LECTURE 12: PYTHON BYTES, TCP/IP (LAB 08)

Computer Systems and Networks

Dr. Pallipuram
(vpallipuramkrishnamani@pacific.edu)

University of the Pacific

Today's Agenda

Python exercises to simulate network communication:
- we'll simulate construction of a client's request for a server
- we'll simulate how the client 'decodes' the server's response
- TCP/IP networking
- If you followed the lecture well, you may finish lab08 today!

Discuss what happens under the hood when you...

type www.google.com/images/srpr/logo3w.png in the URL tab of Mozilla

Python Bytes

Python `bytes()` returns a byte sequence of the passed parameter:

```
byteobject = bytes(parameter, <encoding type>)
```

To decode back from bytes:
```
byteobject.decode(<decoding type>)
```

Useful in networking where messages are sent as bytes

Exercise 1a: Warm-up with Python from Last assignment

Open the Python interpreter, initialize a variable called `url` to http://www.google.com/images/srpr/logo3w.png. Extract `hostname` and `filepath` using Python string operations.

```
filepath should be /images/srpr/logo3w.png
hostname should be www.google.com

#create a string, request that should print like this:
>>> print(request)
Host: www.google.com
File: images/srpr/logo3w.png
Connection: close
```

\r\n\r\n

```
>>>
```

Exercise 1b: Convert the request to bytes

```
>>> send_request = bytes(request,'ascii') #request
to be send to a server in bytes, encoded ascii

>>> print(send_request) #what happens?



#decode it back
>>>decode_req=send_request.decode('ascii')
>>>print(decode_req)
```

Exercise 2: Create a string, `response`, that prints as shown. Create its byte version, `byte_response`

Create a string, `response`, that prints as follows:

```
>>> print(response)
HTTP accepts the command (code 202)
this line is message header

this line is data
```

header

\r\n\r\n

data

Exercise 3: Extract `header` and `data` from `byte_response`

After splitting, header and data should print like this:

```
>>> header
HTTP accepts the command (code 202)\n this line is
messsage header
>>> data
this line is data
>>>
```

Network Model

Application Layer
(Myriad examples: Web browser, web server, etc...)

Transport Layer
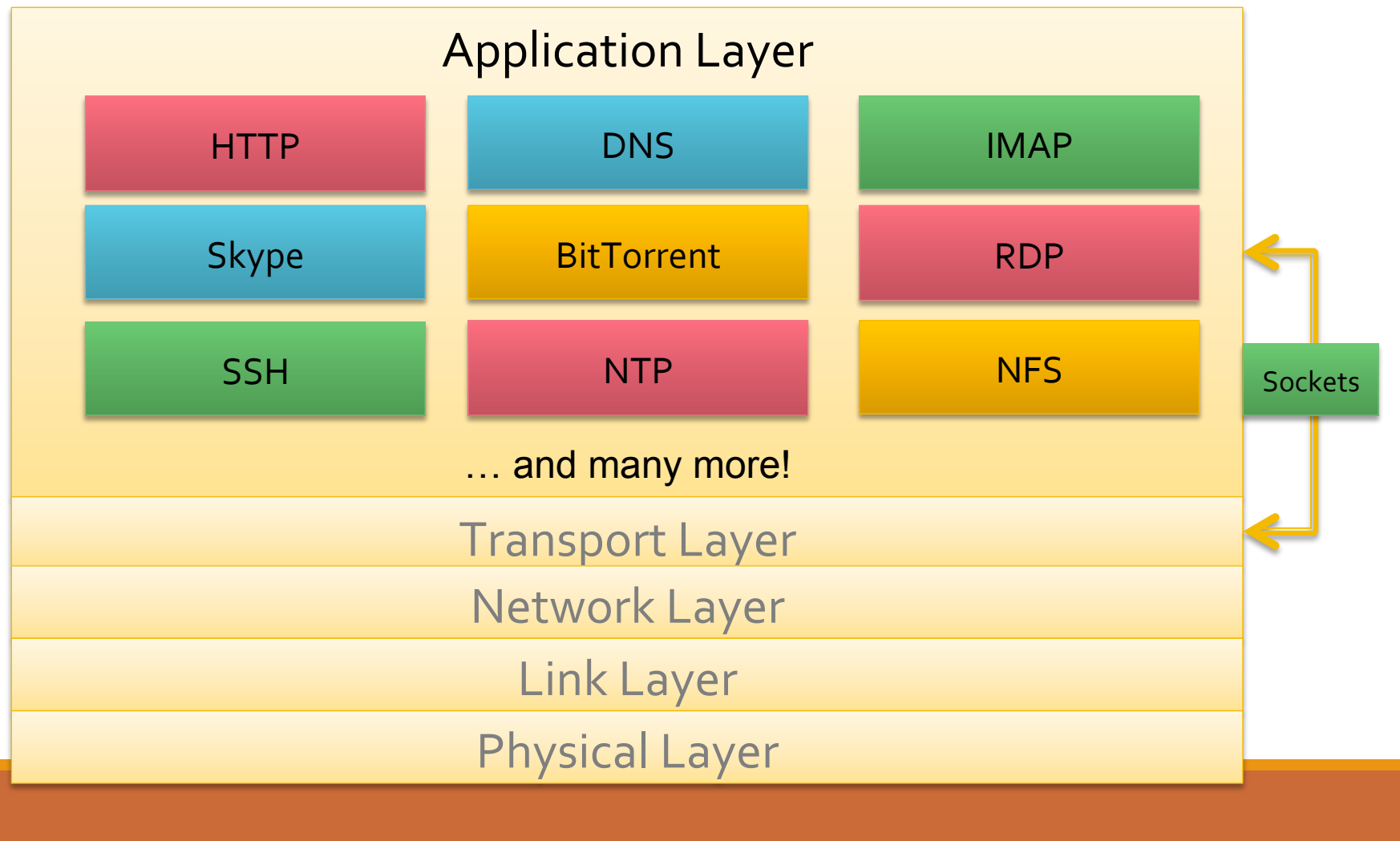(Reliability – e.g. TCP)

Network Layer
(Global Network – e.g. IP)

Link Layer
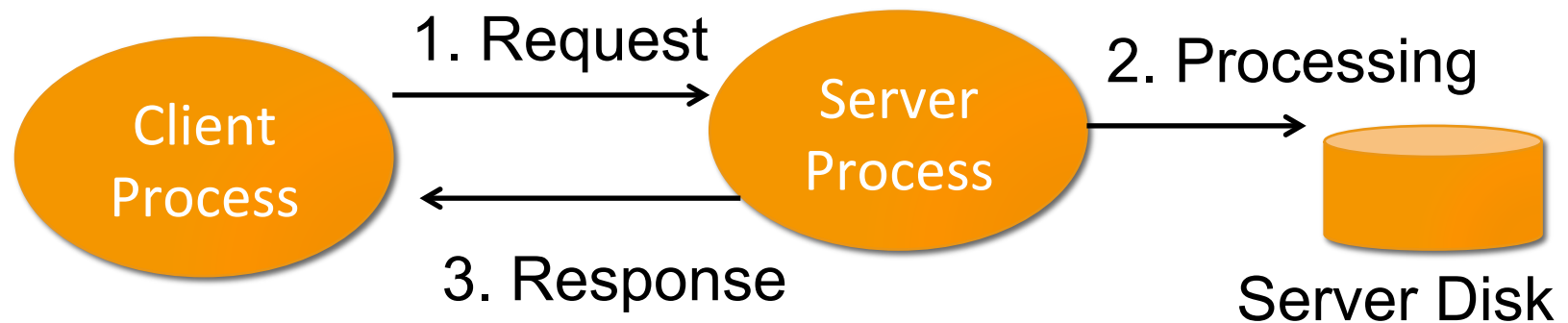(Local Area Network – e.g. Ethernet)

Physical Layer
("Bit on a Wire")

Application Layer

Client-Server Architecture and TCP/IP

Internet Protocol (IP): provides naming scheme and (unreliable) mechanism to transmit packets called *datagrams*

Transmission Control Protocol (TCP) provides mechanism for reliable bi-directional connection

Client and server communicate via Sockets
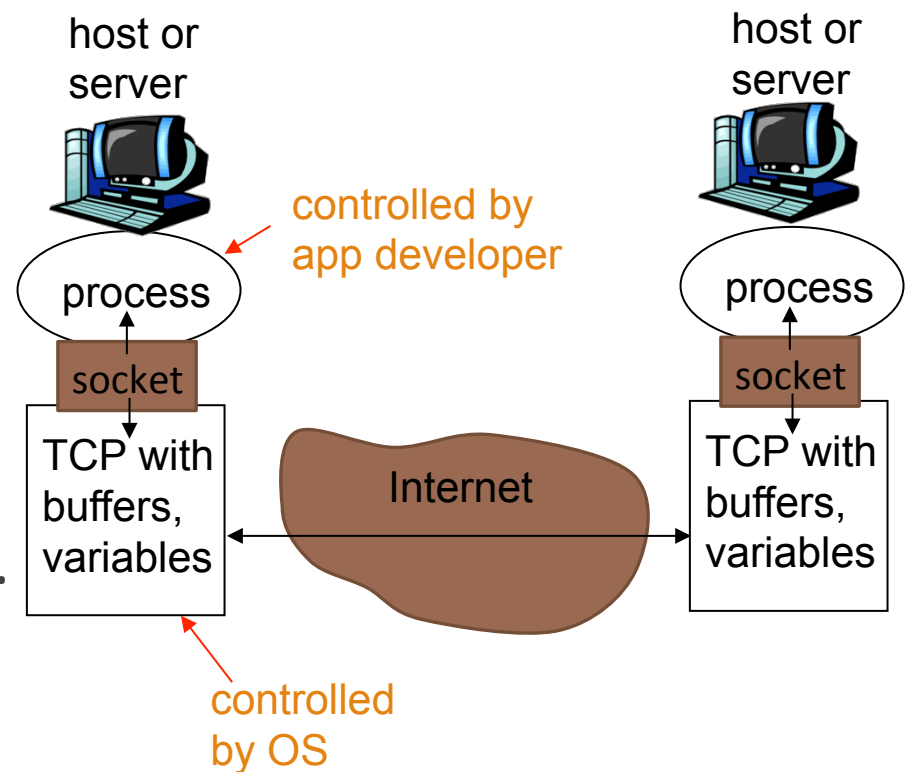
What is a Socket?

Process sends/receives messages to/from its socket

Socket analogous to door
- Sending process shoves message out door
- Transport infrastructure on other side of door carries message to socket at receiving process
- **Imagine you are just writing to a file...**

API allow customization of socket
- Choose transport protocol
- Choose parameters of protocol

host or server

host or server

controlled by app developer

process

process

socket

socket

TCP with buffers, variables

Internet

TCP with buffers, variables

controlled by OS

Addressing Processes

To receive messages, each process on a host must have an **identifier**

◦ IP addresses are unique

◦ **Is this sufficient?**

No, there can thousands of processes running on a single machine (with one IP address)

Identifier must include

◦ IP address

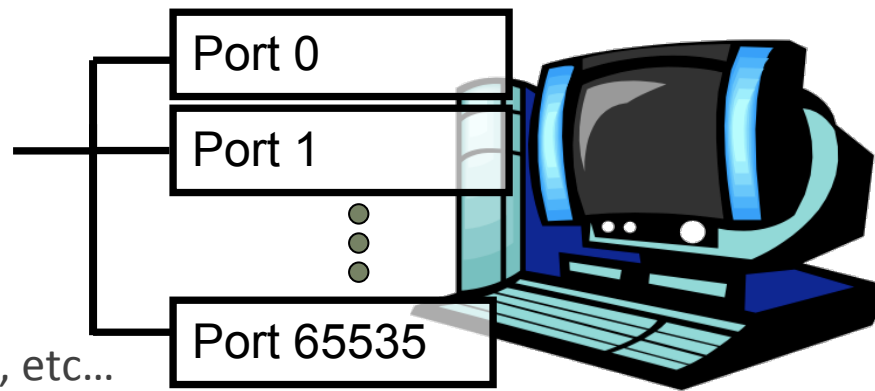◦ **and** port number (example: 80 for web)

Ports

Each host has
65,536 ports

Some ports are
*reserved for
specific apps*
◦ FTP (20, 21), Telnet (23), HTTP (80), etc...

Outgoing ports (on clients) can be dynamically assigned by OS in upper region (above 49,152) – called **ephemeral ports**

See http://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers

Port 0

Port 1

Port 65535

Lab 08 Client Socket Usage: Python functions

Basic socket functions for **connection-oriented (TCP)** <u>clients</u>

1. `socket()`   create the socket descriptor

2. `connect((host,port))`   connect to the remote server

3. `send(),sendall(),recv()` communicate with the server

4. `close()`   end communication by closing
   socket descriptor

Google search and gather information for these functions to perform TCP/IP
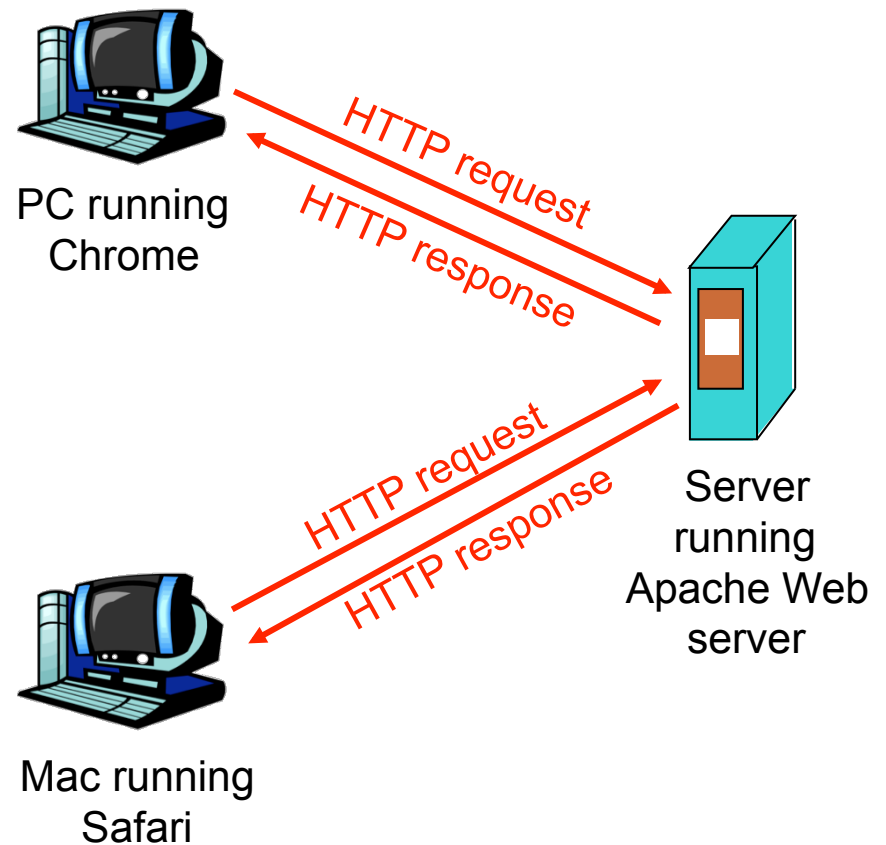
Lab 08 -- Hypertext Transfer Protocol Overview

HTTP is the *application layer protocol* for the web

It is how the client and server communicate

Client/server model
◦ **Client**: browser that requests, receives, "displays" Web objects
◦ **Server**: Web server sends objects in response to requests

PC running Chrome

HTTP request

HTTP response

Server running Apache Web server

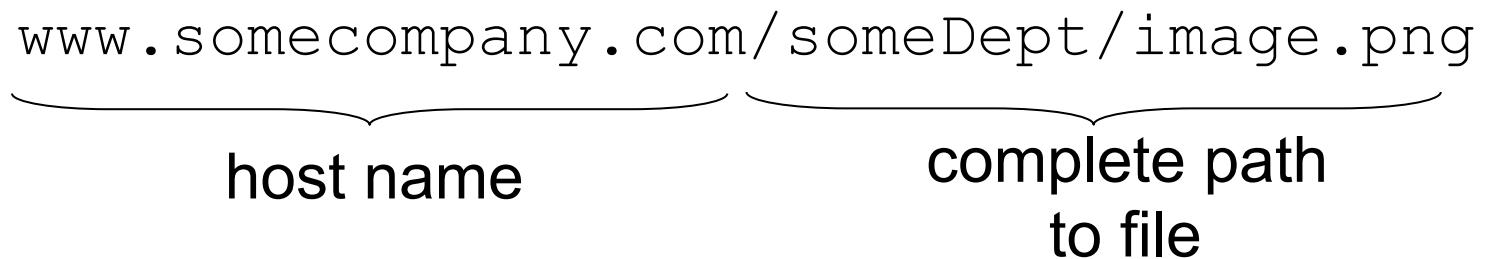HTTP request

HTTP response

Mac running Safari

Web and HTTP

Web **page** consists of base HTML file and (potentially) many referenced **objects**

◦ HTML file, PNG image, Flash video, …

Each object is addressable by a **URL**

Example URL:

```
www.somecompany.com/someDept/image.png
```

host name   complete path
to file

HTTP Response Status 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:)

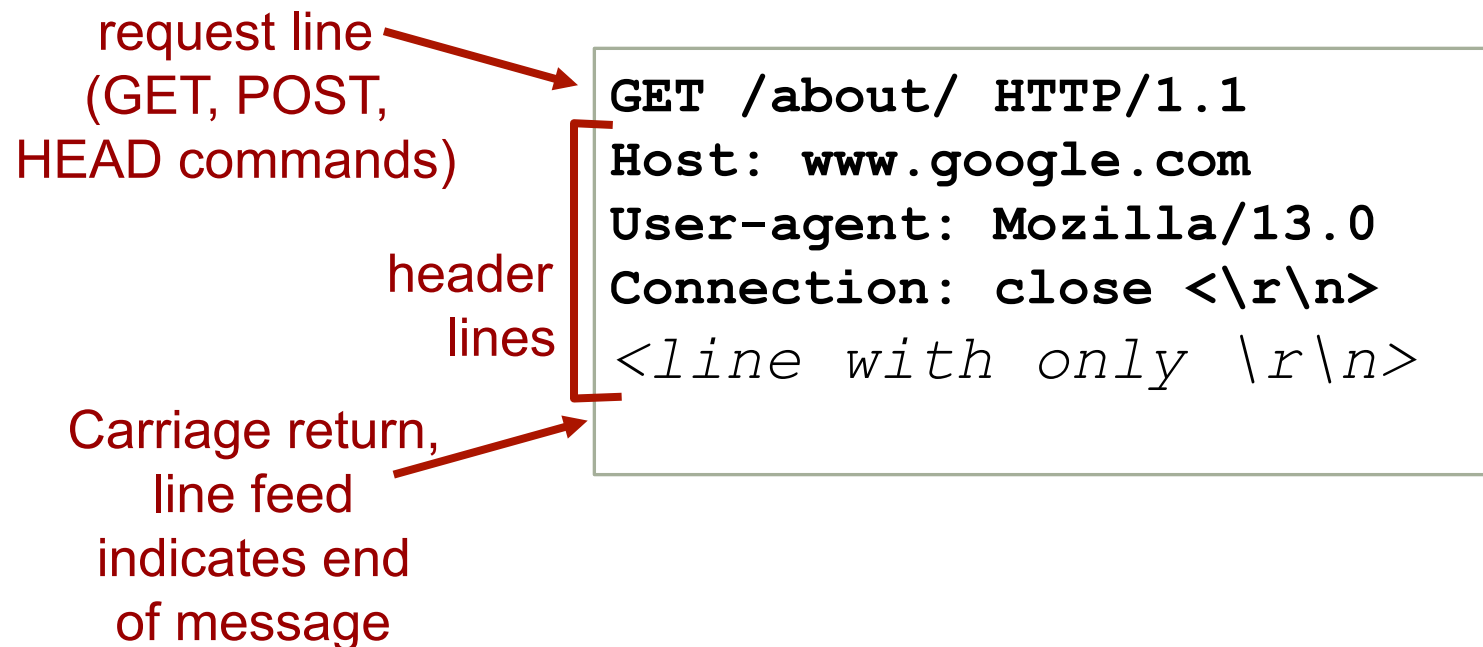**400 Bad Request**
- Request message not understood by server

404 Not Found
- Requested document not found on this server

505 HTTP Version Not Supported

A few examples out of many!

HTTP Request Message (Client->Server)

request line
(GET, POST,
HEAD commands)

header
lines

Carriage return,
line feed
indicates end
of message

```
GET /about/ HTTP/1.1
Host: www.google.com
User-agent: Mozilla/13.0
Connection: close <\r\n>
<line with only \r\n>
```

HTTP is a text-based protocol. The client sends ASCII bytes in the request, and the server responds with ASCII bytes in the reply.

Lab 8 Tasks: display.py (Remember Exercises 1a 1b)

Student Work #1: Build the HTTP request

Append info to a `request` string to construct HTTP request:

Parsed by your boilerplate code!

```
request=     GET <filename> HTTP/1.1
             Host: <host>
             Connection: close <\r\n>
             <line with only \r\n>
```

Lab 8 Tasks: display.py

Student Work #2,3,4: Connect to the server and send the entire request

2. Establish the socket. (Which socket function?)
3. Connect to the host. (Which socket function?)
4. Send the request string as bytes. (Which set of functions?)

HTTP Response Message (Server -> Client) (Remember Exercise 3)

status line
(protocol
status code,
status phrase)

header
lines

data, e.g.,
requested
HTML file

```
HTTP/1.1 200 OK
Vary: Accept-Encoding
Content-Type: text/html
Last-Modified: Tue, 10 Apr 2012 09:33:47
Date: Tue, 10 Apr 2012 17:50:51 GMT
Expires: Tue, 10 Apr 2012 17:50:51 GMT
Cache-Control: private, max-age=0
X-Content-Type-Options: nosniff
Server: sffe
X-XSS-Protection: 1; mode=block
Transfer-Encoding: chunked
<line with only \r\n>
<Data begins here...>
```

Lab 8 Tasks: display.py

Student Work #5: Continue to receive chunks of 64 KB until you receive no more. You need to set up a loop.

Which socket function to receive data (in 64 KB chunks) from server?

How do you continue to receive data until you receive 0 bytes?

How do you separate the header and data?

Lab 8 Task: Close the connection

Student Work#6:
Which python function to close the connection?

Reading Assignment: Read Big-Endian and Little-Endian Format

Assignment: Complete the WireShark tutorial on Lab09 page