	e.root-servers.net.
.	185772	IN	NS	c.root-servers.net.
.	185772	IN	NS	k.root-servers.net.
.	185772	IN	NS	d.root-servers.net.
.	185772	IN	NS	l.root-servers.net.
.	185772	IN	NS	m.root-servers.net.
.	185772	IN	NS	a.root-servers.net.
.	185772	IN	NS	i.root-servers.net.
.	185772	IN	NS	h.root-servers.net.
.	185772	IN	NS	f.root-servers.net.
.	185772	IN	NS	j.root-servers.net.
.	185772	IN	NS	b.root-servers.net.

:: Received 824 bytes from 192.38.118.220#53(192.38.118.220) in 0 ms



Server Distributing a Large File



Server Distributing a Large File

- Sending an F -bit file to N receivers
 - Transmitting NF bits at rate u_s
 - ... takes at least NF/u_s time
- Receiving the data at the slowest receiver
 - Slowest receiver has download rate $d_{min} = \min_i \{d_i\}$
 - ... takes at least F/d_{min} time
- Download time: $\max\{NF/u_s, F/d_{min}\}$

Source:
Freedman



Speeding Up the File Distribution

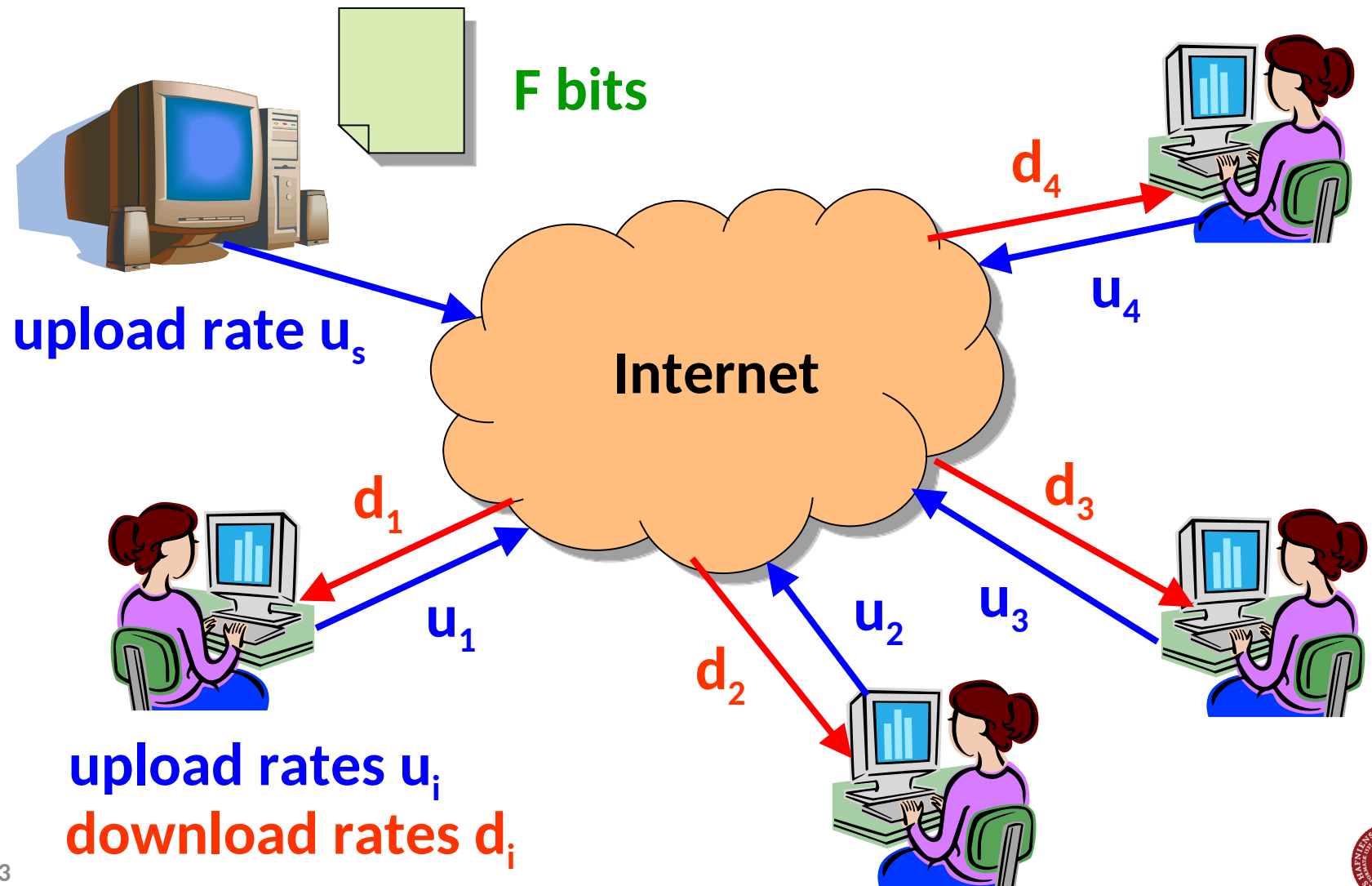
- Increase the server upload rate
 - Higher link bandwidth at the server
 - Multiple servers, each with their own link
- Alternative: have the receivers help
 - Receivers get a copy of the data
 - ... and redistribute to other receivers
 - To reduce the burden on the server



Source:
Freedman



Peers Help Distributing a Large File



Peers Help Distributing a Large File

- Components of distribution latency
 - Server must send each bit: min time F/u_s
 - Slowest peer must receive each bit: min time F/d_{min}
- Upload time using all upload resources
 - Total number of bits: NF
 - Total upload bandwidth $u_s + \sum_i(u_i)$
- Total: $\max\{F/u_s, F/d_{min}, NF/(u_s + \sum_i(u_i))\}$
- *Peer to peer is self-scaling*
 - *Download time grows slowly with N*
 - Client-server: $\max\{NF/u_s, F/d_{min}\}$
 - Peer-to-peer: $\max\{F/u_s, F/d_{min}, NF/(u_s + \sum_i(u_i))\}$



Peer-to-Peer Networks: BitTorrent

- BitTorrent history
 - 2002: B. Cohen debuted BitTorrent
- Emphasis on efficient fetching, not searching
 - Distribute same file to many peers
 - Single publisher, many downloaders
- Preventing free-loading
 - Incentives for peers to contribute

Source:
Freedman



BitTorrent: Tracker

- Infrastructure node
 - Keeps track of peers participating in the torrent
 - Peers register with the tracker when it arrives
- Tracker selects peers for downloading
 - Returns a random set of peer IP addresses
 - So the new peer knows who to contact for data
- Can have “trackerless” system
 - Using distributed hash tables (DHTs)

Source:
Freedman



BitTorrent: Chunk Request Order

- Which chunks to request?
 - Could download in order
 - Like an HTTP client does
- Problem: many peers have the early chunks
 - Peers have little to share with each other
 - Limiting the scalability of the system
- Problem: eventually nobody has rare chunks
 - E.g., the chunks need the end of the file
 - Limiting the ability to complete a download
- Solutions: random selection and rarest first

Source: Freedman



BitTorrent: Rarest Chunk First

- Which chunks to request first?
 - Chunk with fewest available copies (i.e., rarest chunk)
- Benefits to the peer
 - Avoid starvation when some peers depart
- Benefits to the system
 - Avoid starvation across all peers wanting a file
 - Balance load by equalizing # of copies of chunks

Source: Freedman



Free-Riding in P2P Networks

- Vast majority of users are free-riders
 - Most share no files and answer no queries
 - Others limit # of connections or upload speed
- A few “peers” essentially act as servers
 - A few individuals contributing to the public good
 - Making them hubs that basically act as a server
- BitTorrent prevent free riding
 - Allow the fastest peers to download from you
 - Occasionally let some free loaders download

Source: Freedman



Bit-Torrent: Preventing Free-Riding

- Peer has limited upload bandwidth
 - And must share it among multiple peers
 - Tit-for-tat: favor neighbors uploading at highest rate
- Rewarding the top four neighbors
 - Measure download bit rates from each neighbor
 - Reciprocate by sending to the top four peers
- Optimistic unchoking
 - Randomly try a new neighbor every 30 seconds
 - So new neighbor has a chance to be a better partner
 - Compatible peers find each other

Source: Freedman



Peer-to-Peer Naming

- But...
 - Peers may come and go
 - Peers need to find each other
 - Peers need to be willing to help each other

Source: Freedman



Locating the Relevant Peers

- Three main approaches
 - Central directory (Napster)
 - Query flooding (Gnutella)
 - Hierarchical overlay (Kazaa, modern Gnutella)
- Design goals
 - Scalability
 - Simplicity
 - Robustness
 - Plausible deniability

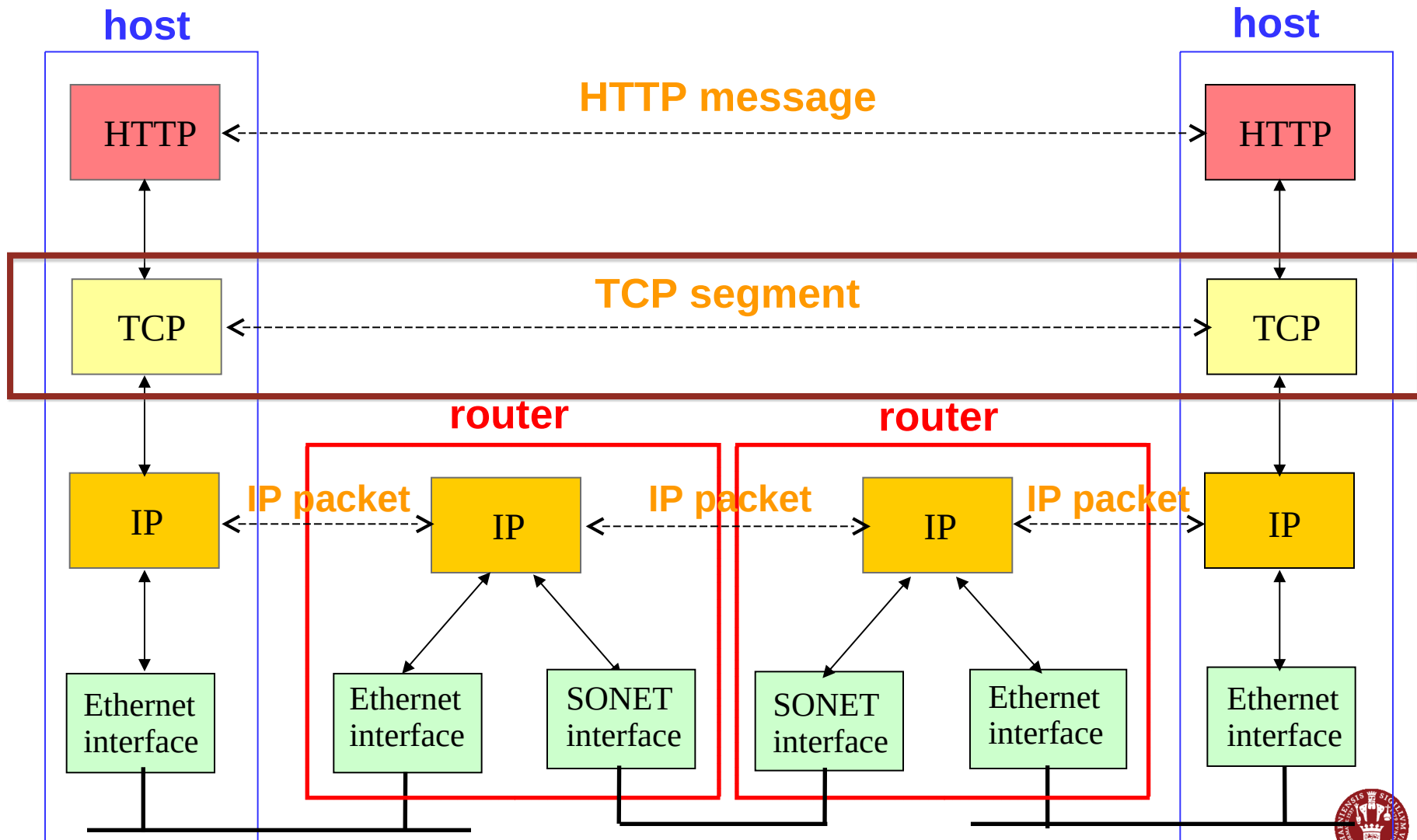
Source: Freedman



Summary

- DNS
 - Hierarchical names
 - Recursive vs. iterative name resolution, caching
 - Addresses, aliases, resource records
- P2P applications
 - Self scalability
 - BitTorrent – Popular P2P file sharing protocol
 - Rarest chunk first, fair trading + optimistic unchoking

Recap: Internet Layering Model

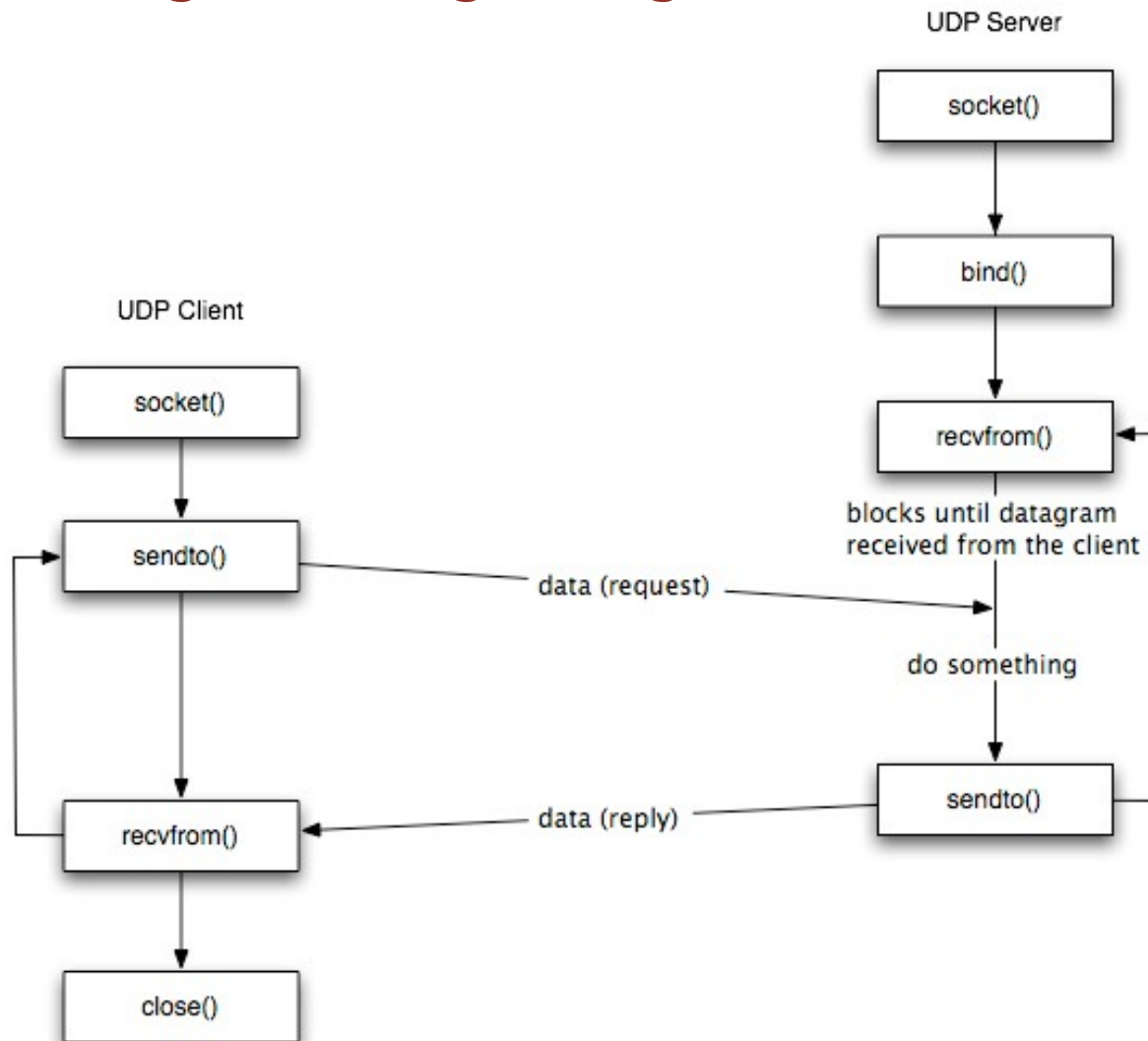


Transport Layer

- Logical Communication between processes
 - Sender divides messages into segments.
 - Receiver re-assembles messages into segments.
- Principles underlying transport-layer services
 - (De)multiplexing
 - Detecting corruption
 - Optional: Reliable delivery, Flow control, Congestion control
- Transport-layer protocols in the Internet
 - User Datagram Protocol (UDP)
 - Simple (unreliable) message delivery
 - Transmission Control Protocol (TCP)
 - Reliable bidirectional stream of bytes



Socket Programming Using UDP



Source: Campbell



Socket Programming Using UDP

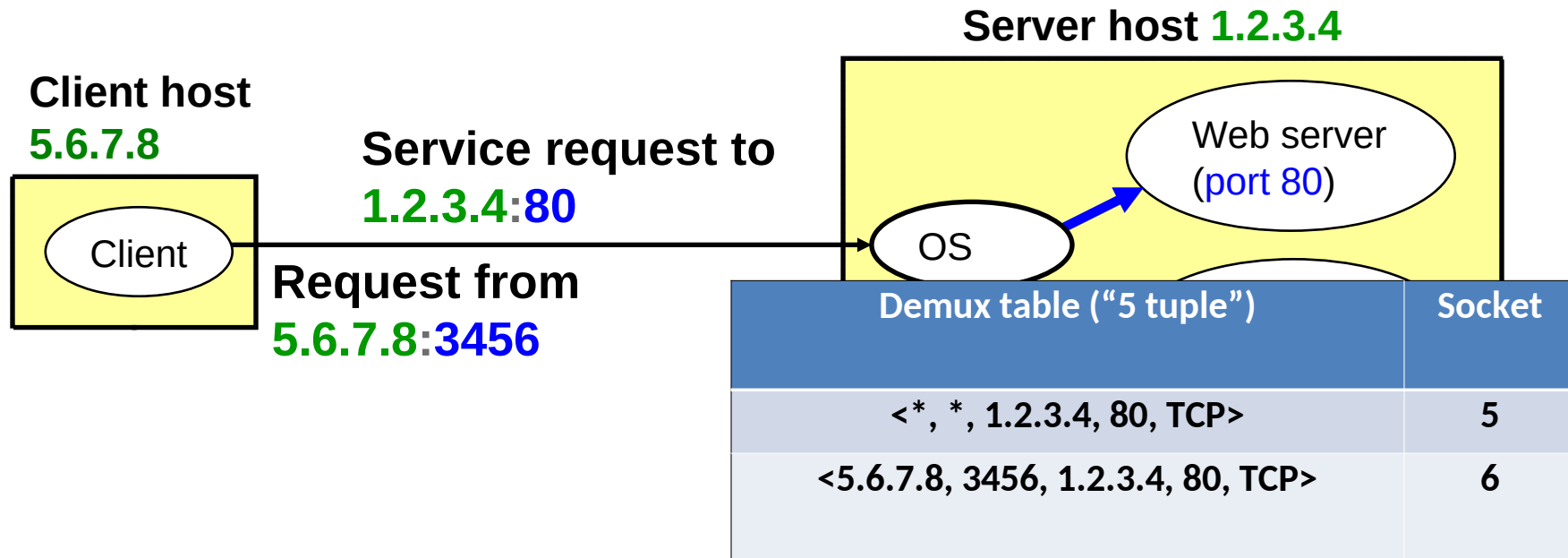
```
ssize_t recvfrom(int sockfd, void* buff,  
    size_t nbytes, int flags, struct sockaddr* from,  
    socklen_t *addrlen);
```

```
ssize_t sendto(int sockfd, const void *buff,  
    size_t nbytes, int flags,  
    const struct sockaddr *to, socklen_t addrlen);
```

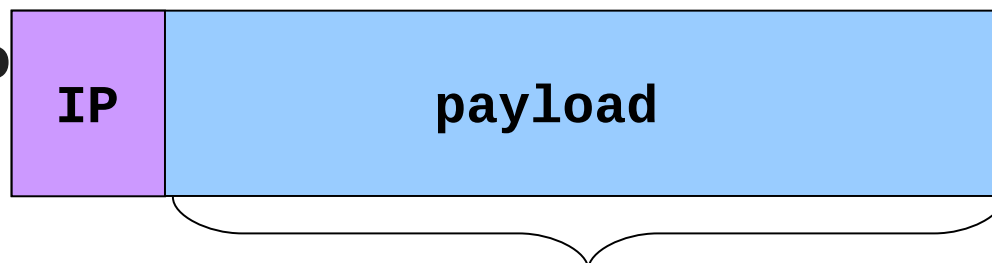


Two Basic Transport Features

- **Demultiplexing:** port numbers



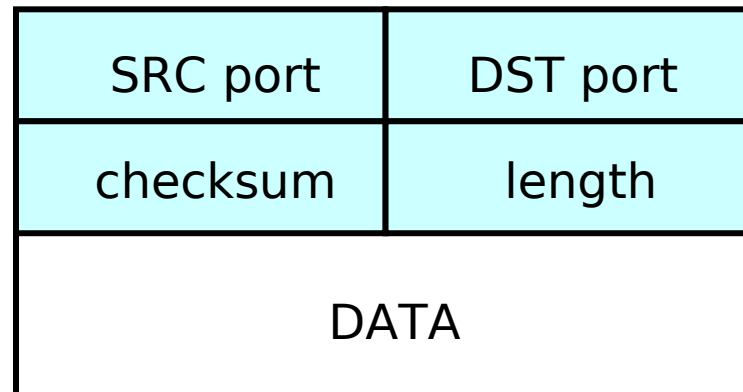
- **Error**



Source: Freedman

User Datagram Protocol (UDP)

- Datagram messaging service
 - Demultiplexing of messages: port numbers
 - Detecting corrupted messages: checksum
- Lightweight communication between processes
 - Send messages to and receive them from a socket
 - Avoid overhead and delays of ordered, reliable delivery



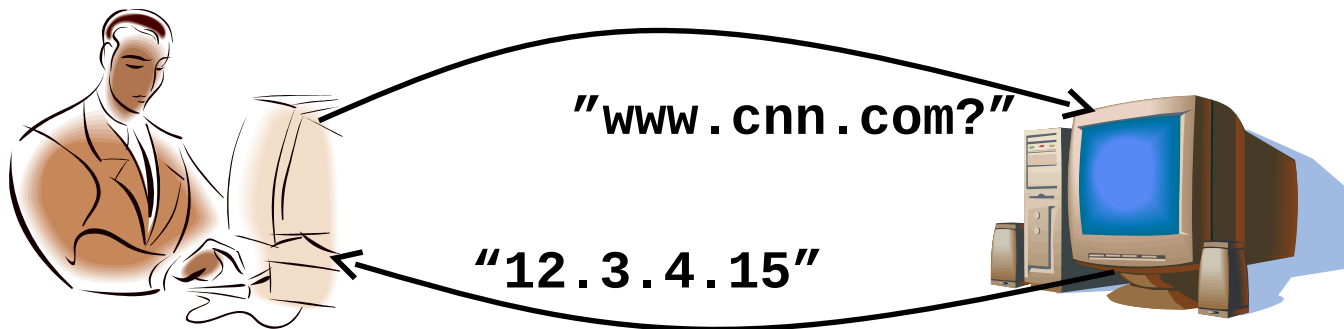
Why Would Anyone Use UDP?

- Fine control over what data is sent and when
 - As soon as app process writes into socket
 - ... UDP will package data and send packet
- No delay for connection establishment
 - UDP blasts away without any formal preliminaries
 - ... avoids introducing unnecessary delays
- No connection state (no buffers, sequence # ' s, etc.)
 - Can scale to more active clients at once
- Small packet header overhead (header only 8B long)



Popular Applications That Use UDP

- Simple query protocols like DNS
 - Overhead of connection establishment is overkill
 - Easier to have the application retransmit if needed



- Multimedia streaming (VoIP, video conferencing, ...)
 - Retransmitting lost/corrupted packets is not worthwhile
 - By time packet is retransmitted, it's too late

Summary

- UDP
 - basic multiplexing, checksums



What's next ? Reliable Data Transfer & TCP

