Chapter 9
Applications

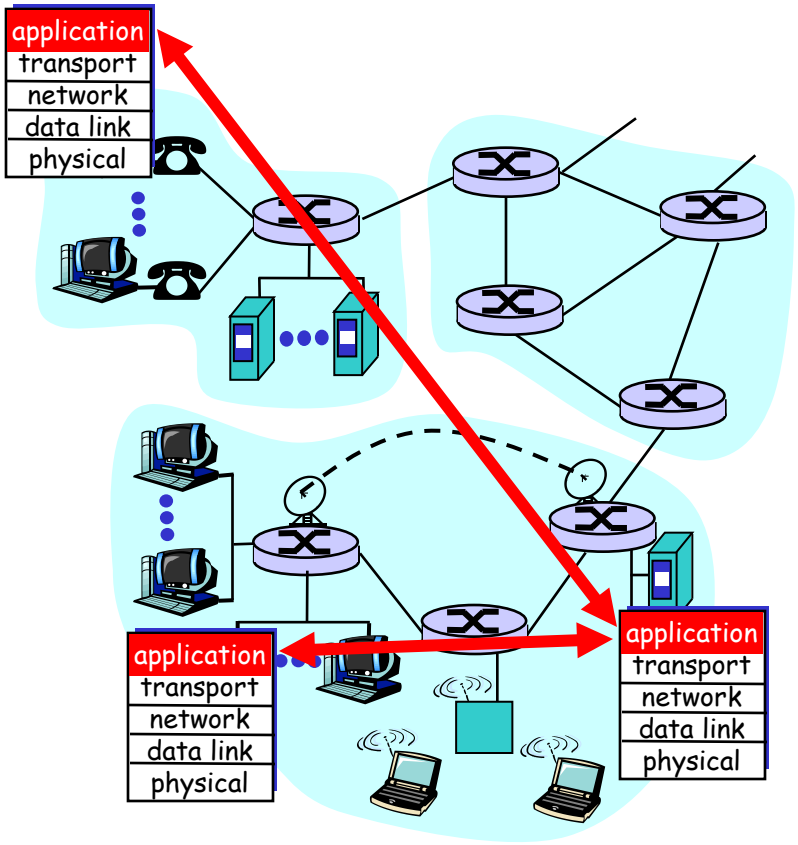
Applications and application-layer protocols

Application: communicating, distributed processes

- running in network hosts in "user space"
- exchange messages to implement application
- e.g., email, ftp, Web

Application-layer protocols

- one "piece" of an app
- define messages exchanged by apps and actions taken
- use communication services provided by lower layer protocols (TCP, UDP)



Network applications: some jargon

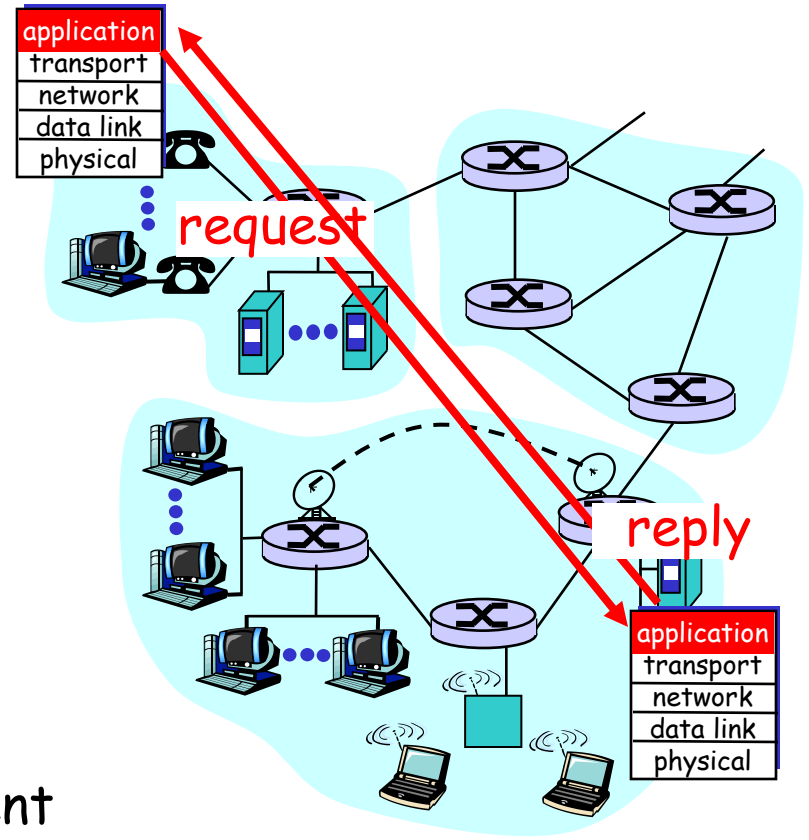
Process: program running within a host.

- within same host, two processes communicate using interprocess communication (defined by OS).
- processes running in different hosts communicate with an application-layer protocol

- user agent: software process, interfacing with user "above" and network "below".
  - implements application-level protocol
  - Web: browser
  - E-mail: mail reader
  - streaming audio/video: media player

Client-server paradigm

Typical network app has two pieces: *client* and *server*



Client:

- initiates contact with server ("speaks first")
- typically requests service from server,
- Web: client implemented in browser; e-mail: in mail reader

Server:

- provides requested service to client
- e.g., Web server sends requested Web page, mail server delivers e-mail

Internet apps: application, transport protocols

Application	Application layer protocol	Underlying transport protocol
e-mail	smtp [RFC 821]	TCP
remote terminal access	telnet [RFC 854]	TCP
Web	http [RFC 2068]	TCP
file transfer	ftp [RFC 959]	TCP
streaming multimedia	proprietary (e.g. RealNetworks)	TCP or UDP
remote file server	NSF	TCP or UDP
Internet telephony	proprietary (e.g., Vocaltec)	typically UDP

Traditional Applications

Traditional Applications

- The traditional applications are directly invoked by users:
  - **World Wide Web**, **E-mail** and **Network Management**
- Focus on four application protocols:
  - **SMTP: Simple Mail Transfer Protocol**
    - Is used to exchange electronic mail
  - **HTTP: HyperText Transport Protocol**
    - Is used to communicate between Web browsers and Web servers
  - **DNS: Domain Name System Protocol**
    - Is used to query name servers and send the responses
  - **SNMP: Simple Network Management Protocol**
    - Is used to query or modify the state of remote network nodes

Traditional Applications

- All of these application protocols follow the same **request/reply** communication mechanism
  - All are implemented on top of either **TCP or UDP**
- Each protocol (except DNS) has a companion protocol that specifies **the format of the data** that can be exchanged
  - **SMTP: RFC 822 and MIME (Multipurpose Internet Mail Extensions)** define the format of email messages
  - **HTTP: HTML (HyperText Markup Language)** is a specification that defines the form of those pages
  - **SNMP: MIB (Management Information Base)** defines the variables that can be queried

Electronic Mail
(SMTP, MIME, IMAP)

Message Format (RFC 822)

- **RFC 822** defines messages to have two parts:
  - A **header** and a **body**
  - Both parts are represented in **ASCII text**
- The message header is a series of **<CRLF>**-terminated lines
  - <CRLF> stands for carriage-return + line-feed
    - Are a pair of ASCII control characters often used to indicate **the end of a line of text**
  - The header is separated from the message body by a **blank line**
  - Each header line contains a **type** and **value** separated by a colon (:)

American Standard Code for Information Interchange (ASCII)

- 7 bit code
- Reserves the first 32 codes (numbers 0–31 decimal) for control characters
- Line feed 0A
- Carriage return 0D

USASCII code chart

Bits b_7 b_6 b_5 →	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b_4 b_3 b_2 b_1 ↓ Row / Column →	0	1	2	3	4	5	6	7
0 0 0 0	0	NUL	DLE	SP	0	@	P	`	p
0 0 0 1	1	SOH	DC1	!	1	A	Q	a	q
0 0 1 0	2	STX	DC2	"	2	B	R	b	r
0 0 1 1	3	ETX	DC3	#	3	C	S	c	s
0 1 0 0	4	EOT	DC4	$	4	D	T	d	t
0 1 0 1	5	ENQ	NAK	%	5	E	U	e	u
0 1 1 0	6	ACK	SYN	&	6	F	V	f	v
0 1 1 1	7	BEL	ETB	'	7	G	W	g	w
1 0 0 0	8	BS	CAN	(	8	H	X	h	x
1 0 0 1	9	HT	EM	)	9	I	Y	i	y
1 0 1 0	10	LF	SUB	*	:	J	Z	j	z
1 0 1 1	11	VT	ESC	+	;	K	[	k	{
1 1 0 0	12	FF	FS	,	<	L	\	l	|
1 1 0 1	13	CR	GS	-	=	M	]	m	}
1 1 1 0	14	SO	RS	.	>	N	^	n	~
1 1 1 1	15	SI	US	/	?	O	_	o	DEL

Message Format (RFC 822)

- The message header includes the following types:
  - **To:** identifies the message **recipient**
  - **Subject:** something about the **purpose** of the message
  - Other headers are filled in by the underlying **mail delivery system**
  - **Date:** when the message was transmitted
  - **From:** what user sent the message
  - **Received:** each mail server that handled this message
  - There are also many other header lines
- RFC 822 was extended in 1993 to allow email messages to carry many **different types of data**
  - Audio, video, images, Word documents, and so on

Message Format (MIME, First Piece)

- **MIME (Multipurpose Internet Mail Extensions)** consists of **three** basic pieces
- The **first piece** is a collection of **header lines**
  - Expand the original set defined by RFC 822
- These **header lines** describe the data being carried in the message body
  - **MIME-Version:** the version of MIME being used
  - **Content-Description:** a human-readable description of what's in the message, analogous to the **Subject:** line
  - **Content-Type:** the type of data contained in the message
  - **Content-Transfer-Encoding:** how the data in the message body is encoded

Message Format (MIME, Second Piece)

- The **second piece** is **definitions** for a set of **content types and subtypes**
  - MIME defines two different **still image types**: **image/gif** and **image/jpeg**
  - MIME also defines **text types**: **text/plain** (simple text) and **text/richtext** (a message that contains "marked up" text)
    - Marked up text: e.g., text using special fonts, italics, etc.
  - MIME defines an **application type**:
    - **Subtypes:** the output of different application programs (e.g., **application/postscript** and **application/msword**)
- MIME also defines a **multipart type** that says how a message carrying more than one data type is structured

Message Format (MIME, Third Piece)

- The **third piece** is a way to **encode the various data types** so they can be shipped in an **ASCII email message**
  - Email messages contain only ASCII
- They might pass through a number of **intermediate systems** that assume the message is in ASCII
  - The messages would **corrupt** if it contained **non-ASCII characters**
- Some data types (e.g. a JPEG image):
  - Any given **8-bit byte** in the image might contain one of **256 different values**
  - Only a **subset** of these values are **valid ASCII characters**
    - Number of ASCII characters < 256

Message Format (MIME, Third Piece)

- MIME uses a **straightforward** encoding of binary data into the ASCII character set
  - The encoding is called **base64**
    - Map every **three bytes** of the original binary data into **four ASCII characters**
  - Group the binary data into **24-bit** (3×8) units, and breaking each such unit into **four 6-bit** (4×6 = 24) **pieces**
  - Each **6-bit piece** maps onto one of **64 valid ASCII** characters (the first 64 values in the ASCII character set)
    - The **52** upper- and lower-case letters, the **10** digits 0 through 9, and the special characters **+ and /**
- "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"

Message Format (MIME)

- A message that contains some plain text, a JPEG image and a PostScript file would look something like this:

```
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="-------417CA6E2DE4ABCAFBC5"
From: Alice Smith <Alice@cisco.com>
To: Bob@cs.Princeton.edu
Subject: promised material
Date: Mon, 07 Sep 1998 19:45:19 -0400

---------417CA6E2DE4ABCAFBC5
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit

Bob,
```

Message Format (MIME)

```
Bob,

Here's the jpeg image and draft report I promised.

--Alice

---------417CA6E2DE4ABCAFBC5
Content-Type: image/jpeg
Content-Transfer-Encoding: base64
```

... unreadable encoding of a jpeg figure

```
---------417CA6E2DE4ABCAFBC5
Content-Type: application/postscript; name="draft.ps"
Content-Transfer-Encoding: 7bit
```

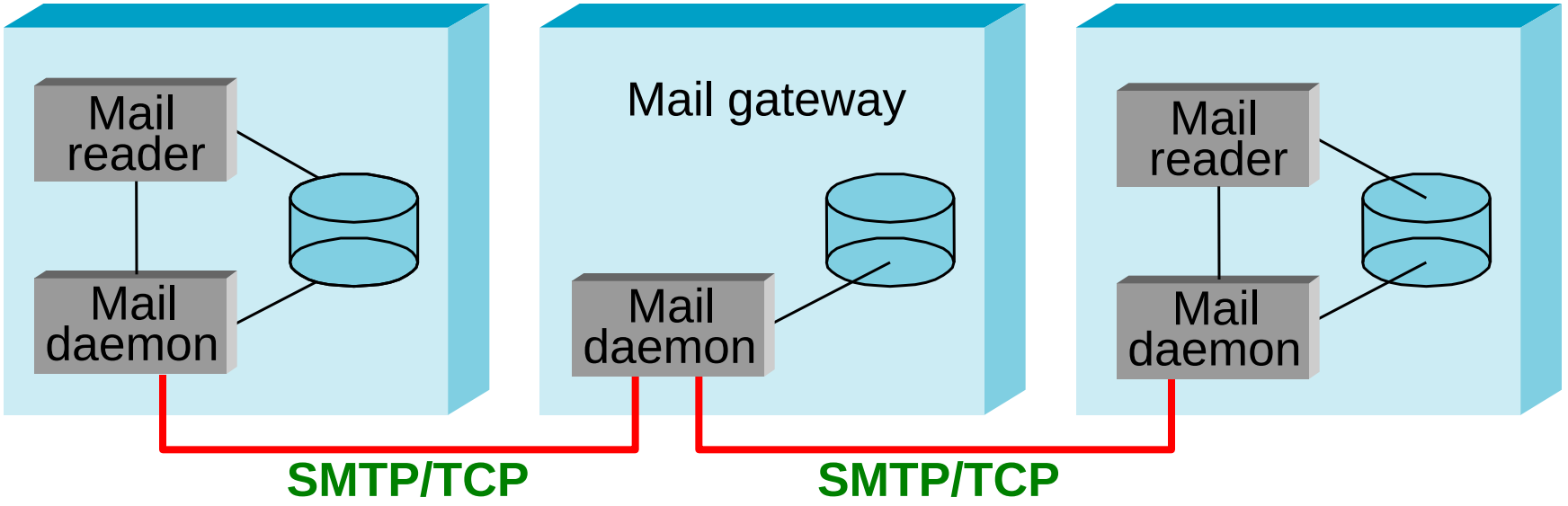
... readable encoding of a PostScript document

Message Transfer (SMTP)

- **SMTP:** the protocol used to transfer messages from one host to another
  - Users interact with a **mail reader**
    - Use to compose, file, search, and read their email
  - There is a **mail daemon** (or process) running on each host
  - The daemon uses SMTP running **over TCP** to transmit the message to a daemon running on another machine
  - The daemon puts incoming messages into the user's **mailbox**
    - User's mail reader can later find it

Message Transfer (SMTP)

- In many cases, the mail traverses one or more **mail gateways** on its route from the sender's host to the receiver's host
  - These gateways also run a **sendmail** process
- A mail gateway typically **buffers** messages on disk and is willing to **try retransmitting** them to the next machine for several days



Message Transfer (SMTP)

- For example, mail delivered to **Bob@oz.nthu.edu.tw**
  - Is first sent to a **mail gateway** OZ at NTHU
  - Then is forwarded **(a second SMTP/TCP connection)** to the specific machine on which Bob is reading his email
- The forwarding gateway maintains a database that maps **users** into the **machine** on which they currently receive their mails
  - The sender **need not** be aware of this specific name
  - The recipient's machine may not always be up
- Each SMTP session involves a dialog between the two mail daemons
  - One acting as the **client** and the other acting as the **server**
  - SMTP is also **ASCII based**

Message Transfer (SMTP)

- Bob at Princeton (cs.princeton.edu) is trying to send mail to users Alice and Tom at Cicso (cisco.com)
- HELO cs.princeton.edu ← **From cs.princeton.edu**
- 250 Hello daemon@mail.cs.princeton.edu [128.12.169.24] ← **From cisco.com**
- MAIL FROM:<Bob@cs.princeton.edu>
- 250 OK
- RCPT TO:<Alice@cisco.com>
- 250 OK
- RCPT TO:<Tom@cisco.com>
- 550 No such user here
- DATA
- 354 Start mail input; end with <CRLF>.<CRLF>
- ...etc. etc. etc.
- <CRLF>.<CRLF>
- 250 OK
- QUIT
- 221 Closing connection

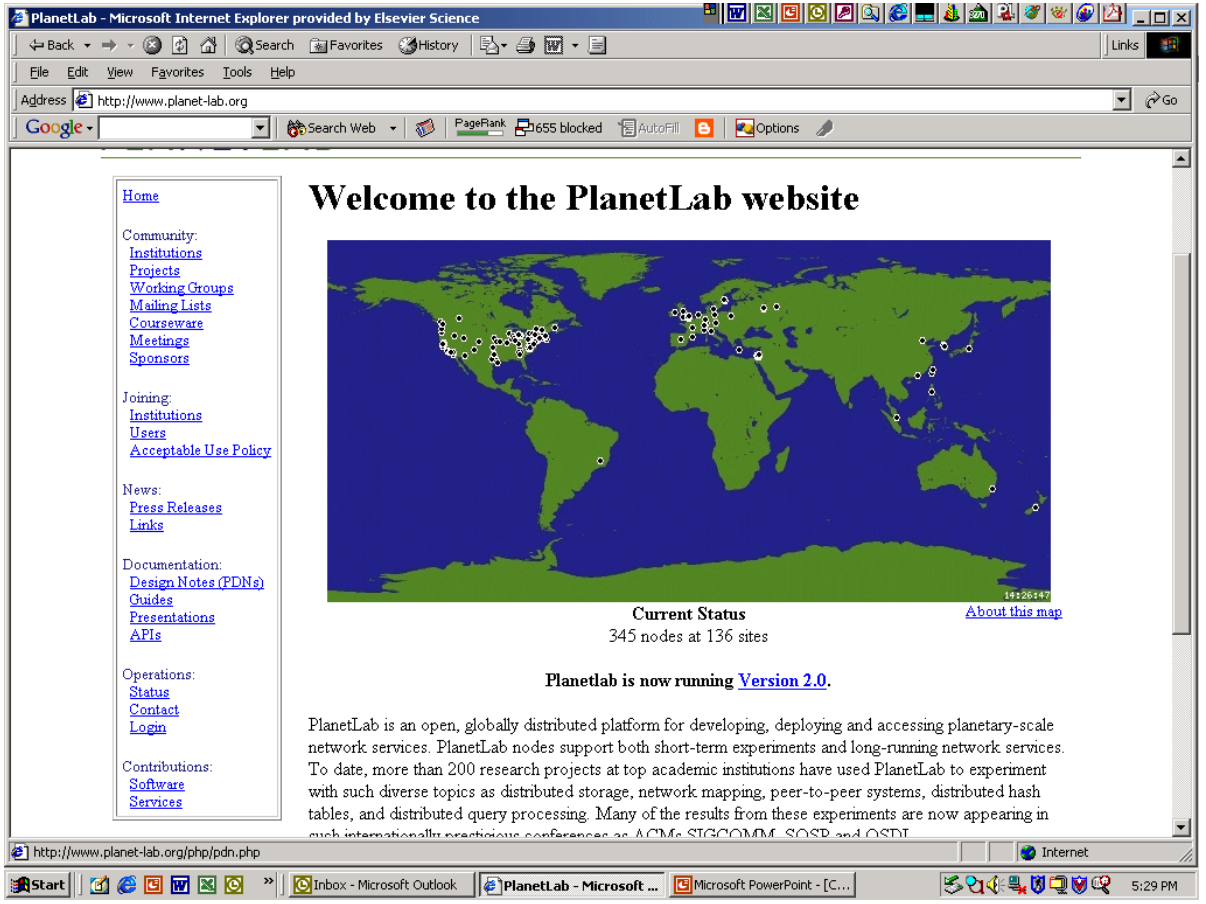
Mail Reader

- The final step is for the user to actually **retrieve** his or her messages from the mailbox
  - Read them, reply to them, and possibly save a copy for future reference
  - The user performs all these actions by interacting with a **mail reader**
- In other cases, the user accesses his or her mailbox **from a remote machine** using yet another protocol
  - **Post Office Protocol (POP)**
  - **Internet Message Access Protocol (IMAP)**

World Wide Web (HTTP)

World Wide Web (HTTP)

- Web is viewed as a set of cooperating clients and servers
  - All of whom speak the same language: **HTTP**
- Web browser:
  - Netscape
  - Explorer



World Wide Web (HTTP)

- Any Web browser has a function that allows the user to **"open a URL"**
  - **http://www.cs.princeton.edu/index.html**
  - It would open a **TCP connection** to the Web server at a machine called **www.cs.princeton.edu**
  - It would Retrieve and display the file called **index.html**
- Most files on the Web contain images and text, and some have audio and video clips
- They also include URLs that point to other files
  - These **embedded URLs** are called **hypertext links**

World Wide Web (HTTP)

- When a page is selected, the browser (the client) acquires the page from the server using HTTP running over TCP
  - HTTP is a **text-oriented** protocol
- Each HTTP message has the general form:
  - **START_LINE <CRLF>**
  - **MESSAGE_HEADER <CRLF>**
  - **<CRLF>**
  - **MESSAGE_BODY <CRLF>**
- **START_LINE:** indicates whether this is a **request** message or a **response** message
- **MESSAGE_HEADER:** defines many possible header types
  - The set is terminated by a **blank line**
- **MESSAGE_BODY:** is the contents of the requested message

Request Messages

- **START_LINE** of a **request message** specifies three things:
  - The operation to be performed
  - The Web page the operation should be performed on
  - The version of HTTP being used
- The two most common operations are
  - **GET:** used to retrieve and display a Web page
  - **HEAD:** used to **test the validity** of a hypertext link or to see if a particular page has been **modified** since the browser last fetched it
- **GET** http://www.cs.princeton.edu/index.html HTTP/1.1
  - The client wants the server on host www.cs.princeton.edu to return the page named index.html
  - Uses an **absolute** URL

Request Messages

- It is also possible to use a **relative** identifier
  - Specify the host name in one of the **MESSAGE_HEADER**
  - **GET index.html HTTP/1.1**
  - **Host: www.cs.princeton.edu**

Operation	**Description**
OPTIONS	Request information about available options
GET	Retrieve document identified in URL
HEAD	Retrieve meta-information about document identified in URL
POST	Give information (e.g., annotation) to server
PUT	Store document under specified URL
DELETE	Delete specified URL
TRACE	Loopback request message
CONNECT	For use by proxies

Response Messages

- **Response messages** begin with a single **START_LINE** with
  - The **version of HTTP** being used
  - A **three-digit code** indicating whether or not the request was successful, and
  - A text string giving the **reason**
  - **HTTP/1.1 202 Accepted HTTP/1.1 404 Not Found**
- There are five general types of response codes

Code	Type	Example Reasons
1xx	Informational	Request received, continuing process
2xx	Success	Action successfully received, understood, and accepted
3xx	Redirection	Further action must be taken to complete the request
4xx	Client Error	Request contains bad syntax or cannot be fulfilled
5xx	Server Error	Server failed to fulfill an apparently valid request

Response Messages

- Response messages can also contain one or more **MESSAGE_HEADER** lines
  - Relay additional information back to the client
- For example, the **Location header** line specifies that the requested URL is available at another location
  - http://www.cs.princeton.edu/index.html had moved to http://www.princeton.edu/cs/index.html
- The server at the original address might respond with
  - **HTTP/1.1 301 Moved Permanently**
  - **Location: http://www.princeton.edu/cs/index.html**

TCP Connections

- The original version of **HTTP (1.0)** established a **separate** TCP connection **for each data item** retrieved from the server
  - This was a very inefficient mechanism
  - Retrieving a page that included some text and a dozen icons
    - Would result in **13** separate TCP connections being established and closed
- The most important improvement in the latest version of **HTTP (1.1)** is to allow **persistent connections**
  - The client and server can exchange **multiple** request/response messages over **the same** TCP connection

TCP Connections

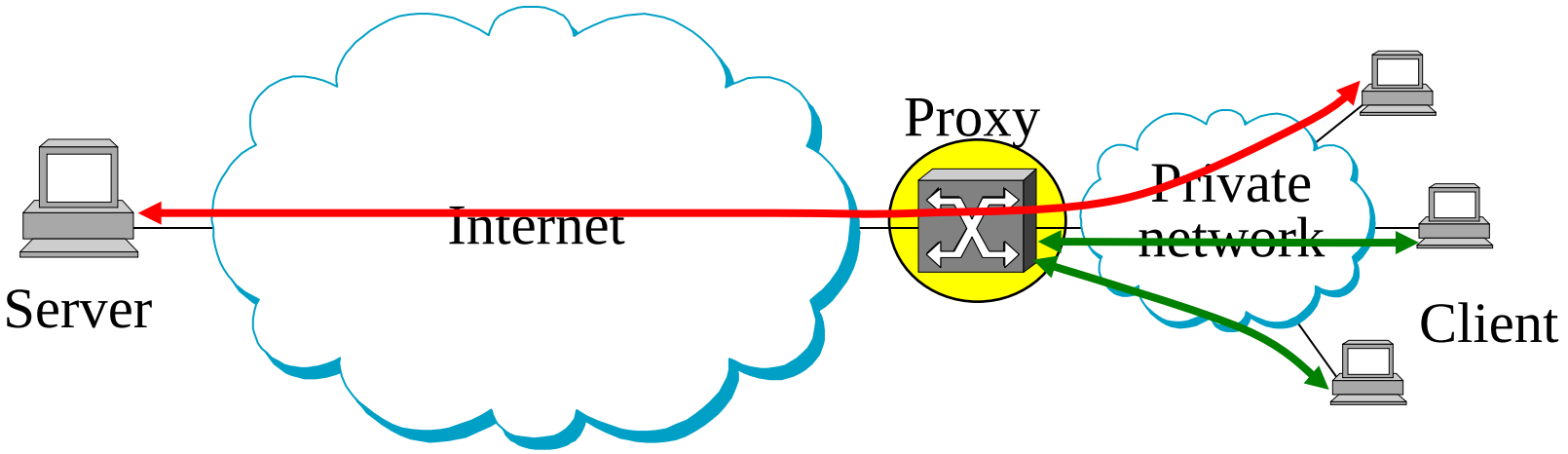
- Persistent connections have two advantages:
  - Eliminate the connection setup overhead
    - **Reduces** the **delay** and the **load** on the server, as well as on the network, caused by the additional TCP packets
  - TCP's congestion window mechanism is able to operate **more efficiently**
    - Does not go through the **slow start** phase for each page
- Neither the client nor server necessarily knows **how long** to keep a particular TCP connection open
  - The server must **time out** and **close** a connection if it has received no requests on the connection for a period of time

Caching

- From the **client's** perspective, a page that can be retrieved from a nearby cache can be **displayed much more quickly**
- From the **server's** perspective, having a cache intercept and satisfy a request **reduces the load on the server**
- Caching can be implemented in many different places:
  - A **user's browser** can cache recently accessed pages
    - Display the cached copy if the user visits the same page again
  - A **site** can support a single sitewide cache
    - Allows users to take advantage of pages previously downloaded by **other users**
  - **ISPs** can cache pages

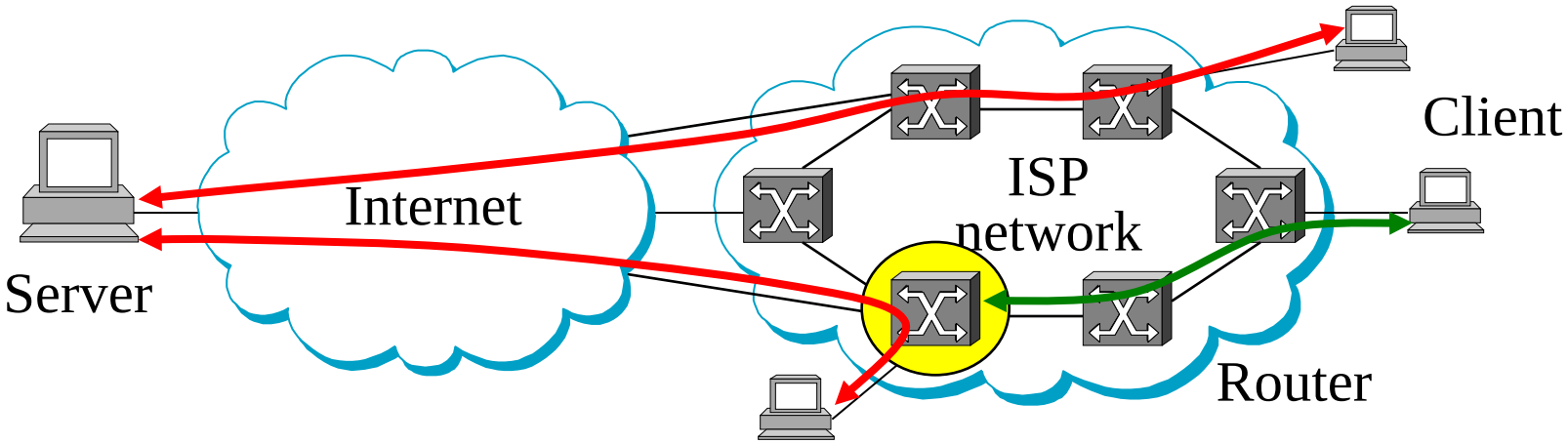
Caching

- In the second case, the users within the site most likely know **what machine** is caching pages on behalf of the site
  - They configure their browsers to connect directly to the **caching host** (called a **proxy**)



Caching

- In contrast, the sites that connect to the ISP are probably not aware that the **ISP is caching pages**
  - The HTTP requests coming out of the various sites pass through **a common ISP router**
  - This router **peeks inside the URL** for the requested page
    - If it has the page in its cache, it returns this page
    - If not, it forwards the request to the server



Caching

- No matter where pages are cached, the cache needs to make sure it is not responding with an **out-of-date version** of the page
  - The **server** assigns an **expiration date** to each page it sends back to the client
    - In the Expires header field
  - The **cache remembers** this date
    - It need **not re-verify** the page each time it is requested until after that expiration date has passed

Domain Name Service (DNS)

Domain Name Service (DNS)

- We have been using **addresses** to identify hosts (192.12.69.5)
  - Addresses are perfectly suitable for processing by **routers**
  - Addresses are **not exactly user friendly**
- A **unique name** is also assigned to each host in a network
- **Host names** differ from host addresses in two important ways:
  - They are usually of **variable length** and containing letters
    - Easier for humans to remember
  - They typically contain **no information** that helps the network to locate the host
- A **naming server** must be developed to map **user-friendly** names into **router-friendly** addresses

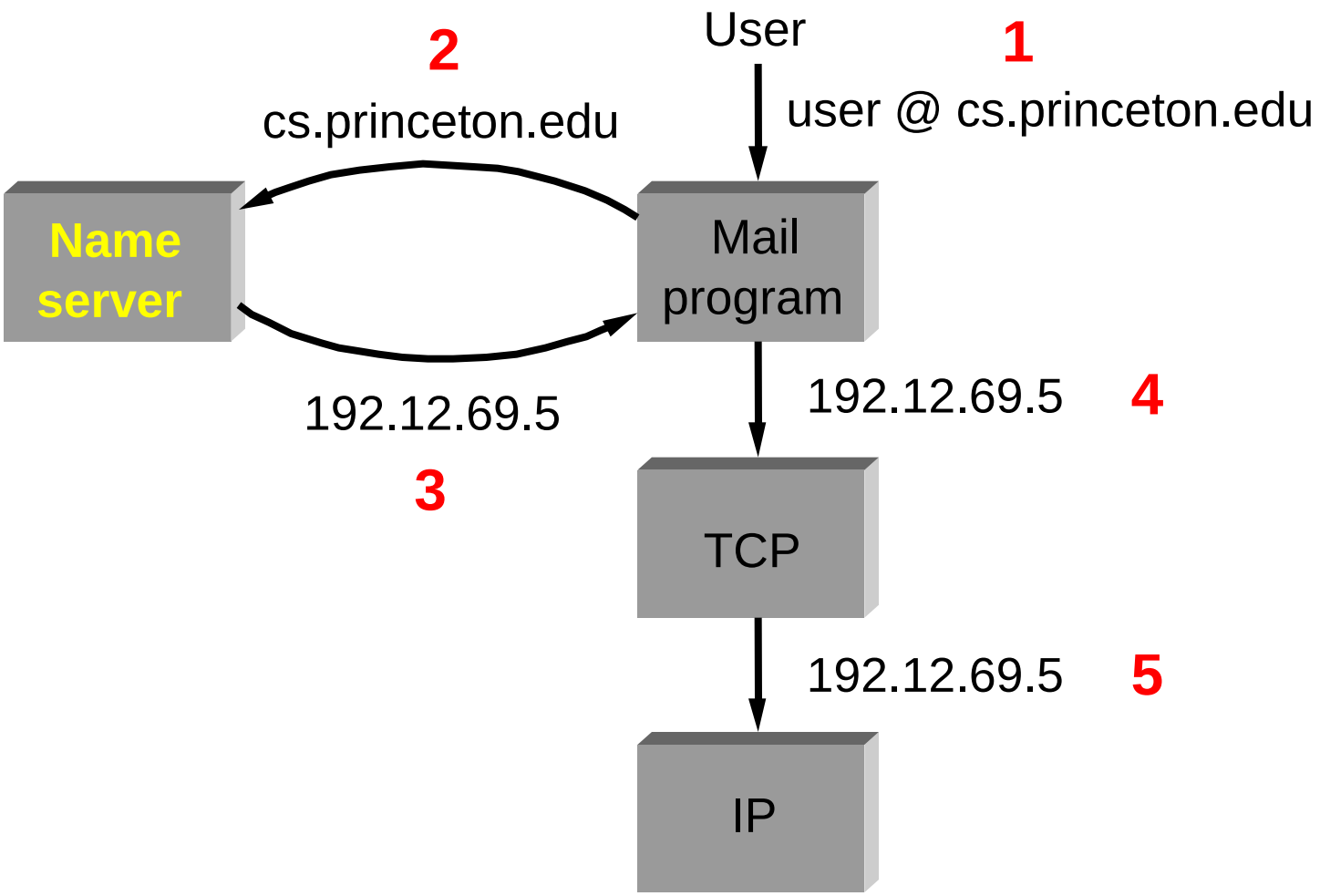
Domain Name Service (DNS)

- **Name space:** defines the set of possible names
  - **Flat:** names are not divisible into components, or
  - **Hierarchical:** XXX.com, XXX.tw, XXX.nthu.edu.tw
- The naming system maintains a collection of bindings of **names to values**
  - The value is generally an **IP address**
- **Resolution mechanism** is a procedure that returns the corresponding value
- **Name server** is a specific implementation of a resolution mechanism that is available on a network
  - It can be queried by sending it a message

Domain Name Service (DNS)

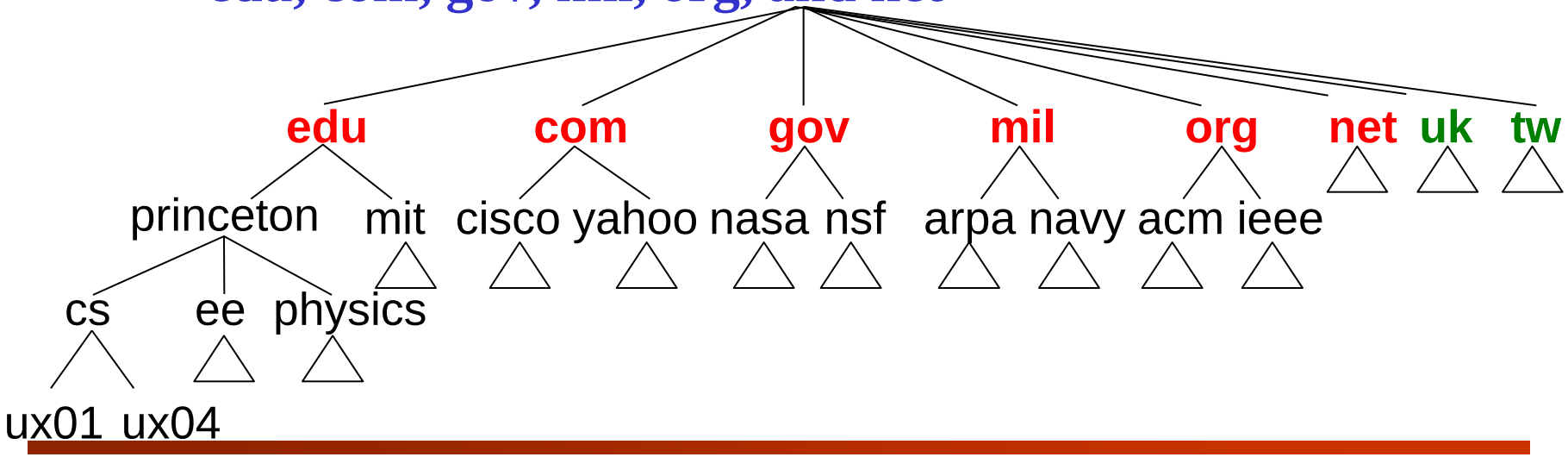
- The Internet has a particularly well-developed naming system — **Domain Name System (DNS)**
  - DNS employs a **hierarchical** name space
  - The name space is partitioned into **disjoint** pieces and distributed throughout the Internet
- A user presents a **host name** to an application program (such as the Web Browser)
  - This program engages the naming system to **translate** this name into a **host address**
  - According to the returned **host's IP address**, the application then **opens a connection** to this host

Domain Name Service (DNS)



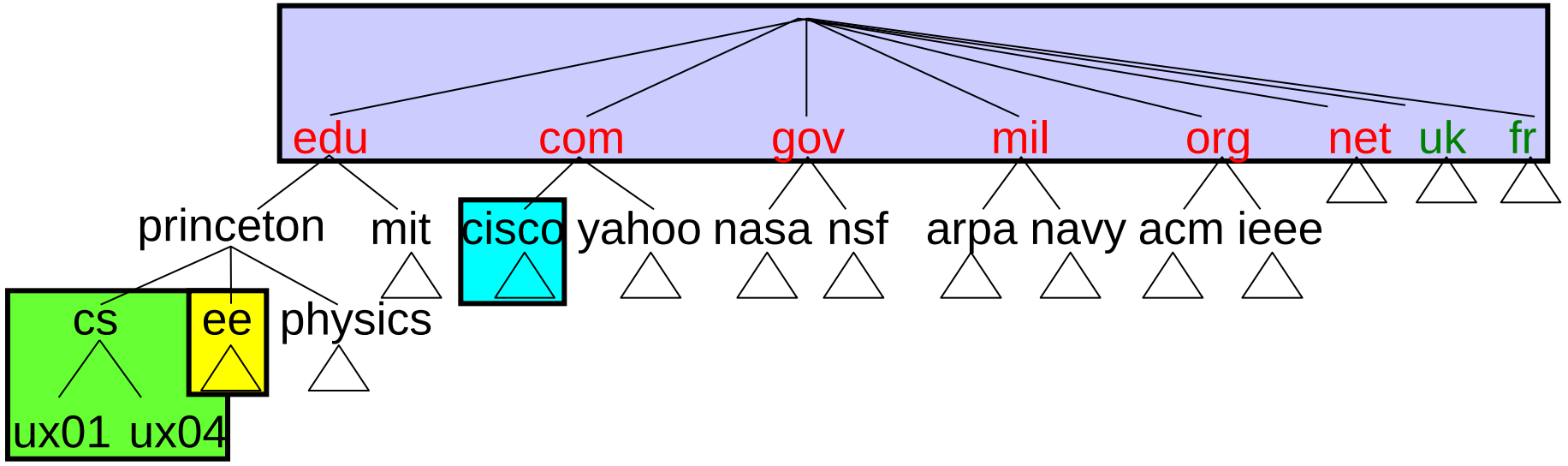
Domain Hierarchy

- DNS names are processed from **right to left** (ee.nthu.edu.tw)
- DNS maps domain names into values
  - We assume that these values are **IP addresses**
- Each node in the tree corresponds to a **domain**
  - The leaves in the tree correspond to the **hosts** being named
- There are domains for **each country**, plus **"big six" domains**:
  - **edu, com, gov, mil, org, and net**



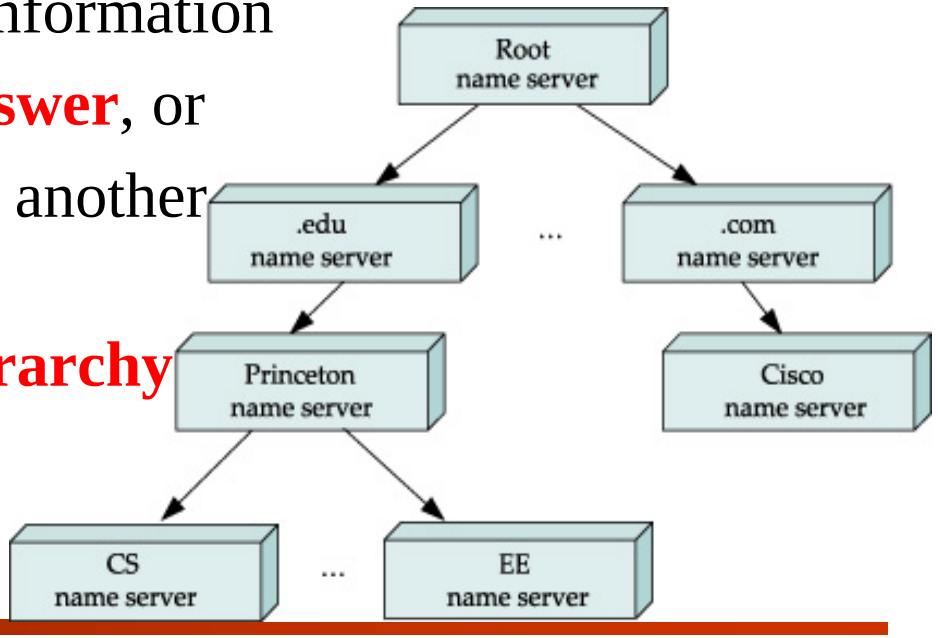
Name Servers

- How this hierarchy is actually implemented?
  - The first step is to partition the hierarchy into subtrees called **zones**
- Each zone can be thought of as corresponding to some **administrative authority**
  - Which is responsible for that portion of the hierarchy



Name Servers

- **Zone:** corresponds to the fundamental unit of implementation in DNS — the **name server**
- The information is implemented in **two or more** name servers
  - For the sake of **redundancy**
- Clients send queries to name servers, and name servers respond with the requested information
  - May contain the **final answer**, or
  - May contain a **pointer** to another server for next query
- DNS is represented by a **hierarchy of name servers** rather than by a hierarchy of domains



Name Servers

- The zone information is implemented as resource records
  - A resource record is a **name-to-value** binding
  - A 5-tuple that contains the following fields:

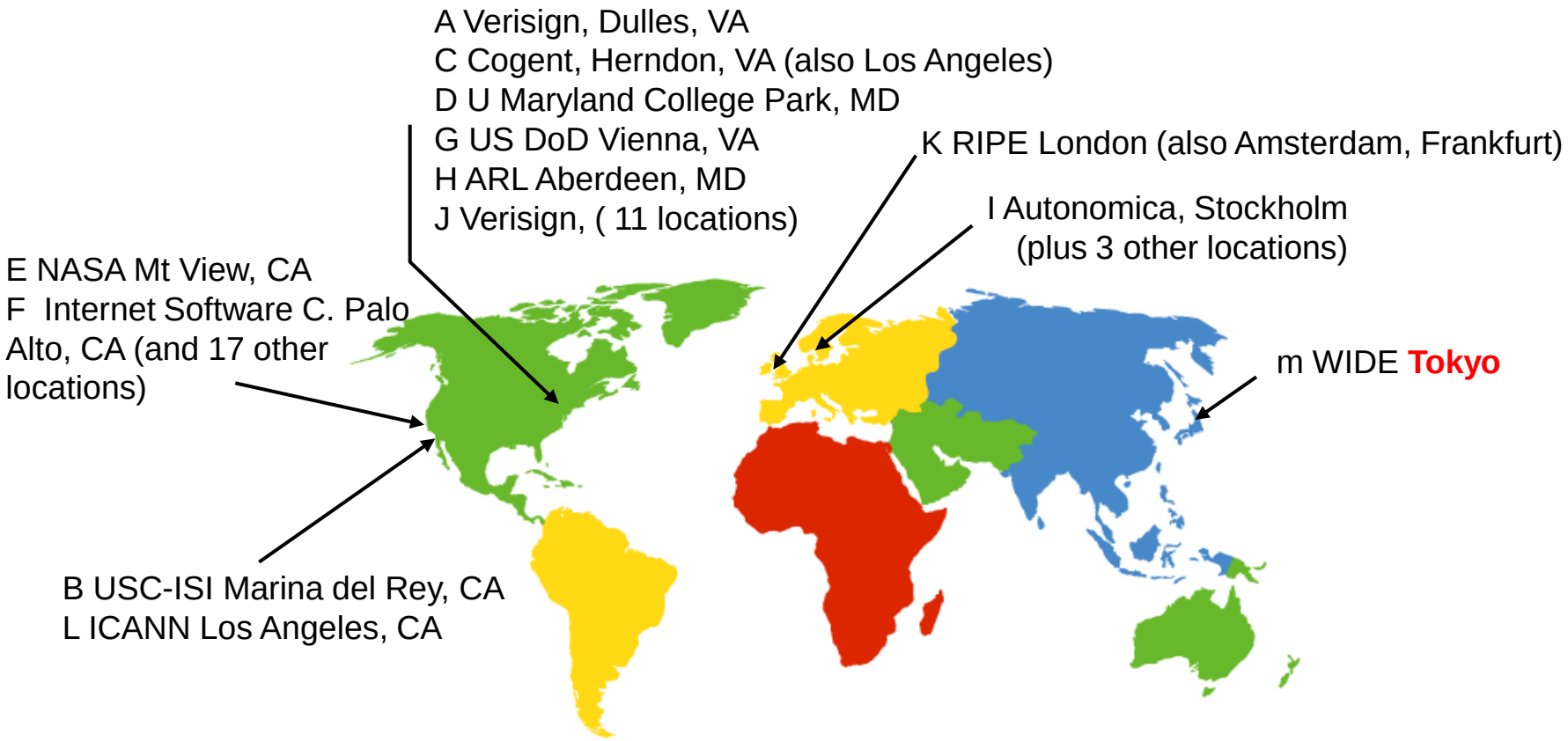
    <Name, Value, Type, Class, TTL>
- **Type:** specifies how the **Value** should be interpreted
  - **A: Value** is an **IP address**
  - **NS: Value** gives **the domain name for a name server** that knows how to resolve names within the domain
  - **CNAME: Value** gives the **canonical name** for a particular host
  - **MX: Value** gives **the domain name for a mail server** that accepts messages for the specified domain

Name Servers

- **Class:** allows entities to define useful record types
  - **IN:** the only widely used Class which is used by the Internet
- **TTL:** shows how long this resource record is valid
  - When the TTL expires, the server must erase the record from its cache

Root Servers

- **13** root servers (see **http://www.root-servers.org/**)
  - Labeled **A** through **M**



Name Servers **(Root Server)**

- First, the root name server contains an **NS record** for each **second-level server**
- It also has an **A record** that translates this name into the corresponding **IP address**
- These two records effectively implement a **pointer** from the root name server to each of the second-level servers

< princeton.edu, cit.princeton.edu, NS, IN>

<cit.princeton.edu, 128.196.128.233, A, IN>

Second-level server

<cisco.com, ns.cisco.com, NS, IN>

<ns.cisco.com, 128.96.32.20, A, IN>

...

Name Servers **(Second-Level Server)**

- The domain **princeton.edu** has a name server available on host **cit.princeton.edu** that contains the following records
  - Some records give the **final answer (IP addresses)**
  - Others point to **third-level name servers**

<cs.princeton.edu, gnat.cs.princeton.edu, NS, IN>

<gnat.cs.princeton.edu, 192.12.69.5, A, IN>

<ee.princeton.edu, helios.ee.princeton.edu, NS, IN>

<helios.ee.princeton.edu, 128.196.28.166, A, IN>

<jupiter.physics.princeton.edu, 128.196.4.1, A, IN>

<saturn.physics.princeton.edu, 128.196.4.2, A, IN>

<mars.physics.princeton.edu, 128.196.4.3, A, IN>

<venus.physics.princeton.edu, 128.196.4.4, A, IN>...

Name Servers **(Third-Level Server)**

- Finally, a third-level name server, such as the one managed by domain **cs.princeton.edu**, contains **A records** for **all** of its hosts
- It might also define **a set of aliases (CNAME records)** for each of those hosts
  - Aliases are sometimes just **convenient (e.g., shorter) names** for machines
  - For example, **www.cs.princeton.edu** is an alias for the host named **cicada.cs.princeton.edu**

Name Servers **(Third-Level Server)**

<cs.princeton.edu, gnat.cs.princeton.edu, MX, IN>

<cicada.cs.princeton.edu, 192.12.69.60, A, IN>

<cic.cs.princeton.edu, cicada.cs.princeton.edu, CNAME,IN>

<gnat.cs.princeton.edu, 192.12.69.5, A, IN>

<gna.cs.princeton.edu, gnat.cs.princeton.edu, CNAME, IN>

<www.cs.princeton.edu, 192.12.69.35, A, IN>

<cicada.cs.princeton.edu, roach.cs.princeton.edu, CNAME, IN>

...

Name Resolution **(Root Server)**

- Suppose the client wants to resolve **cicada.cs.princeton.edu**
- The client first sends a **query** containing this name to the **root server**
- The root server, unable to match the entire name,
  - Returns **the best match** it has — the **NS record** for **princeton.edu**
- The root server also returns **all** records that are related to this record
  - The **A record** for **cit.princeton.edu**

    < princeton.edu, cit.princeton.edu, NS, IN>

    <cit.princeton.edu, 128.196.128.233, A, IN>

Name Resolution **(Second-Level Server)**

- The client next sends **the same query** to the name server at IP host **128.196.128.233**
- This server also **cannot match** the whole name
  - Returns the **NS** and corresponding **A records** for the **cs.princeton.edu** domain

<cs.princeton.edu, gnat.cs.princeton.edu, NS, IN>

<gnat.cs.princeton.edu, 192.12.69.5, A, IN>

Name Resolution **(Third-Level Server)**

- Finally, the client sends **the same query** as before to the server at IP host **192.12.69.5**
- This time gets back the **A record** for **cicada.cs.princeton.edu**

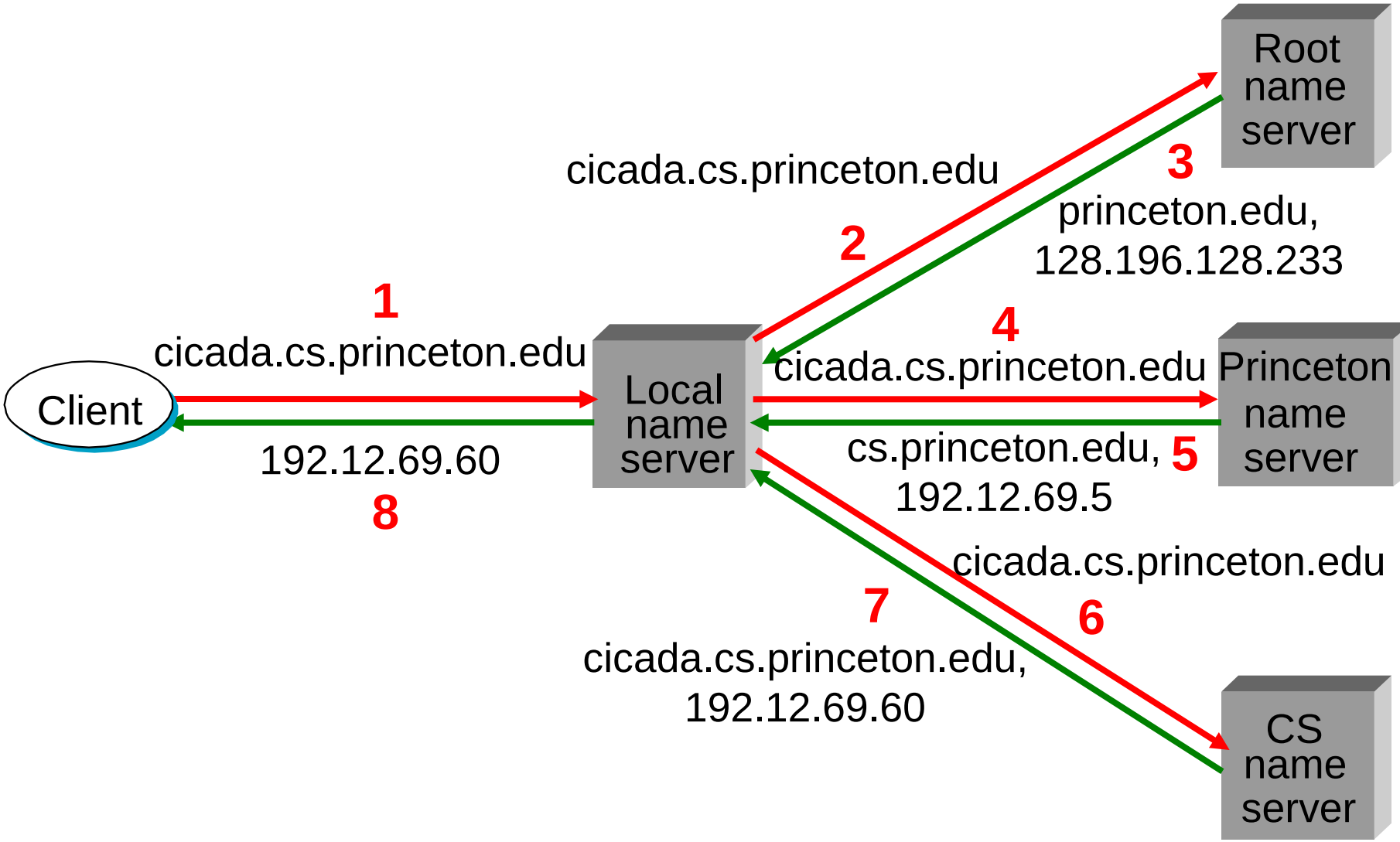
  <cicada.cs.princeton.edu, 192.12.69.60, A, IN>

- In practice, not all clients know about the root servers
  - The client program running on each Internet host is **initialized** with the address of a **local name server**
- For example, a local name server **gnat.cs.princeton.edu**

  <'root', venera.isi.edu, NS, IN>

  <venera.isi.edu, 128.9.0.32, A, IN>

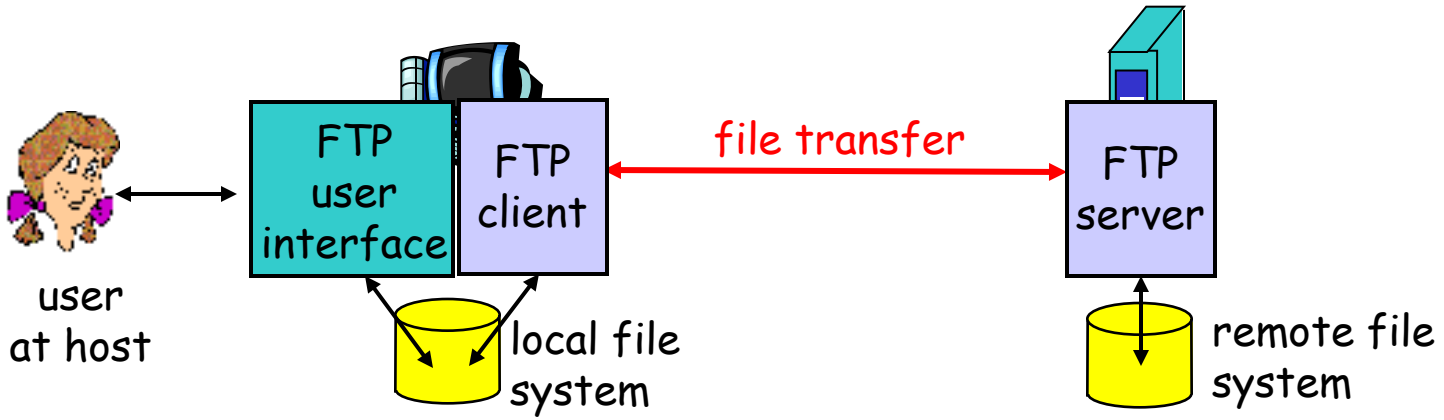
Name Resolution



Simple Network Management Protocol (SNMP)

- Allow us to read, write state information on different network nodes
- It is a request/reply protocol: GET and SET
- SNMP runs on top of UDP
- Management information base (MIB) -- A MIB is a collection of managed objects residing in a virtual information store.
  - 10 different groups in MIB-II
  - for example: System, Interfaces, Address Translation, IP. TCP, UDP
- Presentation format: ASN.1

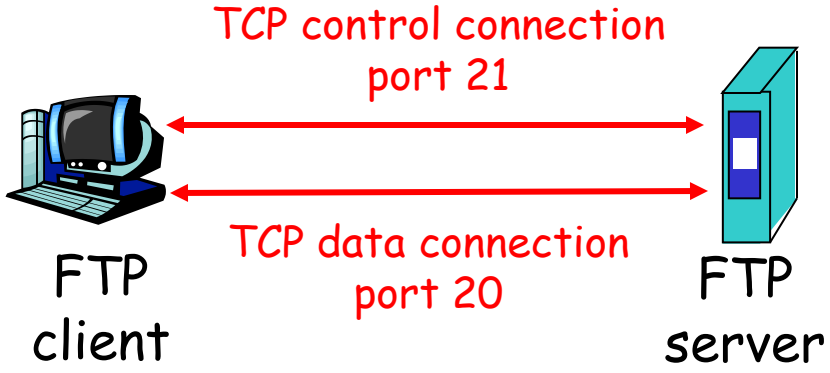
ftp: the file transfer protocol



- transfer file to/from remote host
- client/server model
  - *client:* side that initiates transfer (either to/from remote)
  - *server:* remote host
- ftp: RFC 959
- ftp server: port 21

ftp: separate control, data connections

- ftp client contacts ftp server at port 21, specifying TCP as transport protocol
- two parallel TCP connections opened:
  - control: exchange commands, responses between client, server.
    "out of band control"
  - data: file data to/from server
- ftp server maintains "state": current



ftp commands, responses

Sample commands:

- sent as ASCII text over control channel
- **USER *username***
- **PASS *password***
- **LIST** return list of file in current directory
- **RETR filename** retrieves (gets) file
- **STOR filename** stores (puts) file onto remote host

Sample return codes

- status code and phrase (as in http)
- **331 Username OK, password required**
- **125 data connection already open; transfer starting**
- **425 Can't open data connection**
- **452 Error writing file**

Web Services

- SOAP (Simple Object Access Protocol)
  - It relies on Extensible Markup Language (XML) as its message format
  - It relies on other Application Layer protocols (most notably Remote Procedure Call (RPC) and HTTP) for message negotiation and transmission
  - SOAP can form the foundation layer of a web services protocol stack, providing a basic messaging framework upon which web services can be built.
  - The SOAP architecture consists of several layers of specifications for message format, message exchange patterns (MEP), underlying transport protocol bindings, message processing models, and protocol extensibility.
  - Partial stardards(profiles): WS-I Basic Profile

Web Services

- REpresentational State Transfer (REST)
- Individual web services are regarded as resources identified by URIs and accessed via HTTP
- Competition with SOAP
  - The most important HTTP methods are POST, GET, PUT and DELETE. These are often respectively associated with the CREATE, READ, UPDATE, DELETE operations associated with database technologies
  - SOAP may have an advantage in adapting or wrapping previously written legacy applications to conform to web services
- 80% of Amazon traffic uses REST interface

Multimedia Applications

Multimedia Applications

- We need to develop a number of general-purpose protocols for use by **multimedia applications**
  - **QoS signaling protocol:** To request the **allocation of resources** so that the desired QoS can be provided
    - **RSVP (Resource Reservation Protocol)**
  - **Transport protocol:** With rather different characteristics than TCP and with more functionality than UDP
    - **Real-time Transport Protocol (RTP)** (Chapter 5)
  - **Session control protocol:** For example, to make **IP-based telephone calls** across the Internet
    - **SIP (Session Initiation Protocol)**
    - **H.323**

Session Control and Call Control

Session Control and Call Control

- Suppose you want to hold a videoconference at a certain time and make it available to a wide number of participants
  - Use the **multicast IP address** for transmission
  - Send it using **RTP over UDP** port number 4000
- A working group (**Multiparty Multimedia Session Control group**) has defined protocols for this purpose
  - **SDP (Session Description Protocol)**
  - **SAP (Session Announcement Protocol)**
  - **SIP (Session Initiation Protocol)**
  - **SCCP (Simple Conference Control Protocol)**

Session Description Protocol (SDP)

- The name and the purpose of the session
- Start and end time for the session
- The media type (e.g., audio, video) that comprise the session
- Detailed information needed to receive the session (e.g., the multicast address, the transport protocol, the port numbers, the encoding schemes)
- SDP provides the information formatted in ASCII
- SDP sends out the session information
- Still need SIP to locate the user and negotiate the encoding schemes.

Session Control and Call Control (SIP)

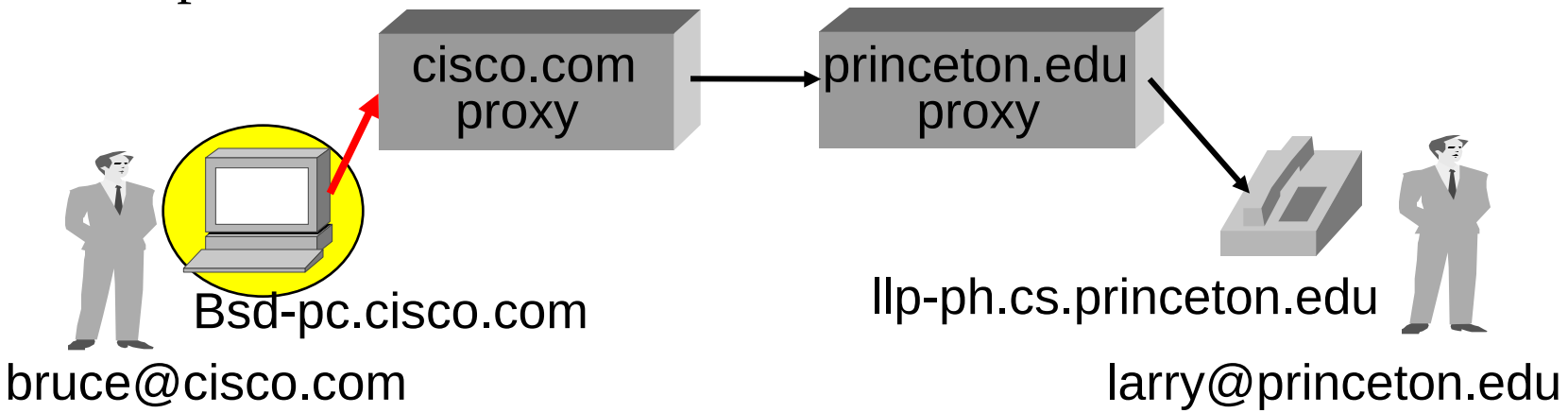
- **SIP** is an application-layer protocol based on a similar **request/response** model
- The SIP capabilities are grouped into five categories:
  - **User location:** determines the **correct device** with which to communicate to reach a particular user
  - **User availability:** determine if the user is **willing or able to** take part in a particular communication session
  - **User capabilities:** determine such items as the choice of **media** and **coding scheme** to use
  - **Session setup:** establish **session parameters** such as port numbers to be used by the communicating parties
  - **Session management:** has a range of functions including **transferring** sessions and **modifying** session parameters

Session Control and Call Control (SIP)

- **SIP** is primarily used for **human-to-human** communication
  - Need to locate **individual users**, not just machines
  - Need to know **where the user is right now**
- This is further complicated by the fact that a user might choose to communicate using a range of **different devices**
- The user must be able to have **control** over when, where, and from whom he receives calls
- **SIP proxy:**
  - Enables a user to have control over his calls
  - Proxies also perform functions **on behalf of callers**

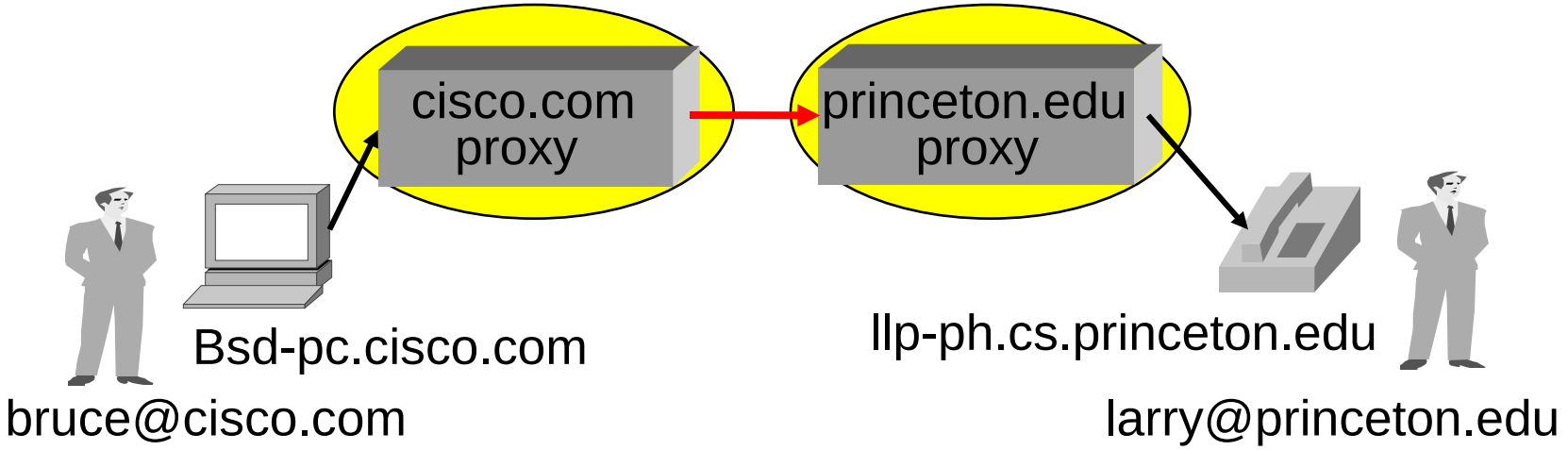
Session Control and Call Control (SIP)

- Each user has a name in the format **user@domain**
- User Bruce wants to initiate a session with Larry
  - Sends his **initial SIP message** to the **local proxy** for his domain — cisco.com
  - This initial message contains a SIP URI (uniform resource identifier) ⇒ **SIP: larry@princeton.edu**
- SIP URI provides complete **identification** of a user, but **does not** provide his **location**



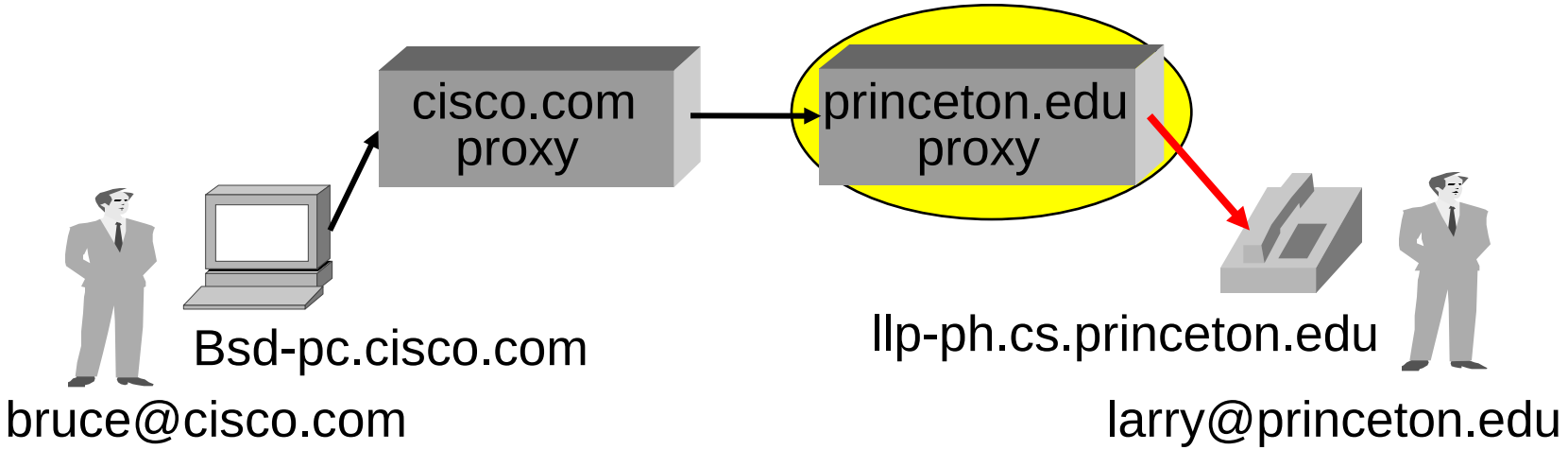
Session Control and Call Control (SIP)

- The cisco.com proxy looks at the SIP URI and deduces that this message should be **sent to the princeton.edu proxy**
- Assume that the princeton.edu proxy has a mapping from the name larry@princeton.edu to the **IP address** of **one or more** devices at which Larry **currently** wishes to receive messages



Session Control and Call Control (SIP)

- The proxy can therefore forward the message to Larry's **chosen device(s)**
- (Forking) Sending the message to **more than one device** may be done either **in parallel** or **in series**
  - e.g., send it to his cellphone if he doesn't answer the phone at his desk



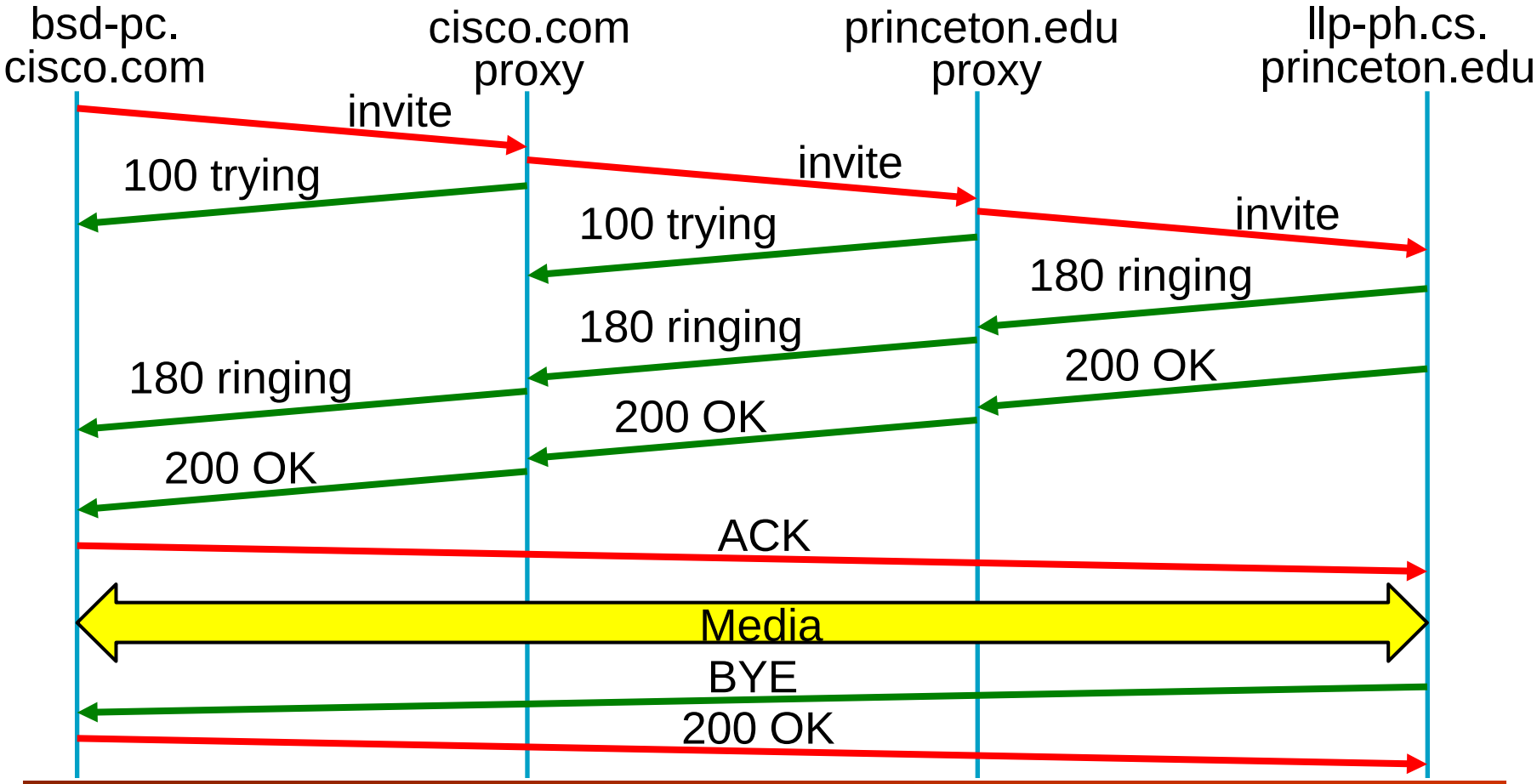
Session Control and Call Control (SIP)

- The initial message from Bruce to Larry is likely to be a **SIP invite message**
- **Via:** header in this example identifies the **device** from which this message originated
- **Content-Type:** and **Content-Length:** headers describe the contents of the message following the header

```
INVITE sip:larry@princeton.edu SIP/2.0
Via: SIP/2.0/UDP bsd-pc.cisco.com;branch=z9hG4bK433yte4
To: Larry <sip:larry@princeton.edu>
From: Bruce <sip:bruce@cisco.com>;tag=55123
Call-ID: xy745jj210re3@bsd-pc.cisco.com
CSeq: 271828 INVITE
Contact: <sip:bruce@bsd-pc.cisco.com>
Content-Type: application/sdp
Content-Length: 142
```

Session Control and Call Control (SIP)

- **100 trying:** Indicates that the message was received **without error**



Session Control and Call Control (SIP)

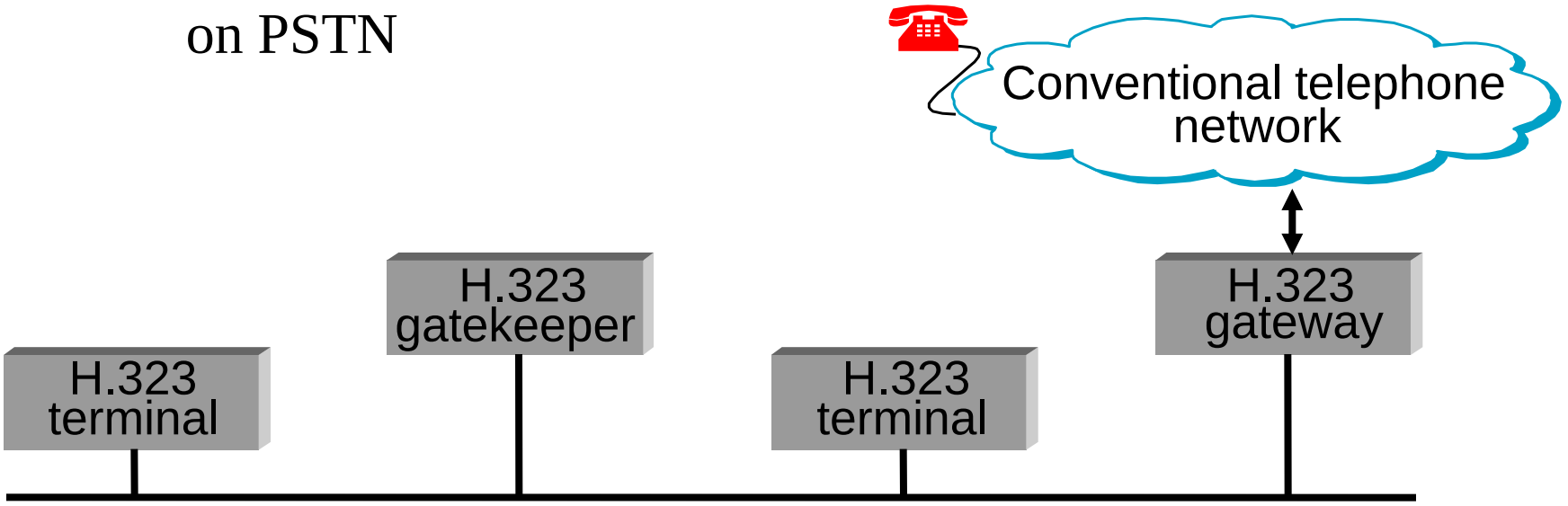
- **180 ringing:** is a sign that it can generate a **"ringtone"**
- **200 OK:** is sent when Larry pick up his phone
- **ACK:** Bruce's computer responds with an ACK
  - At this point the parties know each other's addresses, so the ACK can be sent **directly**, **bypassing the proxies**
- At this point media (e.g., an RTP-encapsulated audio stream) can begin to flow between the two parties
  - The **proxies** are **no longer involved** in the call
  - The media will typically **take a different path** through the network than the original signalling messages
- Even if one or both of the proxies were to **crash** at this point
  - The call could continue on **normally**
- **BYE:** when one party wishes to end the session

Session Control and Call Control (H.323)

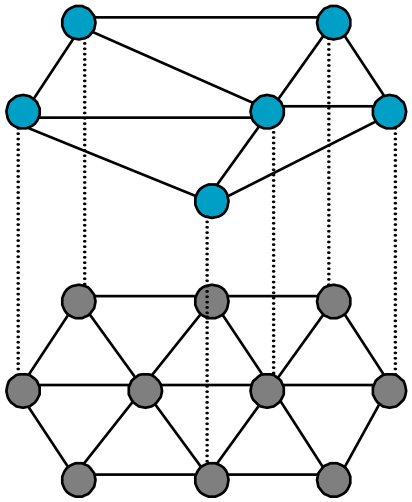
- **H.323:** The major **ITU** recommendation for multimedia communication over packet networks
  - H.323 is popular as a protocol for **Internet telephony**
- **H.323 terminal:** a device that originates or terminates calls
- The calls are frequently mediated by a **gatekeeper**
  - H.323 terminals can also talk to each other **directly**
- Gatekeepers perform a number of functions
  - **Translating** among the various **address formats**
  - **Controlling** how many calls can be placed at a given time to limit the **bandwidth** used by the H.323 applications
  - One useful function performed by the gatekeeper is to help a terminal **find a gateway**

Session Control and Call Control (H.323)

- A **gateway** of H.323 connects the H.323 network to **other types of networks**
  - Connects an H.323 network to the **public switched telephone network (PSTN)**
  - Enables a user running an H.323 application on a computer to talk to a person using a conventional phone on PSTN

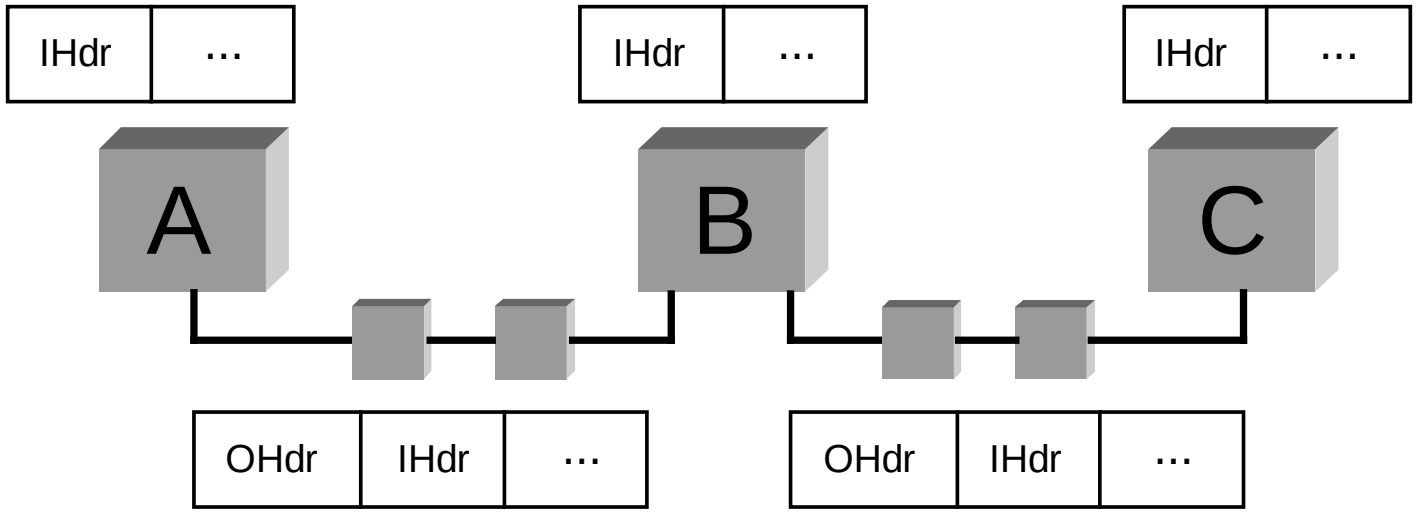


Overlay networks



- Building a network on the top of the Internet
- Virtual private networks (VPN)
- Experimental networks for new ideas (IPv6, M-bone, Q-bone)

Tunneling



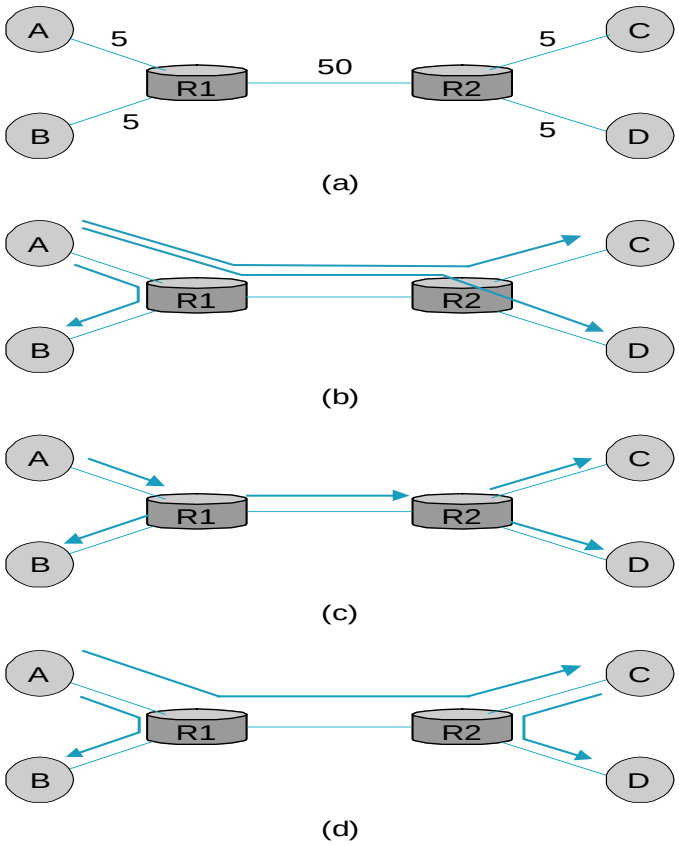
• Use Inner Header Address for the overlay netowrk

•Use the Outer Header Address for the Internet

Routing Overlays

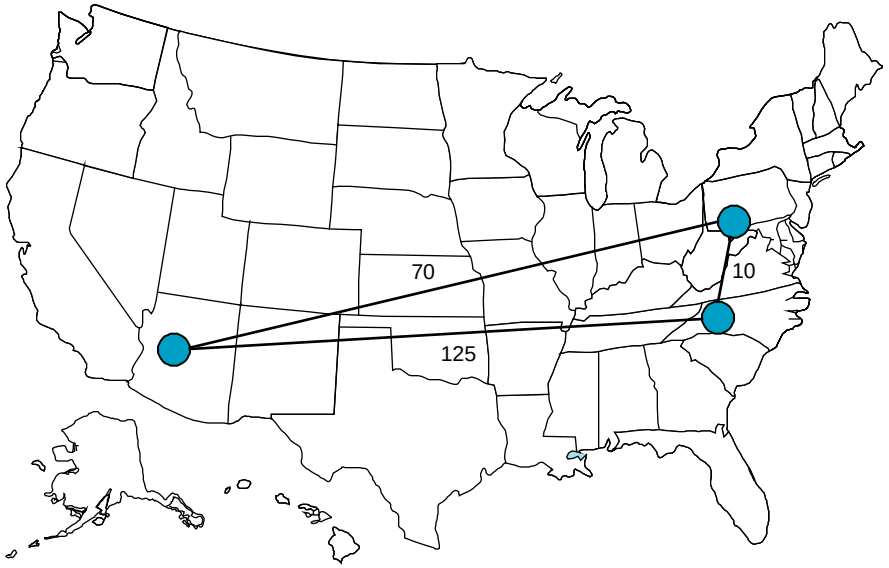
- Support an alternative routing strategy by using overlays
- Experimental versions of IP
  - The multicast backbone (MBone)
  - IPv6 (6-BONE)

End System Multicast



- IP multicast failed
- Long delay between R1 and R2
- (a) Physical topology
- (b) Naïve unicast
- © IP multicast
- (d) End system multicast

Resilient Overlay Networks



• Triangle inequality does not necessarily hold

•BGP dies select the best route and it is slow to adapt to outage

•RON uses an NxN monitoring for N nodes: latency, available bandwidth, and loss probability

Content Distribution Networks

Outline

- Implementation Techniques
- Hashing Schemes
- Redirection Strategies

Potential bottlenecks

- The first mile: slow access link, e.g., 56Kbps modem
- The last mile: server is overloaded by too many requests.
- Peering points: little motivation to provide high-capacity connectivity to other peers.

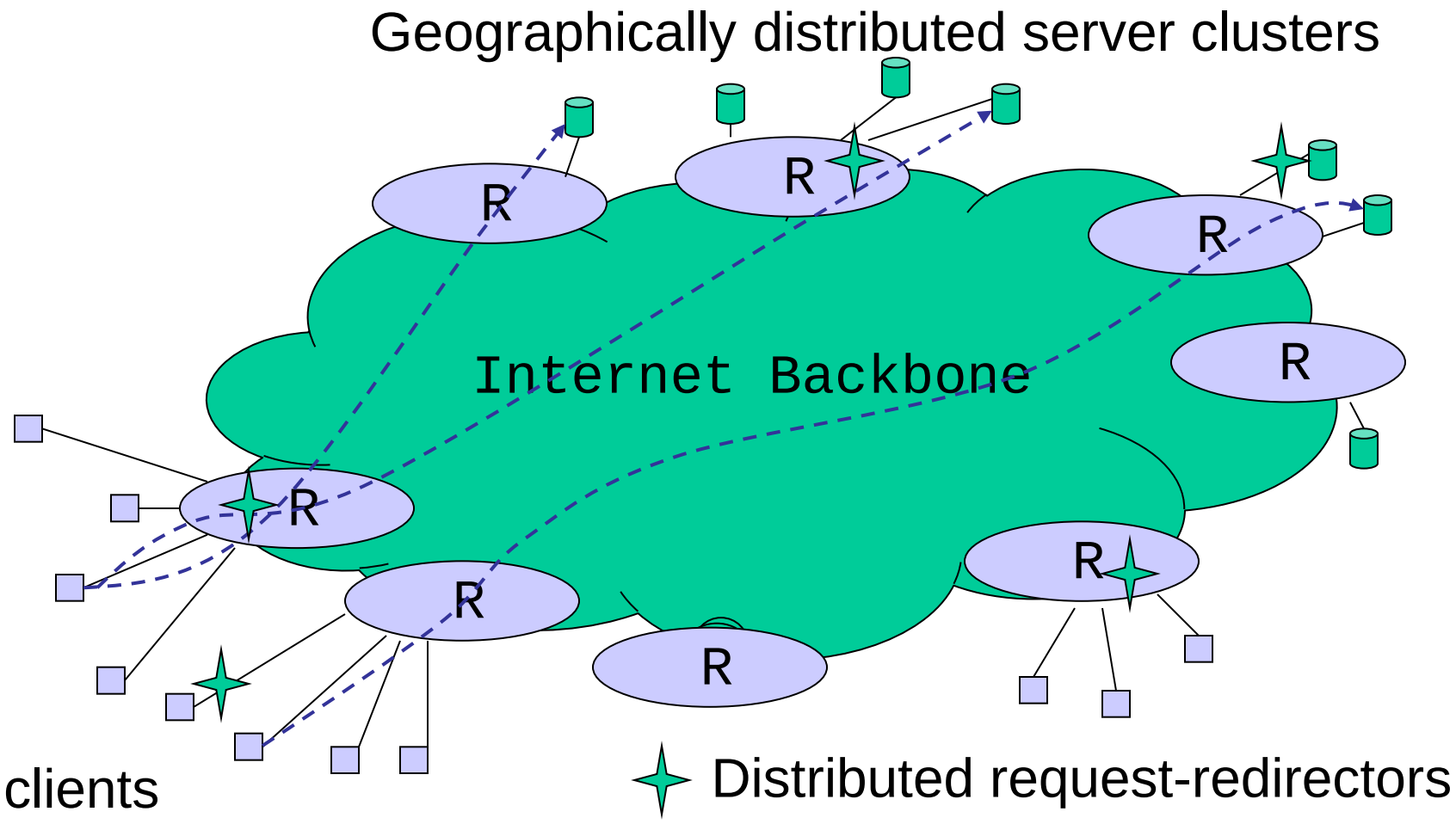
Design Space

- Caching
  - explicit
  - transparent (hijacking connections)
- Replication
  - server farms
  - geographically dispersed (CDN)

CDNs

- Flash crowd
- Redirectors (that forward client requests to the most appropriate server)
- The primary objective: the best response time
- A secondary objective: system throughput
- Factors for redirection:
  - Network proximity
  - Load balance

Redirection Overlay



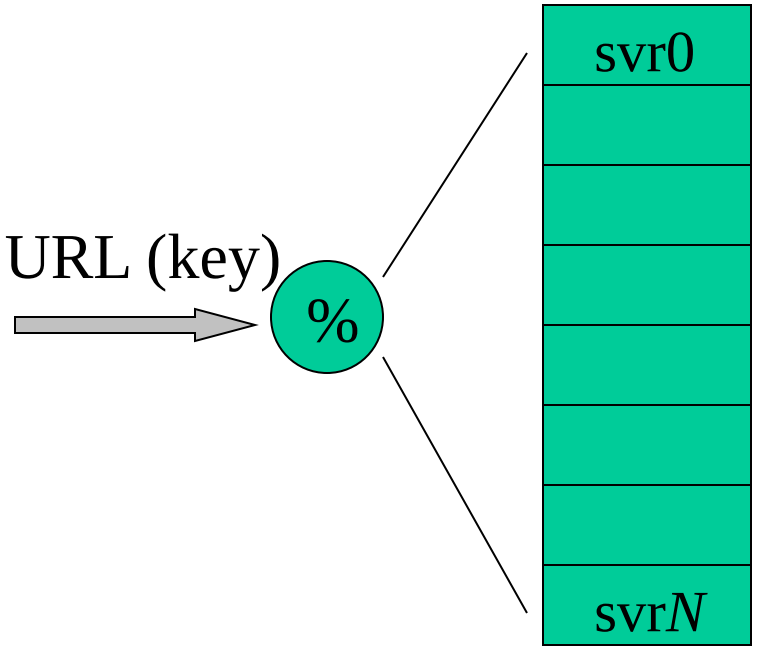
Redirection Mechanisms

- DNS
  - one name maps onto many addresses
  - Return the IP address of the most appropriate server (e.g., the server with the lightest load)
- URL Rewriting
  - embedded links pointing the client at the most appropriate server
- HTTP
  - requires an extra round trip
  - vulnerable to being overloaded by the redirection task itself

Redirection Policies

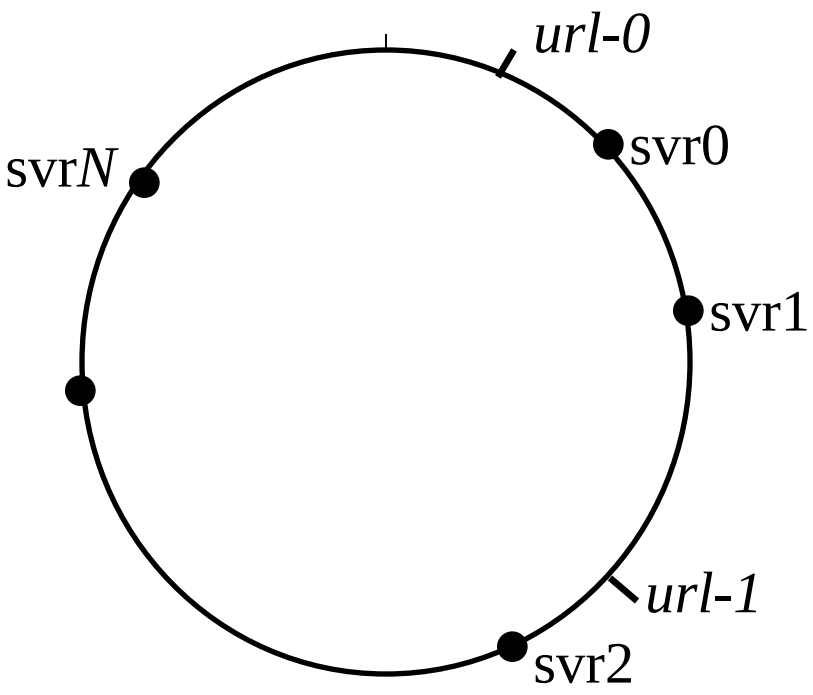
- Round-robin
- Random selection
- Need to take into network proximity and locality
- Redirect the same URL requests to go to the same server
  - Hashing (modulo)
  - Consistent hashing

Hashing Schemes: Modulo



- Easy to compute
- Evenly distributed
- Good for fixed number of servers
- Many mapping changes after a single server change

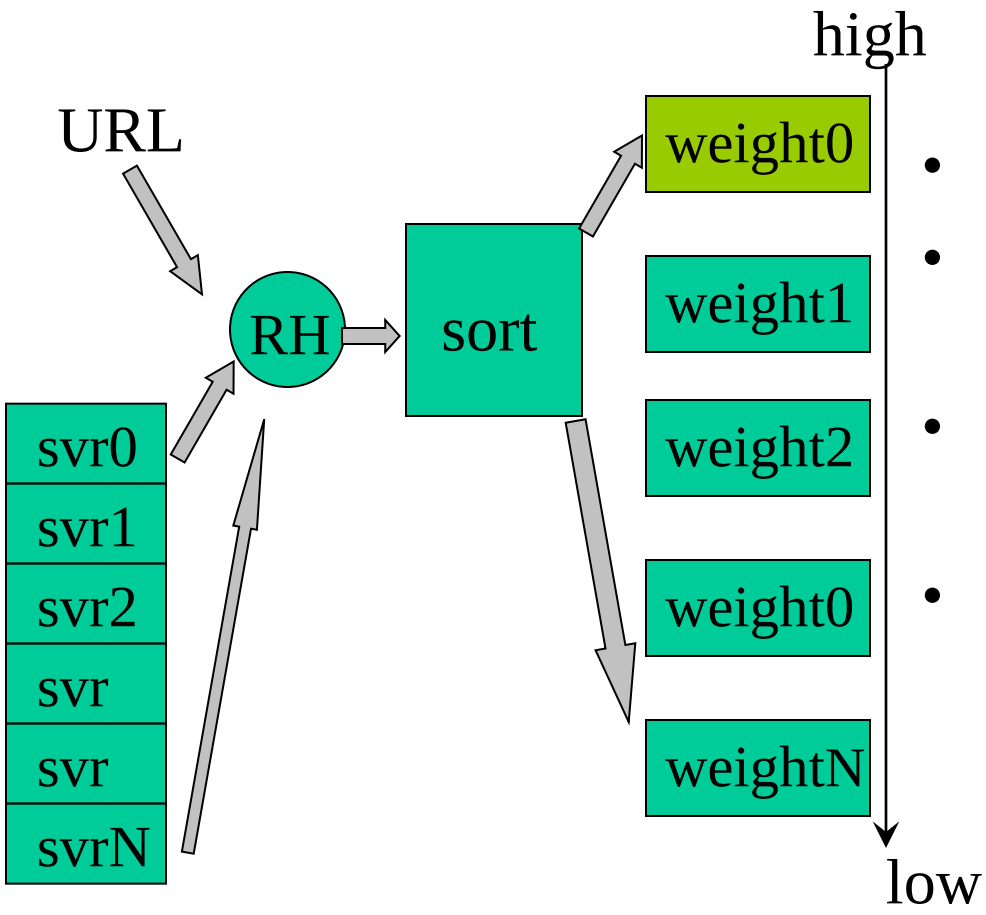
Consistent Hashing (CHash)



Unit circle

- Hash server, then URL
- Closest match
- Only local mapping changes after adding or removing servers
- Used by State-of-the-art CDNs

Highest Random Weight (HRW)



- Hash(url, svrAddr)
- Deterministic order of access set of servers
- Different order for different URLs
- Load evenly distributed after server changes

Cache Array Routing Protocol (CARP)

- SelectServer(URL,S)

  for each server s_i in server set S

  $weight_i$ =hash(URL,address(s_i))

  sort weight

  for each server s_j in decreasing order of $weight_j$

  If load(s_j) < threshold then

  Return s_j

  Return server with highest weight

CARP

- As the load increase. This scheme changes from using the first server on the sorted list to spreading requests across several servers.
- Adding network proximity into the equation: use the measured response time as the “server load.”