Module II

Network Programming And Applications

Topics

- Internet services and communication paradigms
- Client-server model and alternatives
- Network programming with a simplified API
- The socket API
- Application layer protocols
- Examples of standard application protocols

Internet Services And Communication Paradigms

General Principle: Intelligence At The Edge

The Internet does not provide services. Instead, the Internet only provides communication, and application programs provide all services.

- Consequence
  - Every Internet communication, including voice and video teleconferencing, involves communication among application programs

Communication Paradigms

- The Internet offers two communication paradigms

Stream Paradigm	Message Paradigm
Connection-oriented	Connectionless
1-to-1 communication	Many-to-many communication
Sequence of individual bytes	Sequence of individual messages
Arbitrary length transfer	Each message limited to 64 Kbytes
Used by most applications	Used for multimedia applications
Built on TCP protocol	Built on UDP protocol

- Each paradigm has surprising characteristics

Stream Paradigm (TCP)

- Transfers a sequence of bytes
- Connection-oriented: data sent between two applications
- Bidirectional (one stream in each direction)
- No meaning attached to data and no boundaries inserted in data
- Surprising characteristic:

Although it delivers all bytes in sequence, the stream paradigm does not guarantee that the chunks of bytes passed to a receiving application correspond to the chunks of bytes transferred by the sending application.

Message Paradigm (UDP)

- Connectionless: network accepts and delivers individual messages
- If the sender places *N* bytes in a message, a receiver will find exactly *N* bytes in the incoming message
- Paradigm allows unicast, multicast, or broadcast delivery (one destination, multiple destinations, or all destinations)
- Surprising characteristic:

Although it preserves boundaries, the message paradigm allows messages to be lost, duplicated, or delivered out-of-order; neither the sender nor receiver is informed when such errors occur.

Stream Transport And Data Chunks

- The protocol system may
  - Divide the data from the sender into multiple segments and deliver a few bytes at a time to the receiver
  - Combine data from multiple transmissions into a single large chunk and deliver it to the receiver all at once
- Consequence: receiving application cannot know exactly which pieces were sent

Example #1

- Assume a stream connection between two applications
- Sender
  - Places 1000-byte message in buffer *buf*
  - Makes a single request to send all 1000 bytes
- Receiver
  - Allocates a buffer *b* with 1000 bytes
  - Reads 1000 bytes from the stream into buffer *b*
- The OS may return between 1 and 1000 bytes
- Application *must* make repeated calls until all 1000 bytes have been acquired

Example #2

- Assume a stream connection between two applications
- Sender transmits a sequence of four messages that are each 100 bytes long
- Receiver allocates a large buffer *b* of 1000 bytes and requests that up to 1000 bytes from stream be read into buffer *b*
- The OS may choose to return all four messages (400 bytes) with a single read request
- Receiving application *must* be able to separate received data into four separate messages

Programming Hints

- When using the stream paradigm
  - Devise a way that a receiver knows where a message ends
  - Read from a socket until the entire message has been acquired
- When considering using the message paradigm
  - Don’t (at least not yet)

Identifying Individual Messages In A Stream

- Possibilities
  - Send exactly one message followed by *end of file (EOF)*
  - Send multiple messages with an integer length before each message
  - Send multiple messages with a termination character (or sequence) following each message
- Notes
  - Any technique can be used as long as both sides agree
  - If sending a multi-byte length value or multi-byte termination sequence, remember that the application may need multiple calls receive all bytes

Questions

- In a realistic setting
  - Is division of a message likely to occur?
  - Is aggregation of multiple messages likely to occur?
- Answers yes! (depending on the size of the messages)
  - Messages larger than 1400 characters are usually divided into multiple packets for transmission, and *may* be delivered together or separately
  - The stream service is designed to aggregate small messages before making them available to a receiving application

Buffering In The Stream Paradigm

- Aggregation, which makes bulk transfer more efficient, can occur on the sending or receiving side
- The stream paradigm includes a *push* operation that an application can use to force transmission and delivery
- Unix convention: automatically *push* for each individual *write* call
- Programming hints
  - To ensure a small message is transmitted and delivered without delay, use a separate *write*
  - Even with *push*, network delays mean applications must be written to tolerate aggregation
- More details later in the course

Client-Server Model
And Alternatives

Client-Server Model Of Interaction

- Used by applications to establish communication
- One application acts as a *server*
  - Starts execution first
  - Awaits contact
- The other application becomes a *client*
  - Starts after server is running
  - Initiates contact
- Important concept: once communication has been established, data (e.g., requests and responses) can flow in either direction between a client and server

Characteristics Of A Client

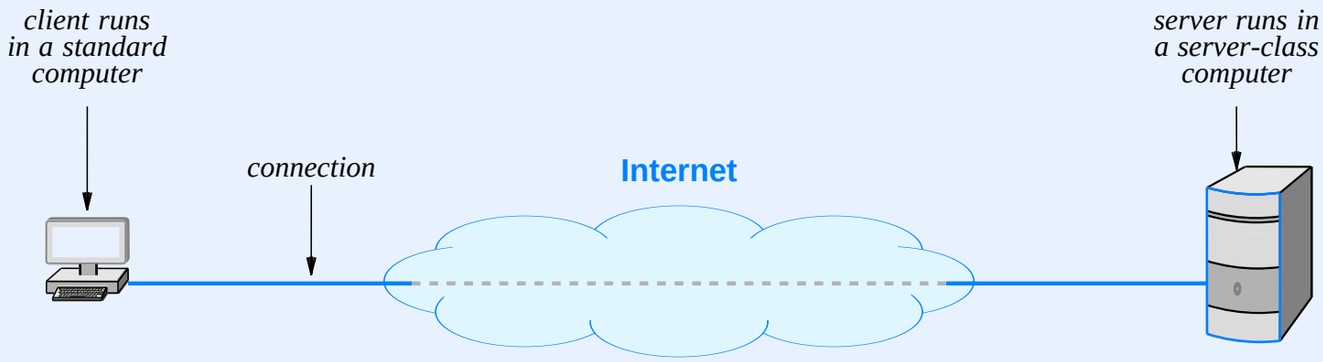
- Arbitrary application program that becomes a client temporarily
- Usually invoked directly by a user, and usually executes only for one session
- Actively initiates contact with a server, exchanges messages, and then terminates contact
- Can access multiple services as needed, but usually contacts one remote server at a time
- Runs locally on a user's personal computer or smart phone
- Does not require especially powerful computer hardware

Characteristics Of A Server

- Special-purpose, privileged program dedicated to providing a service
- Usually designed to handle multiple remote clients at the same time — complicates the design
- Invoked automatically when a system boots, and continues to execute through many client sessions
- Waits passively for contact from arbitrary remote clients and then exchanges messages
- Requires powerful hardware and a sophisticated operating system
- Runs on a large, powerful computer

Server Programs And Server-Class Computers

- Confusion exists between scientific and marketing terminology
- Scientific: a *client* and a *server* are each programs
- Marketing: a *server* is a powerful computer



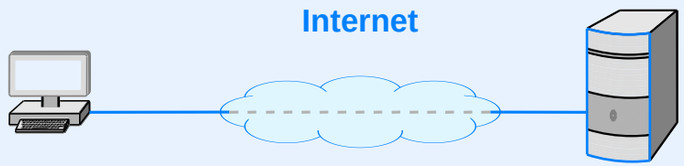
Summary Of Client-Server Interaction

Server Application	Client Application
Starts first	Starts second
Does not need to know which client will contact it	Must know which server to contact
Waits passively and arbitrarily long for contact from a client	Initiates a contact whenever communication is needed
Communicates with a client by sending and receiving data	Communicates with a server by sending and receiving data
Stays running after servicing one client, and waits for another	May terminate after interacting with a server

Illustration Of Steps Taken By Client And Server

Client Side

- Agree a priori on a port number, N
- Start after server is already running
- Obtain server name from user
- Use DNS to translate name to IP address
- Contact server using IP address and port N
- Interact with server and then exit



Server Side

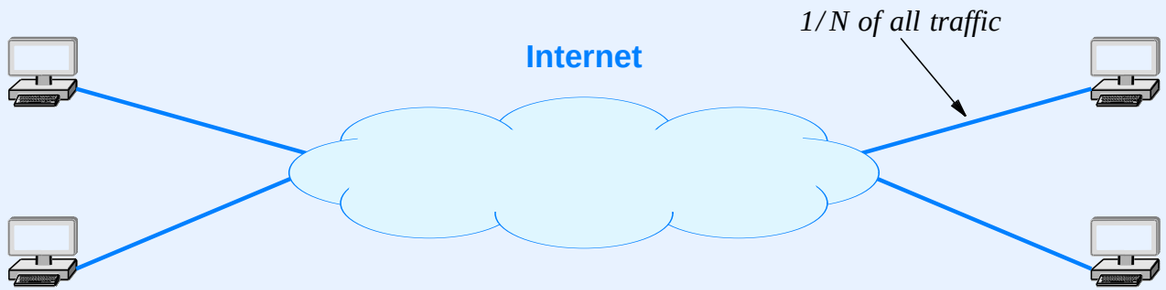
- Agree a priori on a port number, N
- Start before any of the clients
- Register port N with the local system
- Wait for contact from a client
- Interact with client until client finishes
- Wait for contact from the next client…

Alternatives To Client-Server

- Broadcast
  - Sender broadcasts message and all stations receive it
  - Does not scale well (becomes inefficient)
  - Difficult to restrict data access
- Rendezvous point
  - Intermediary connects communicating applications
  - In essence, there are two clients and a server
  - Rendezvous point becomes a bottleneck

Alternatives To Client-Server
(continued)

- Peer-To-Peer Interaction
  - Designed to avoid central server bottleneck
  - Data divided among *N* computers
  - Each computer acts as a server for its data and as a client for other data
  - Given computer receives *1 / N* of the traffic



Network Programming With A Simplified API

Network Programming

- General term that refers to the creation of client and server applications that communicate over a network
- Programmer uses an *Application Program Interface* (*API*)
  - Set of functions
  - Include control as well as data transfer functions (e.g., establish and terminate communication)
- Defined by the operating system; not part of the Internet standards
- *Socket API* has become a de facto standard

A Simplified API

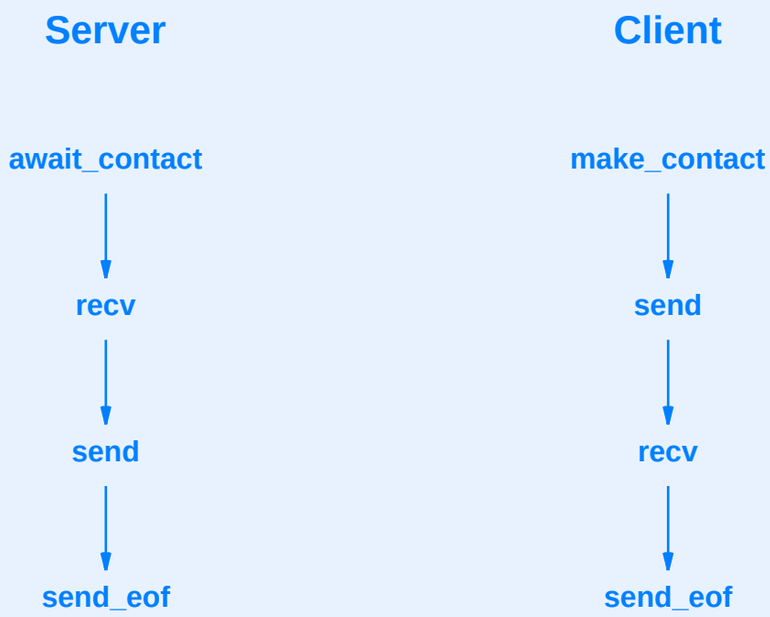
- Will help you get started
- General idea
  - Server is identified by pair (computer, application)
  - Server starts first and waits for contact
  - Client specifies server's location
  - Once a connection is established, client and server can exchange data
- Only seven functions in the simplified API

Our Simplified API

Operation	Meaning
await_contact	Used by a server to wait for contact from a client
make_contact	Used by a client to contact a server
appname_to_appnum	Used to translate a program name to an equivalent internal binary value
cname_to_comp	Used to translate a computer name to an equivalent internal binary value
send	Used by either client or server to send data
recv	Used by either client or server to receive data
send_eof	Used by both client and server after they have finished sending data

Client And Server Using The API

- Sequence of calls for a trivial exchange in which a client sends a single request and the server responds



- Both sides must call *send_eof* because communication is bidirectional

Data Types For Our Simplified API

Type Name	Meaning
appnum	A binary value used to identify an application
computer	A binary value used to identify a computer
connection	A value used to identify the connection between a client and server

An Extra Function For Convenience

- Simplified API includes an extra function, *recvln*
- Not required, but convenient
- Similar to *recv*
  - Receives data from a connection
  - Places data in a buffer
- Difference
  - Reads *exactly* the amount requested
  - Technique: repeatedly call *recv* until specified length has been acquired

Argument Types For Our API

Function Name	Type Returned	Type of arg 1	Type of arg 2	Type of args 3–4
await_contact	connection	appnum	–	–
make_contact	connection	computer	appnum	–
appname_to_appnum	appnum	char *	–	–
cname_to_comp	computer	char *	–	–
send	int	connection	char *	int
recv	int	connection	char *	int
recvln	int	connection	char *	int
send_eof	int	connection	–	–

- You will learn more in the PSOs

The Socket API

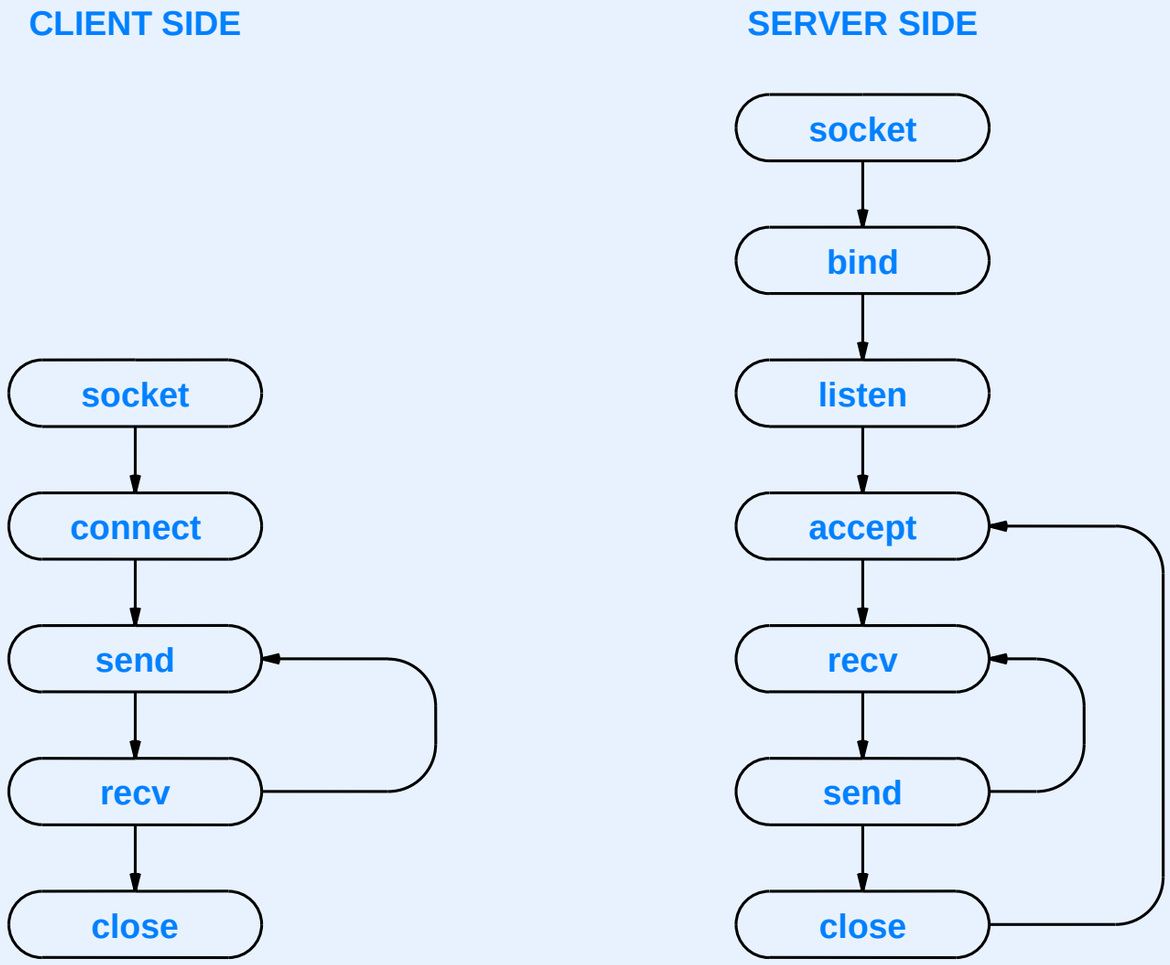
Sockets

- Originally part of BSD Unix
- Now standard in the industry
- AT&T defined an alternative named *TLI* (*Transport Layer Interface*), but TLI is now extinct
- Almost every OS includes an implementation
- MS Windows chose to make minor changes (annoying)

Socket Characteristics

- Socket can be used for
  - Connectionless communication (UDP message)
  - Connection-oriented communication (TCP stream)
- Many functions in the API
- Approach
  - Create a socket
  - Make many function calls to specify type of communication, remote computer's address, port number to be used, etc.
  - Use socket to send / receive data
  - Close the socket (terminate use)

Example Socket Calls For Stream Communication



Application Layer Protocols

Terminology

- Availability of an application protocol
  - *Closed* — vendor defines a protocol for their products
  - *Open* — standardized and available for all vendors
- Basic protocol types
  - *Data representation* — message and data formats
  - *Data transfer* — procedures for exchanging messages and handling unexpected / error conditions
- Notes
  - Application may define separate protocol for each type
  - Term *Transfer* in a protocol title indicates the latter

Defining An Application Layer Protocol

- Programmer specifies representation
  - Format of each message and each data item
  - Meaning of each item in a message
- Programmer specifies transfer
  - Which side sends first
  - Which side closes the connection first
  - What to do if one side crashes unexpectedly

State In An Application Protocol

- Big decision: should state information be kept?
- Stateful protocol assumes previous requests have been honored
- Stateless protocol assumes each request is independent
- Example of stateful interaction
  - Request 1 specifies “read from file X”
  - Request 2 specifies “read next 128 bytes”
- Example of stateless interaction
  - Request 1 specifies “read bytes 0-127 from file X”
  - Request 2 specifies “read bytes 128-255 from file X”

Examples Of Standard Application Protocols

Application Protocol Examples

- Web browsing
- Email
- File transfer
- Remote login and remote desktop
- Domain Name System (name lookup)

Application-Layer Protocols For The Web

Standard	Purpose
HyperText Markup Language (HTML)	A representation standard used to specify the contents and layout of a web page
Uniform Resource Locator (URL)	A representation standard that specifies the format and meaning of a web page identifier
HyperText Transfer Protocol (HTTP)	A transfer protocol that specifies how a browser interacts with a web server to transfer data

- Reminder: keyword *Transfer* in the name of a protocol means the protocol specifies message exchange

HyperText Markup Language (HTML)

- Representation standard for multimedia documents
- Specifies document is entirely in printable text
- Uses declarative rather than procedural approach
- Document includes *metadata* that can link to arbitrary item
- Document contains markup guidelines rather than precise, detailed formatting or typesetting instructions
  - Page can be displayed on arbitrary device
  - Appearance depends on device
- Embedded *tags* control display
  - Form is <*tag_name*> and </*tag_name*>

Uniform Resource Locator (URL)

- Representation standard
- A text string with punctuation characters separating the string into (optional) subfields
- General form is:

protocol : // *computer_name* : *port* / *document_name* ? *parameters*

- Example where protocol, port, and parameters are omitted:

www . cs . purdue . edu / people / comer

HyperText Transfer Protocol (HTTP)

- Transfer protocol used with the Web
- Specifies format and meaning of messages
- Each message represented as text
- Transfers arbitrary binary data
- Can download or upload data
- Incorporates caching for efficiency
- Browser sends *request* to server

Four Major HTTP Request Types

Request	Description
GET	Requests a document; server responds by sending status information followed by a copy of the document
HEAD	Requests status information; server responds by sending status information, but does not send a copy of the document
POST	Sends data to a server; the server appends the data to a specified item (e.g., a message is appended to a list)
PUT	Sends data to a server; the server uses the data to completely replace the specified item (i.e., overwrites the previous data)

- GET request has the form:

GET */item version CRLF*

- Version is HTTP/1.0 or HTTP/1.1

HTTP Response

- Response begins with a header in text, optionally followed by an item (which can be binary)
- Header uses *keyword: information* form like email header
- Header ends with a blank line

HTTP Header Format

- General form

`HTTP/1.0` *status_code status_string CRLF*
`Server:` *server_identification CRLF*
`Last-Modified:` *date_document_was_changed CRLF*
`Content-Length:` *datasize CRLF*
`Content-Type:` *document_type CRLF*
CRLF
... item begins here and contains datasize bytes ...

Telnet Example (Apache Web Server)

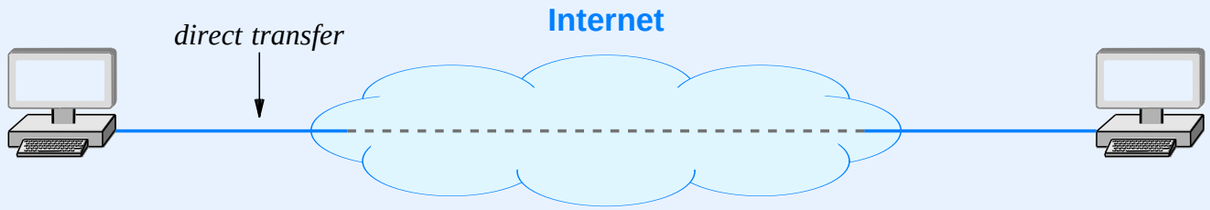
```
$  telnet www.cs.purdue.edu 80
Trying 128.10.19.20...
Connected to lucan.cs.purdue.edu.
Escape character is '^]'.
GET /homes/comer/ HTTP/1.0
HTTP/1.1 200 OK
Date: Sun, 10 Nov 2013 11:38:27 GMT
Server: Apache/2.2.11 (Unix) mod_ssl/2.2.11 OpenSSL/0.9.8r
Last-Modified: Mon, 17 Oct 2011 22:21:41 GMT
ETag: "bafb0-a50-4af8607f7c740"
Accept-Ranges: bytes
Content-Length: 2640
Connection: close
Content-Type: text/html
```

...data from the web page follows here

Application Protocol Examples

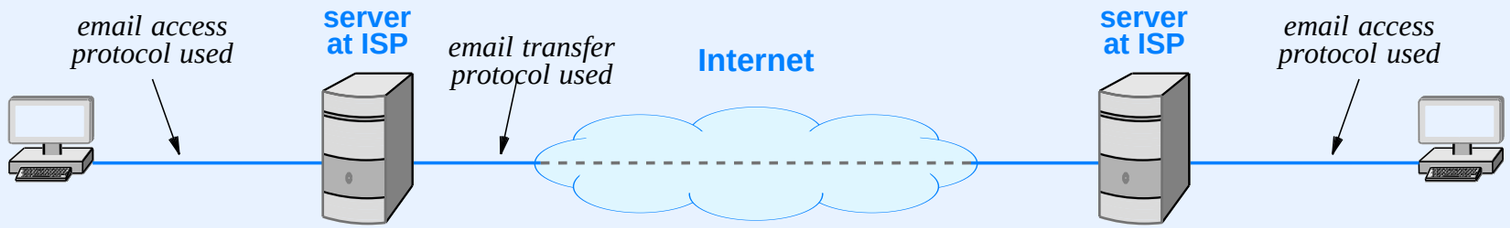
- Web browsing
- Email
- File transfer
- Remote login and remote desktop
- Domain Name System (name lookup)

Original End-To-End Email Paradigm



- Each computer runs
  - Email server to accept incoming email
  - Email client to send outgoing email
- Incoming mail deposited in user's mailbox
- Outgoing mail placed in queue
- User interface to read or compose messages separate from transfer applications

Current Email Paradigm



- User's mailbox located on separate computer (usually at an ISP)
- Mail transfer application deposits message in mailbox
- User interface application accesses remote mailbox
  - A web browser may be used as an access mechanism
  - Special-purpose applications also exist

Simple Mail Transfer Protocol (SMTP)

- Standard for email transfer
- Follows a stream paradigm
- Uses textual control messages
- Only transfers text messages
- Terminates message with *<CR> <LF>* **.** *<CR> <LF>*
- Allows a sender to specify recipients' names and checks each name
- Sends only one copy of a message to a computer, even if destined to multiple recipients on the computer

Example SMTP Session

```
S:  220 somewhere.com Simple Mail Transfer Service Ready
C:  HELO example.edu
S:  250 OK
C:  MAIL FROM:<John_Q_Smith@example.edu>
S:  250 OK
C:  RCPT TO:<Mathew_Doe@somewhere.com>
S:  550 No such user here
C:  RCPT TO:<Paul_Jones@somewhere.com>
S:  250 OK
C:  DATA
S:  354 Start mail input; end with <CR><LF>.<CR><LF>
C:  ...sends body of mail message, which can contain
C:  ...arbitrarily many lines of text
C:  <CR><LF>.<CR><LF>
S:  250 OK
C:  QUIT
S:  221 somewhere.com closing transmission channel
```

Mail Access Protocols

- Two standard protocols
  - Post Office Protocol version 3 (POP3)
  - Internet Mail Access Protocol (IMAP)
- Functionality
  - Provide access to a user's mailbox
  - Permit user to view headers, download, delete, or send individual messages
  - Client runs on user's personal computer
  - Server runs on a computer that stores user's mailbox

RFC2822 Mail Message Format

- Email representation standard
- Name derived from the Internet standard in which it is defined
- Specifies
  - Email message consists of text file
  - Blank line separates *header* from *body*
  - Header lines have the form:

Keyword: information

RFC2822 Mail Message Format
(continued)

- Some keywords have defined meanings:
  - From:
  - To:
  - Subject:
  - Cc:
- Keywords starting with uppercase X have no effect
- Examples:

X-Best-networking-Course: CS422 at Purdue

X-Spam-Check-Results: bulk spam 90% likely

X-Worst-TV-Shows: any reality show

Multimedia Email

- Observe
  - Email was standardized when computers only had character-oriented (textual) interfaces
  - SMTP is limited to transferring plain text messages
  - Users want to email photos, spreadsheets, messages with special fonts and color
- Question: can SMTP be used to transfer such email?
- Answer: it is possible because one can encode arbitrary binary items in plain text (think of a hex dump)

Sending Non-Text Email

- Standard is *MIME* (*Multimedia Internet Mail Extensions*)
- Backward compatible with RFC2822 mail and SMTP
- Sender
  - Encodes arbitrary binary item in plain text
  - Adds lines to email header to specify MIME
  - Places additional headers before each item in the message (including plain text items)
- Sender can specify content type and encoding
- Standard includes *Base64* encoding

Examples Of Mime Headers

- MIME header lines added to other RFC2822 headers

```
MIME-Version: 1.0
Content-Type: Multipart/Mixed; Boundary=xyz123
```

- Each part of the message has a MIME header that starts with the separator and specifies content type and encoding
- Example

```
--xyz123
Content-Type: image/jpeg
```

←*blank line ends header*

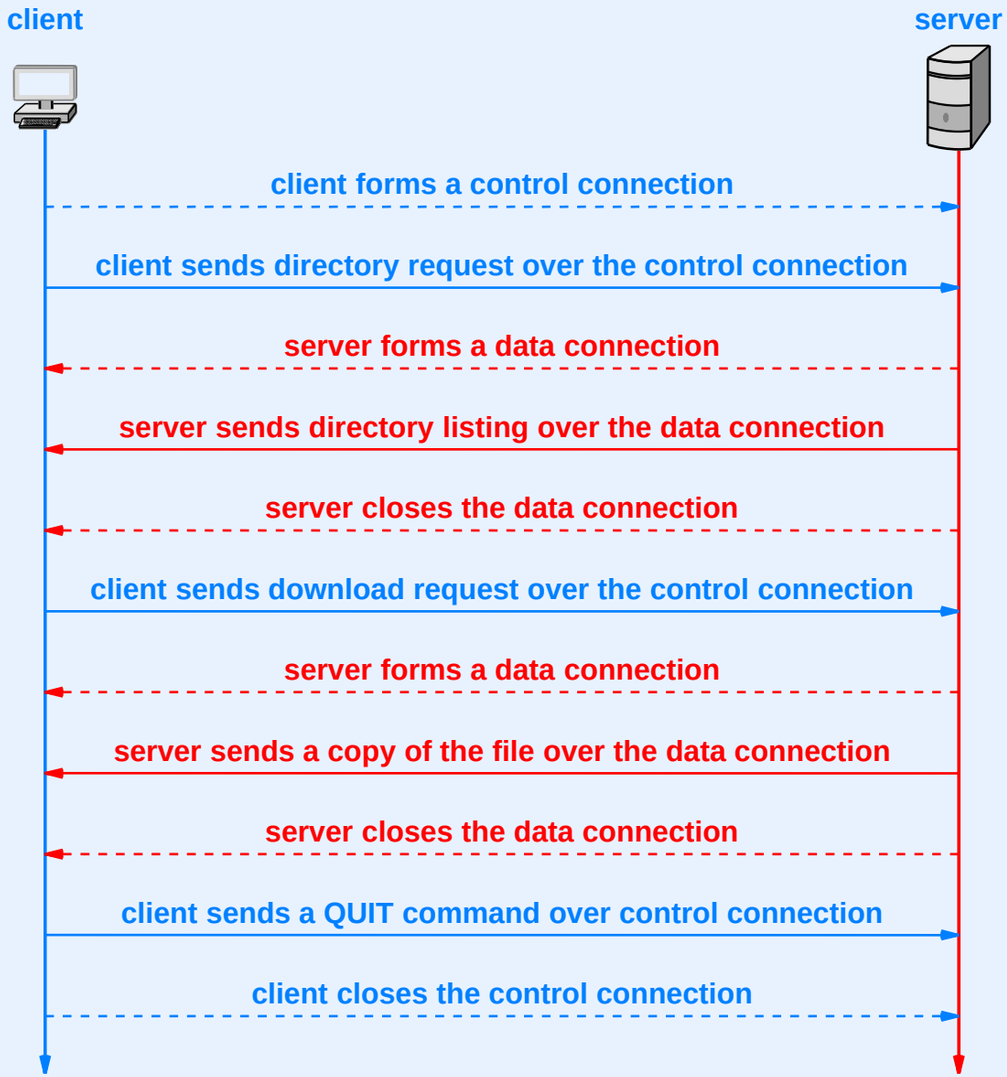
Application Protocol Examples

- Web browsing
- Email
- File transfer
- Remote login and remote desktop
- Domain Name System (name lookup)

File Transfer

- Standard is the *File Transfer Protocol* (*FTP*)
- Once accounted for the most packets on the Internet
- Interesting communication paradigm
  - Client forms a control connection to send requests
  - Server forms data connection for each file transferred
  - Server closes data connection after transfer complete
- Notes
  - Using a separate connection allows arbitrary data transfer
  - For data connections, the server becomes a client and the client becomes a server (important for NAT)

Illustration Of FTP Communication



Application Protocol Examples

- Web browsing
- Email
- File transfer
- Remote login and remote desktop
- Domain Name System (name lookup)

Remote Login And Remote Desktop

- Remote login
  - Intended for systems with command-line interface
  - Internet standard is TELNET
  - Secure shell (ssh) encrypts transfers
  - To appreciate the complexity of application protocols look at the TELNET standard
- Remote desktop
  - Intended for systems that have a Graphical User Interface (GUI)
  - No Internet standards
  - Move to *thin client* has revived interest

Application Protocol Examples

- Web browsing
- Email
- File transfer
- Remote login and remote desktop
- Domain Name System (name lookup)

Domain Name System (DNS)

- Important piece of Internet infrastructure
- Runs at the application layer
- Translates human-readable names into the binary addresses used by the Internet Protocol
- Example
  - Computer www.cs.purdue.edu
  - Has the IP address 128.10.19.20

DNS Terminology

- Names are *hierarchical*
- Each name divided into *segments* by period character, which is read “dot”
- Most significant segment is on the right
- Rightmost segment known as a *top-level domain* (*TLD*)
- Client program known as a *resolver*
  - Used by web browser, email, etc

Top-Level Domains

Domain Name	Assigned To
aero	Air transport industry
arpa	Infrastructure domain
asia	For or about Asia
biz	Businesses
com	Commercial organizations
coop	Cooperative associations
edu	Educational institutions
gov	United States government
info	Information
int	International treaty organizations
jobs	Human resource managers
mil	United States military
mobi	Mobile content providers

Top-Level Domains
(continued)

Domain Name	Assigned To
museum	Museums
name	Individuals
net	Major network support centers
org	Non-commercial organizations
pro	Credentialed professionals
travel	Travel and tourism
xxx	Adult entertainment (porn)
country code	A sovereign nation

- In 2014, ICANN decided to allow many new TLDs

Domain Registration

- Organization
  - Applies under a specific top-level domain
  - Can choose an internal hierarchy
  - Assigns each computer a name
- Geographic registration is possible

  cnri.reston.va.us
- Some countries impose conventions
  - Universities in Great Britain register under

    ac.uk

Domains With Most Hosts (July 2013)

Domain	Hosts	Explanation
net	366592151	Networks
com	163634309	Commercial
jp	74461142	Japan
de	34904481	Germany
br	33691951	Brazil
it	26136473	Italy
cn	19976554	China
mx	17658991	Mexico
fr	17437386	France
au	16900586	Australia
ru	15122103	Russian Federation
nl	14011944	Netherlands
pl	14011944	Poland
ar	13335042	Argentina
edu	12251571	Educational
ca	9004861	Canada
uk	8116718	United Kingdom
in	7429638	India
tr	7146979	Turkey
tw	6429021	Taiwan

- See domain survey at www . isc . org for details

Host Names and Services Offered

- Many organizations choose a host name to match the service a computer offers

mail.foobar.com

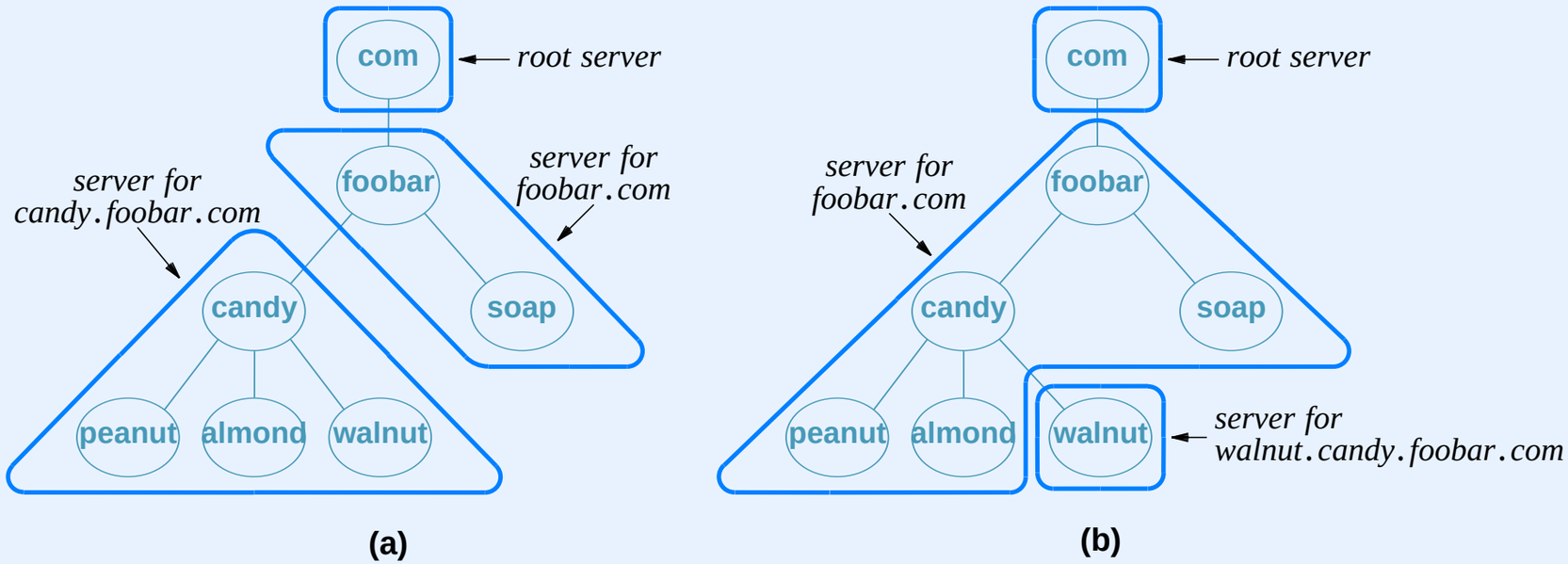
ftp.foobar.com

www.foobar.com

- Although convenient for humans, a host name does not specify which servers are running (e,g., a computer named *mail* could run a web server)

DNS Servers

- Names divided into a hierarchy of servers
- Multiple groupings possible
- Hypothetical example



Name Resolution And Caching

- Resolver
  - Acts as a client
  - Is configured with address of local DNS server
  - Contacts local server first
  - Socket library resolver is *gethostbyname*
- Caching
  - Follows locality of reference principle
  - Each DNS server caches results
  - Cached item never kept when stale

DNS Server Algorithm Part 1

Given:

A request message from a DNS name resolver

Provide:

A response message that contains the address

Method:

```
Extract the name, N, from the request
if  ( server is an authority for N )  {
      Form and send an authoritative response
      to the requester;
else if  ( answer for N is in the cache )  {
      Form and send a nonauthoritative response
      to the requester;
```

DNS Server Algorithm Part 2

```
else  {   /* Need to look up an answer */
     if  ( authority server for N is known )  {
          Send request to authority server;
     }  else  {
          Send request to root server;
     }
     Receive response and place in cache;
     Form and send a response to the requester;
}
```

Summary

- Applications provide all Internet services
- Internet offers connection-oriented stream communication or connectionless message communication
- Most applications follow client-server approach
  - Server starts first and awaits client
  - Client contacts server
- Socket API is a de facto standard
- Application-layer protocol can define
  - Data and message formats (representation)
  - Rules for message exchange (transfer)

Summary
(continued)

- Applications reviewed include
  - Web (URL, HTML, HTTP)
  - Email (SMTP, RFC2822, MIME)
  - File transfer (FTP)
  - Remote login and remote desktop (TELNET)
  - Domain Name System (DNS)