



Computer Systems and Networks

ECPE 170 – Jeff Shafer – University of the Pacific

Networking Fundamentals

Lab Schedule

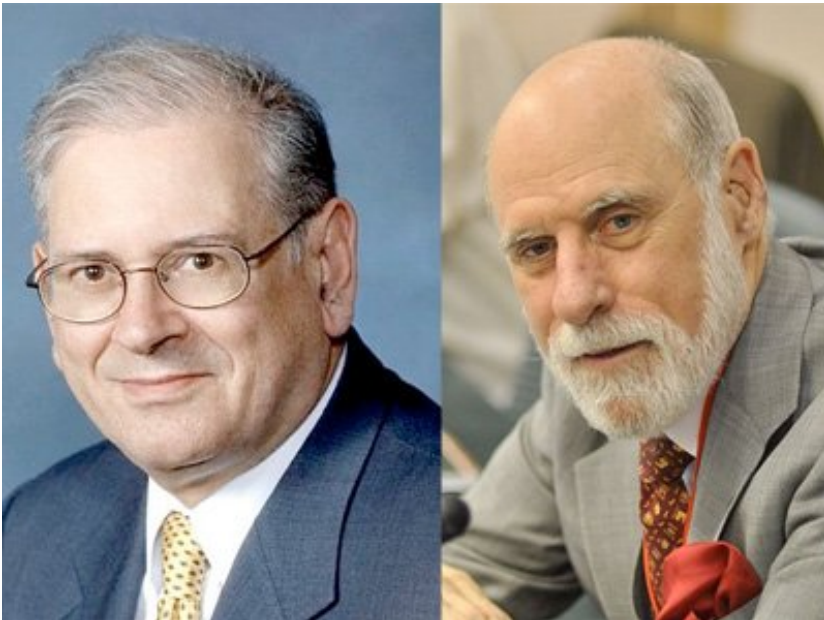
Activities

- **Today**
 - Network Programming
 - Python
- **Next Week**
 - **Lab 12 – Network Programming**

Assignments Due

- **Lab 11**
 - **Due by Dec 1st 5:00am**
- **Lab 12**
 - **Due by Dec 8th 5:00am**
- **Final Exam**
 - **Tue Dec 9th 8-10am**

Persons of the Day: Vint Cerf / Bob Kahn



- Co-designers of TCP/IP protocol suite
 - Enables reliable communication across unreliable network
 - **Foundation of Internet**
- 2004 *ACM Turing Award* winners (shared)
- 2005 *Presidential Medal of Freedom* winners (shared)

Person of the Day: Tim Berners-Lee



- Inventor of “World Wide Web”
- First implementation of **HTTP** (HyperText Transfer Protocol) to communicate between client and server
- Knighted by Queen Elizabeth II in 2004

Computer Networks



Disclaimer

- **These topics take an entire semester of COMP 177 (Computer Networking) to explore!**
- A few days (*most of which is lab time*) is only sufficient for the briefest of overviews...

Network Model

Application Layer

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

Transport Layer

(Reliability – e.g. TCP)

Network Layer

(Global Network – e.g. IP)

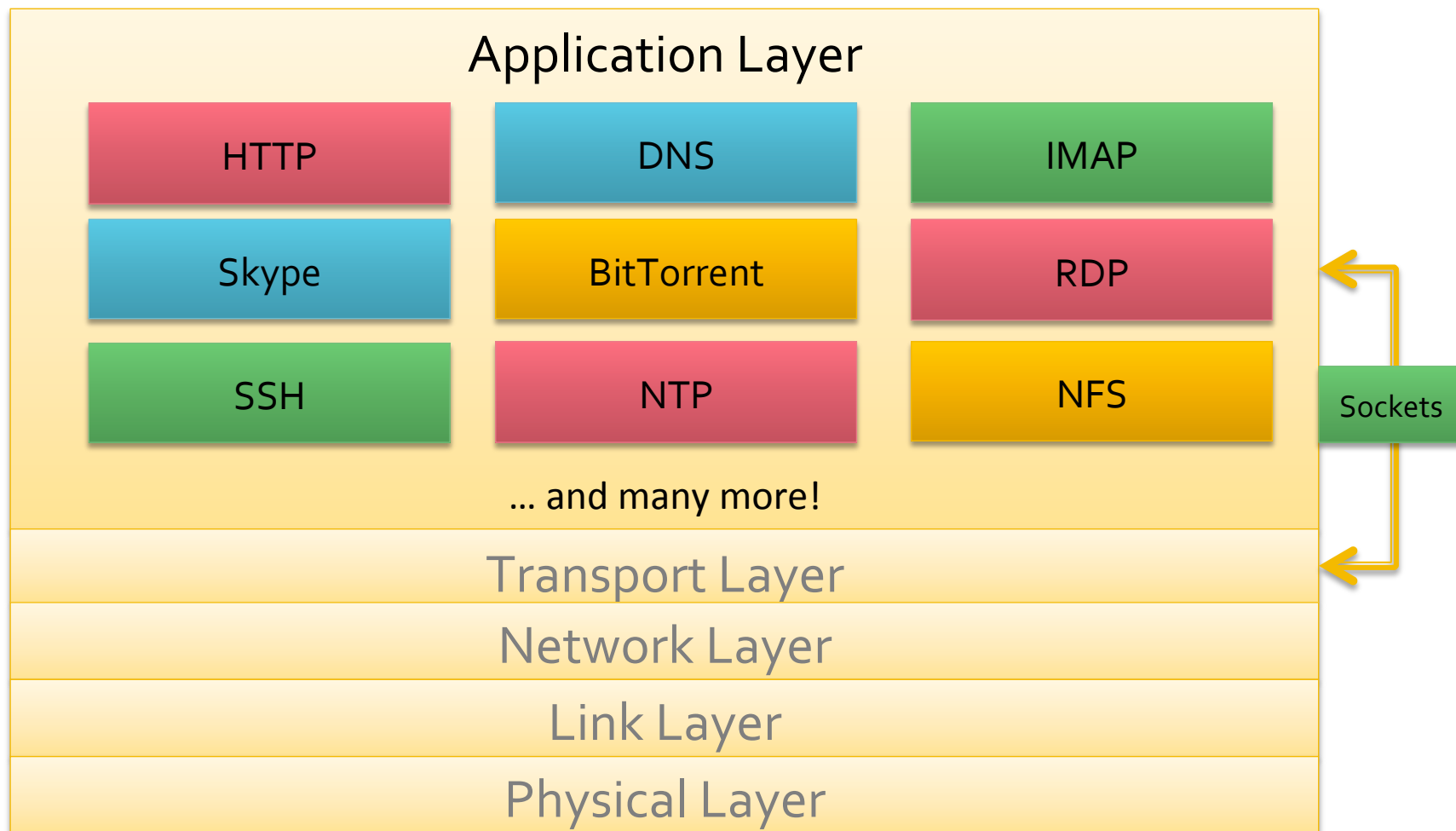
Link Layer

(Local Area Network – e.g. Ethernet)

Physical Layer

(“Bit on a Wire”)

Application Layer



Application Layer

- The **application layer** programmer can make many (fantastic) assumptions about the network
 - The network is reliable
 - Messages are not lost
 - Messages are received in the order they are sent
 - The network can transfer data of infinite length (you can send as much data as desired)
 - You can deliver messages directly to a specific application on a specific computer anywhere on the planet

- The lower layers (transport, network, link, ...) do all the heavy-lifting to make these assumptions true

Client-Server Architecture

Server

- Always-on host
- Always has a known IP address
- Lots of bandwidth
- **Server process:** process that waits to be contacted

Client

- Communicate with server
- May be intermittently connected
- May have dynamic IP addresses
- Do not communicate directly with each other
- **Client process:** process that initiates communication

Why Do We Have Sockets?

- Challenge – **Inter-process communication**
- A **process** is an independent program running on a host
 - Separate memory space
- How do processes communicate with other processes
 - On the same host?
 - On different hosts?
- Send **messages** between each other

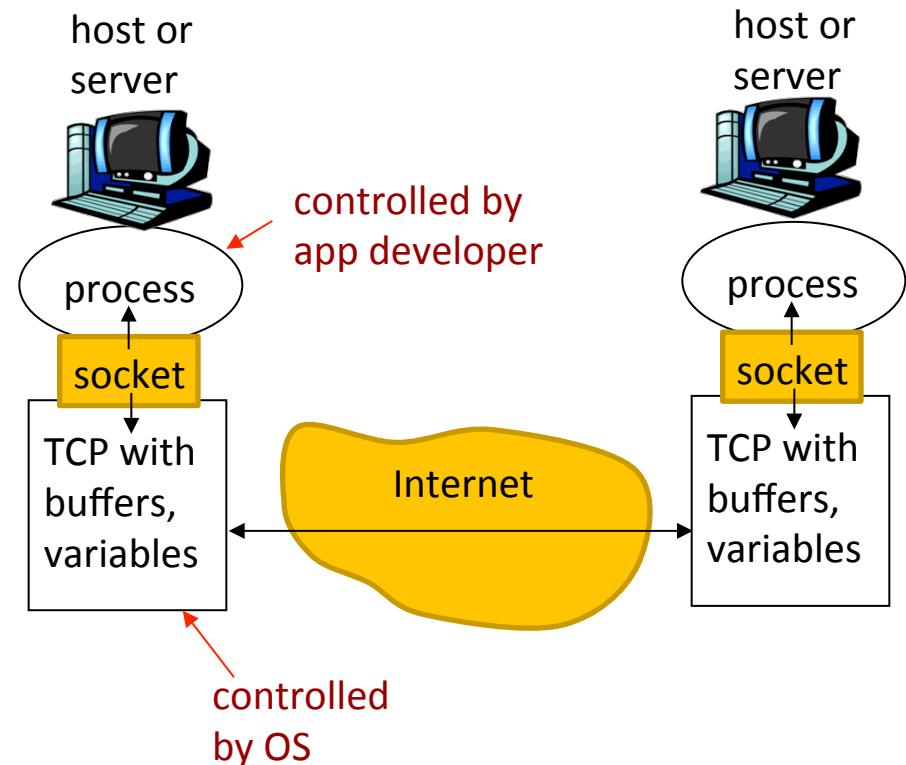
What is a Socket?

- An interface between process (application) and network
 - The application creates a socket
 - The socket *type* dictates the style of communication
 - Reliable vs. best effort
 - Connection-oriented vs. connectionless

- Once configured the application can
 - Pass data to the socket for network transmission
 - Receive data from the socket (transmitted through the network by some other host)

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

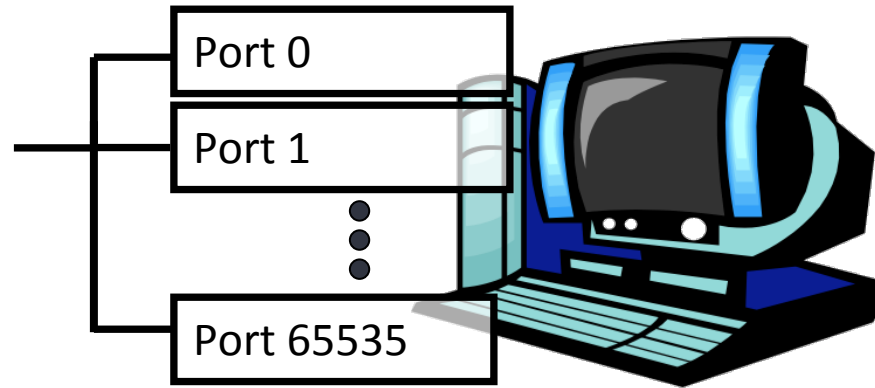


Addressing Processes

- 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



Socket Usage: Client Program

- Basic socket functions for **connection-oriented (TCP) clients**
- 1. **socket ()** create the socket descriptor
- 2. **connect ()** connect to the remote server
- 3. **send () , recv ()** communicate with the server
- 4. **close ()** end communication by closing socket descriptor

Application-Layer Protocol

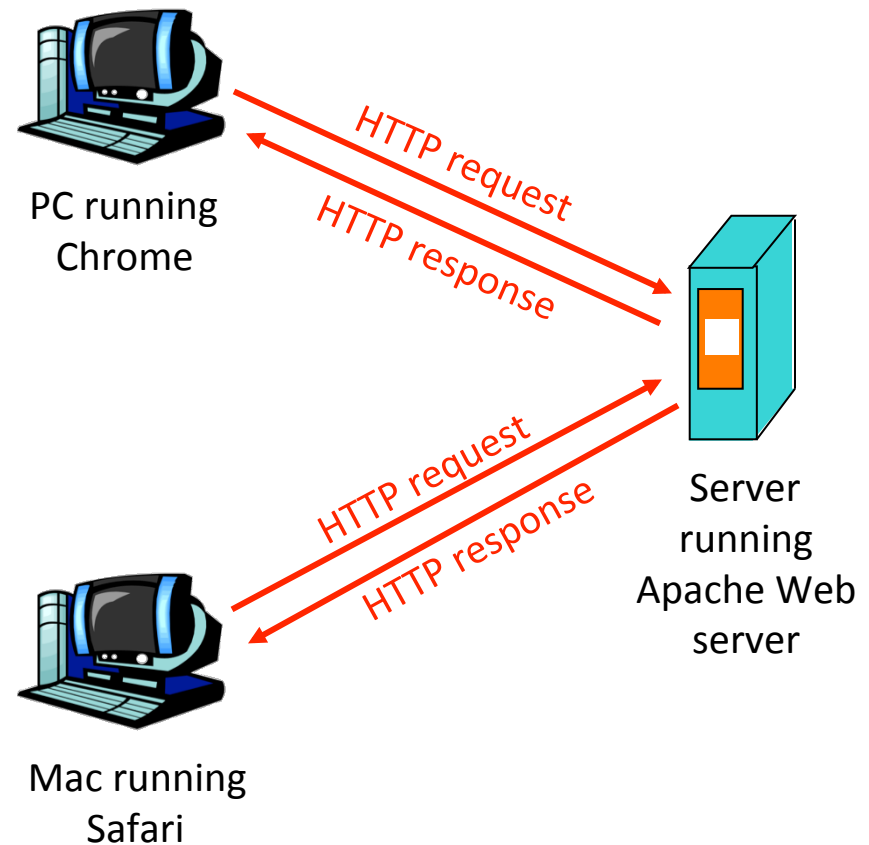
- Sockets just allow us to send raw messages between processes on different hosts
 - Transport service takes care of moving the data
- **What** exactly is sent is up to the application
 - An **application-layer** protocol
 - HTTP, NTP, IMAP, SFTP, Skype, etc...

Application-Layer Protocol

- Both the client and server speaking the protocol must agree on
 - **Types of messages exchanged**
 - e.g., request, response
 - **Message syntax**
 - What fields are in messages
 - How fields are delineated
 - **Message semantics**
 - Meaning of information in fields
 - Rules for **when** and **how** processes send and respond to messages

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



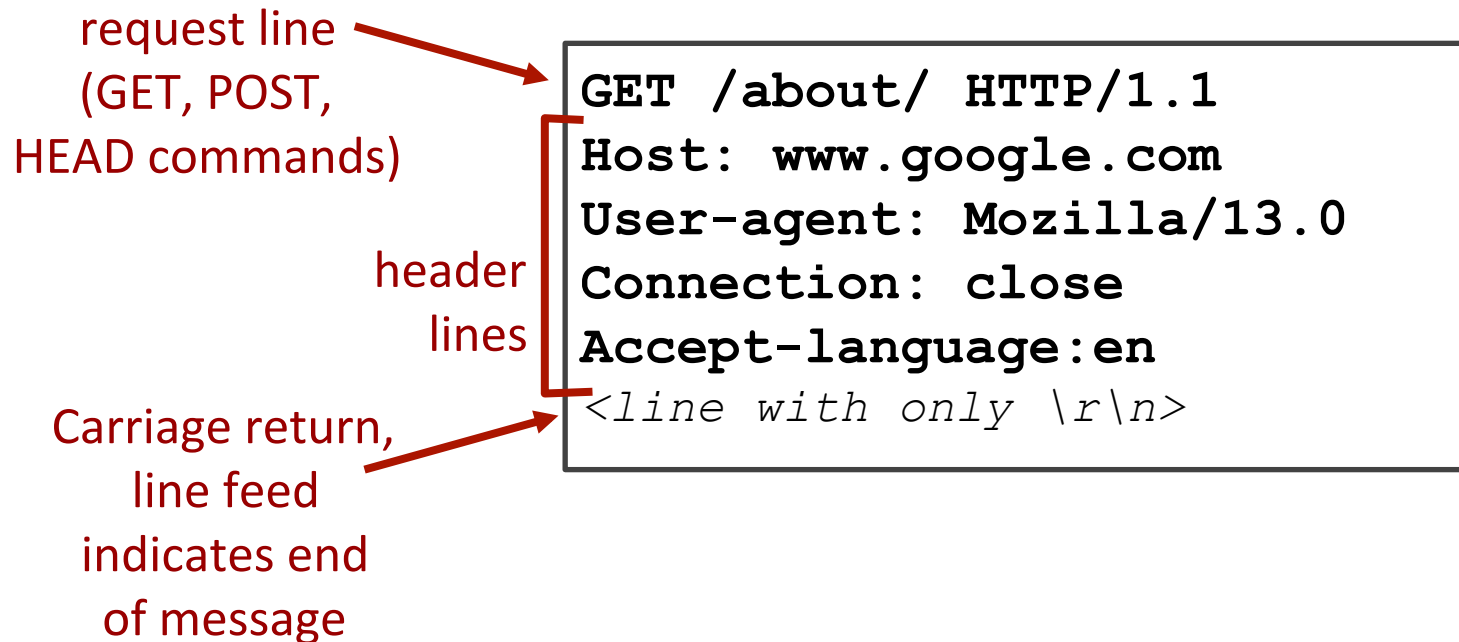
Web and HTTP

- 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 path name

HTTP Request Message (Client->Server)



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

HTTP Response Message (Server -> Client)

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...>
```

HTTP Response Status Codes

*A few
examples
out of
many!*

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

HTTP

➤ Telnet example – impersonate a web browser!

Request:

```
unix> telnet www.google.com 80
```

```
-----  
GET /about/ HTTP/1.1
```

```
Host: www.google.com
```

```
Connection: close
```



**What do we need at the
end of our request?**

Response:

```
HTTP/1.1 200 OK
```

```
Vary: Accept-Encoding
```

```
Content-Type: text/html
```

```
Last-Modified: Tue, 10 Apr 2012 09:33:47 GMT
```

```
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
```

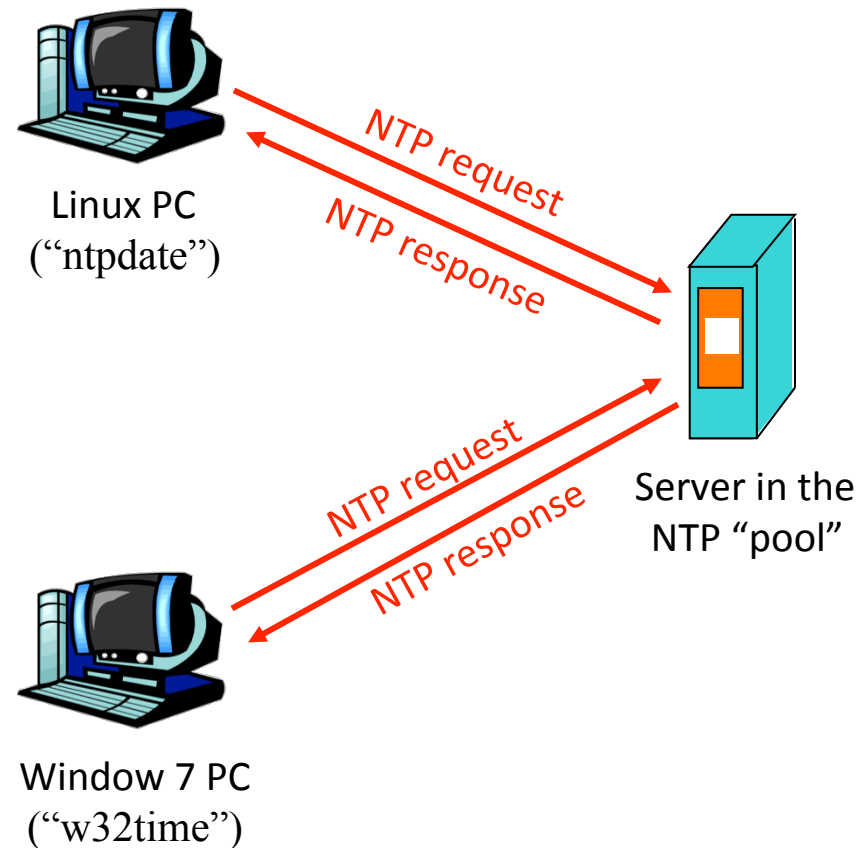
```
X-XSS-Protection: 1; mode=block
```

```
Transfer-Encoding: chunked
```

```
<file>
```


Network Time Protocol Overview

- **NTP** is the *application layer protocol* for syncing computer's clock
- It is how the client and server communicate
- Client/server model
 - **Client:** requests "current time"
 - **Server:** server sends time responses



Clock Strata

- ➔ NTP servers are arranged in a hierarchy called **stratums**
1. Synchronized directly to atomic, GPS or radio clocks
 2. Synchronized to one or more stratum 1 sources
 3. Synchronized to one or more stratum 2 sources
 4.
- ➔ This layering allows the workload to be distributed

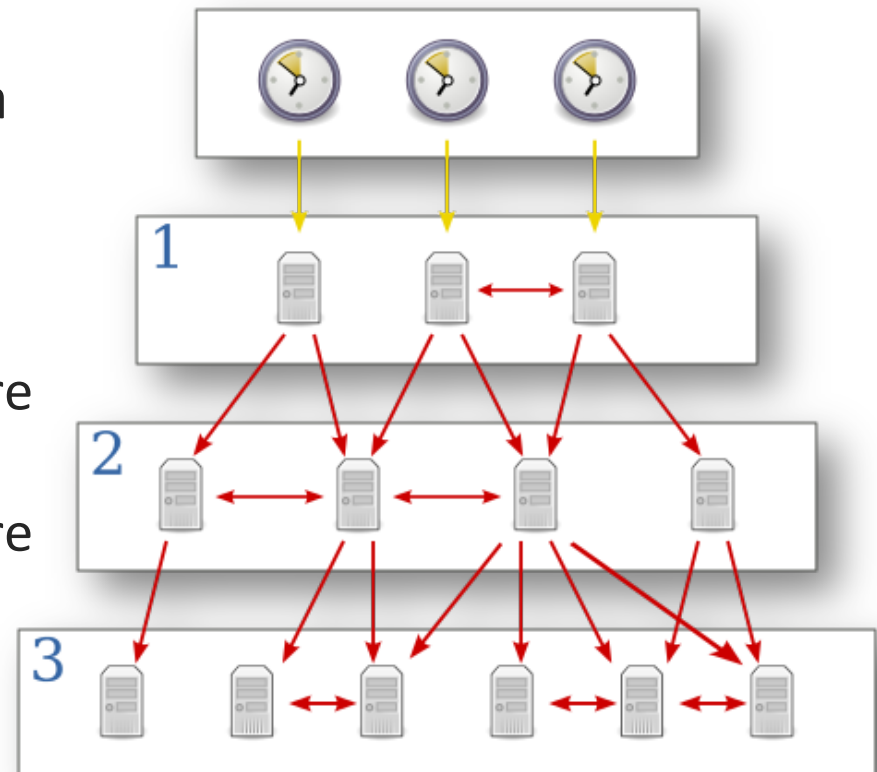


Image by Ben D. Esham, from Wikipedia

NTP Request Message (Client->Server)

Data packet
format is the
same for clients
and servers

Header (version, stratum, etc)
Reference time
Origin Time Nov 21, 2013 08:32:09.1982
Receive time
Transmit time

Client sends
request with
the current
local time

NTP Request Message (Server -> Client)

It also updates the header with the stratum and IP address

Header (version, stratum , etc)	
Reference time	
Origin Time	Nov 21, 2013 08:32:09.1982
Receive time	Nov 21, 2013 08:32:12.7563
Transmit time	Nov 21, 2013 08:32:14.3071

Server records
the time when
the packet was
received...

and the time
when it sends
its reply

NTP Request Message (Client result)

The client notes the time when it receives the reply... then calculates the differences between the transmit and receive delays.

Local time is 1.1904 sec fast!

→ Nov 21, 2013 08:32:15.4853

Header (version, stratum, etc)	
Reference time	
Origin Time	Nov 21, 2013 08:32:09.1982
Receive time	Nov 21, 2013 08:32:12.7563
Transmit time	Nov 21, 2013 08:32:14.3071

- = 1.1872 sec

- = 3.5581 sec

Assuming each delay *should* be identical, it calculates the difference between the local time and the “true” time.

ntpdate example

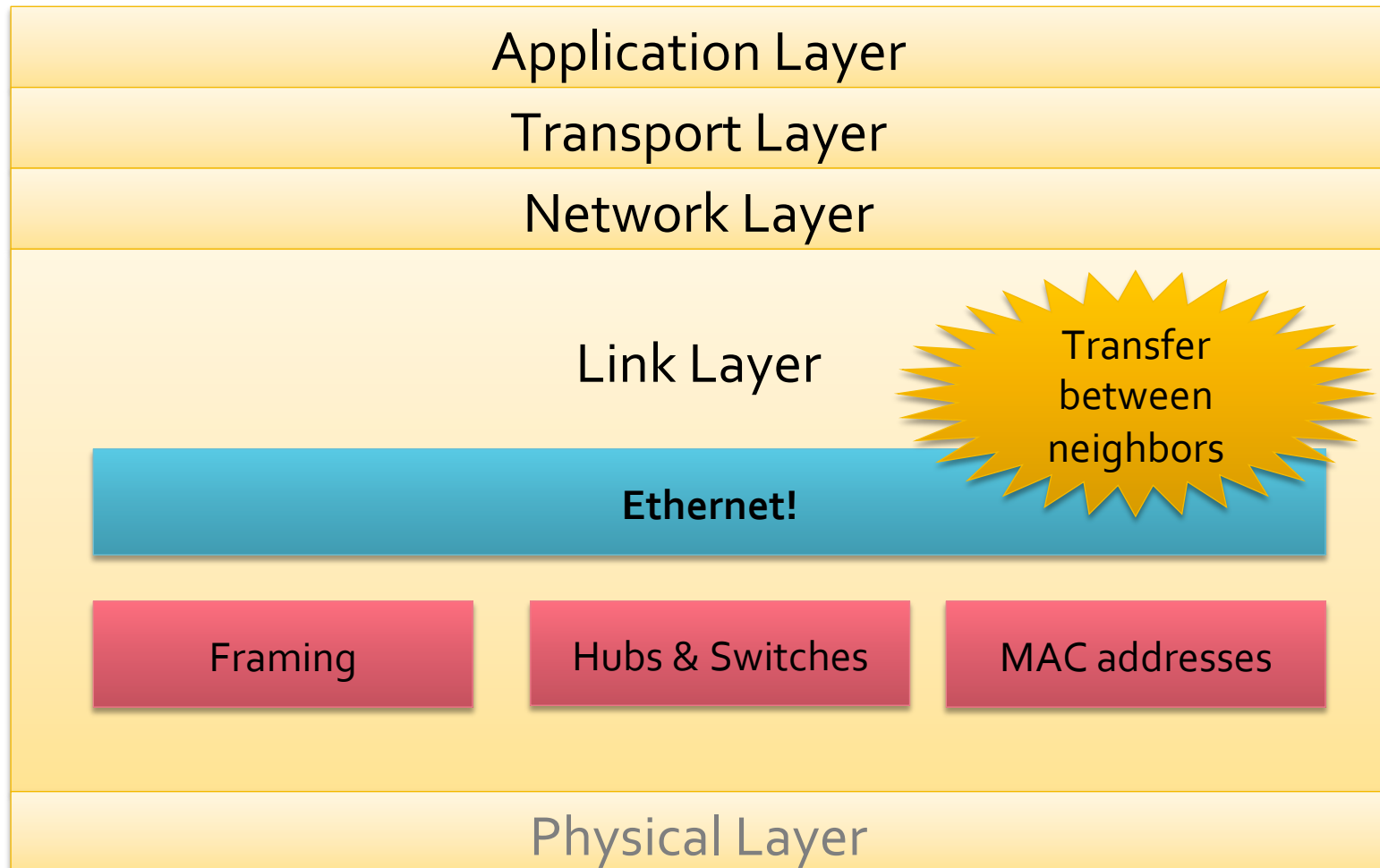
```
unix> sudo ntpdate -v -d 0.pool.ntp.org
Looking for host 0.pool.ntp.org and service ntp
host found : falcon.ca.us.slacked.org
transmit(173.230.158.30)
receive(173.230.158.30)
. . . .
stratum 2, precision -21, leap 00, trust 000
refid [108.61.56.35], delay 0.11089, dispersion 0.00169
transmitted 4, in filter 4
reference time:      d63643a8.54a20764  Tue, Nov 19 2013 12:08:08.330
originate timestamp: d63647e5.6f8ed6de  Tue, Nov 19 2013 12:26:13.435
transmit timestamp:  d63647e5.670059ac  Tue, Nov 19 2013 12:26:13.402
filter delay:   0.11162  0.11189  0.11134  0.11089
                0.00000  0.00000  0.00000  0.00000
filter offset: -0.00647 -0.00742 -0.00764 -0.00923
                0.000000 0.000000 0.000000 0.000000
delay 0.11089, dispersion 0.00169
offset -0.009232

19 Nov 12:23:52 ntpdate[1001]: adjust time server 198.60.22.240 offset 0.062573 sec
unix>
```

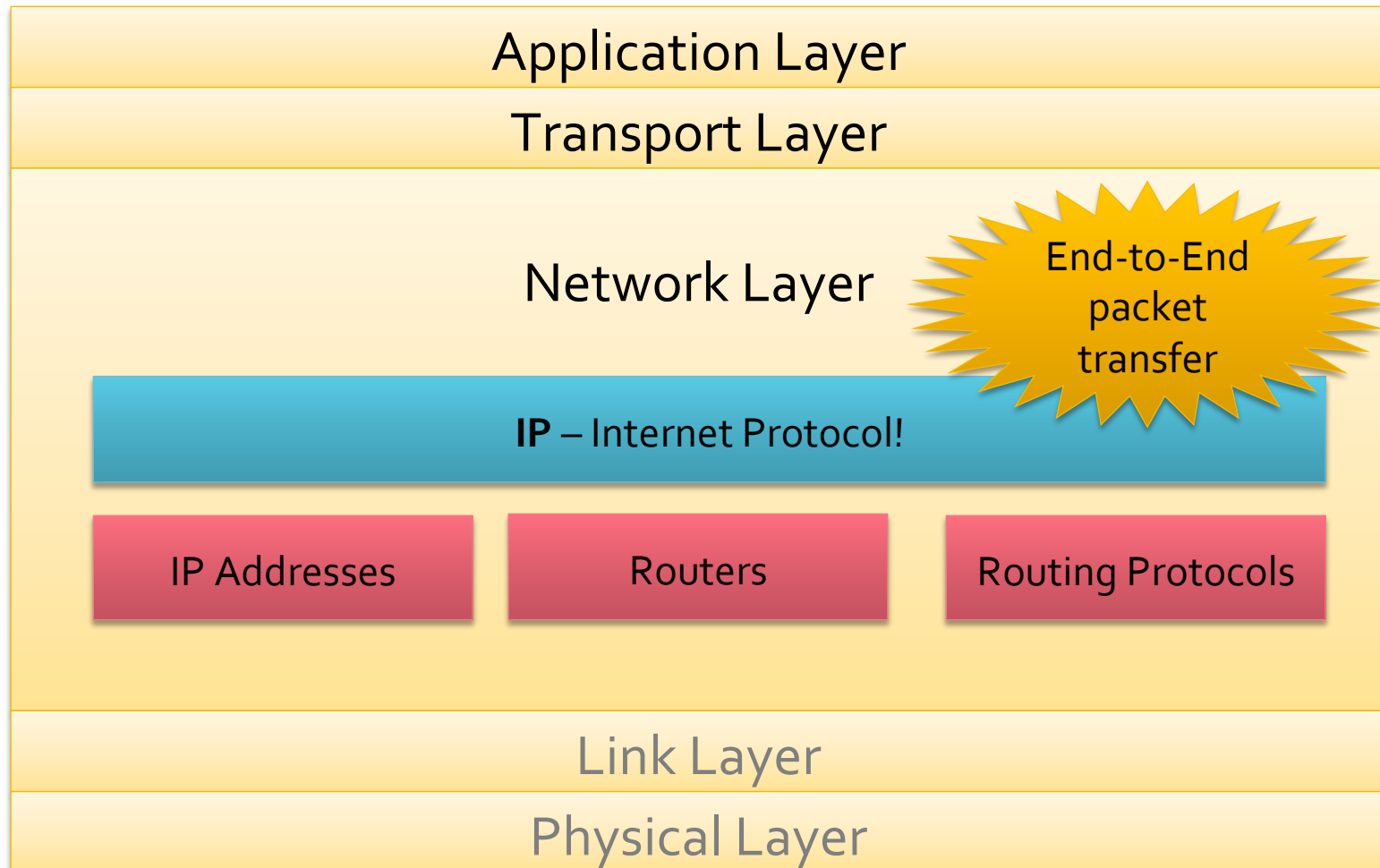
Other Layers



Link Layer



Network Layer



IP Properties

➤ **Datagram**

- Each packet is **individually routed**
- Packets may be **fragmented** or **duplicated** by underlying networks

➤ **Connectionless**

- No guarantee of delivery in sequence

➤ **Unreliable**

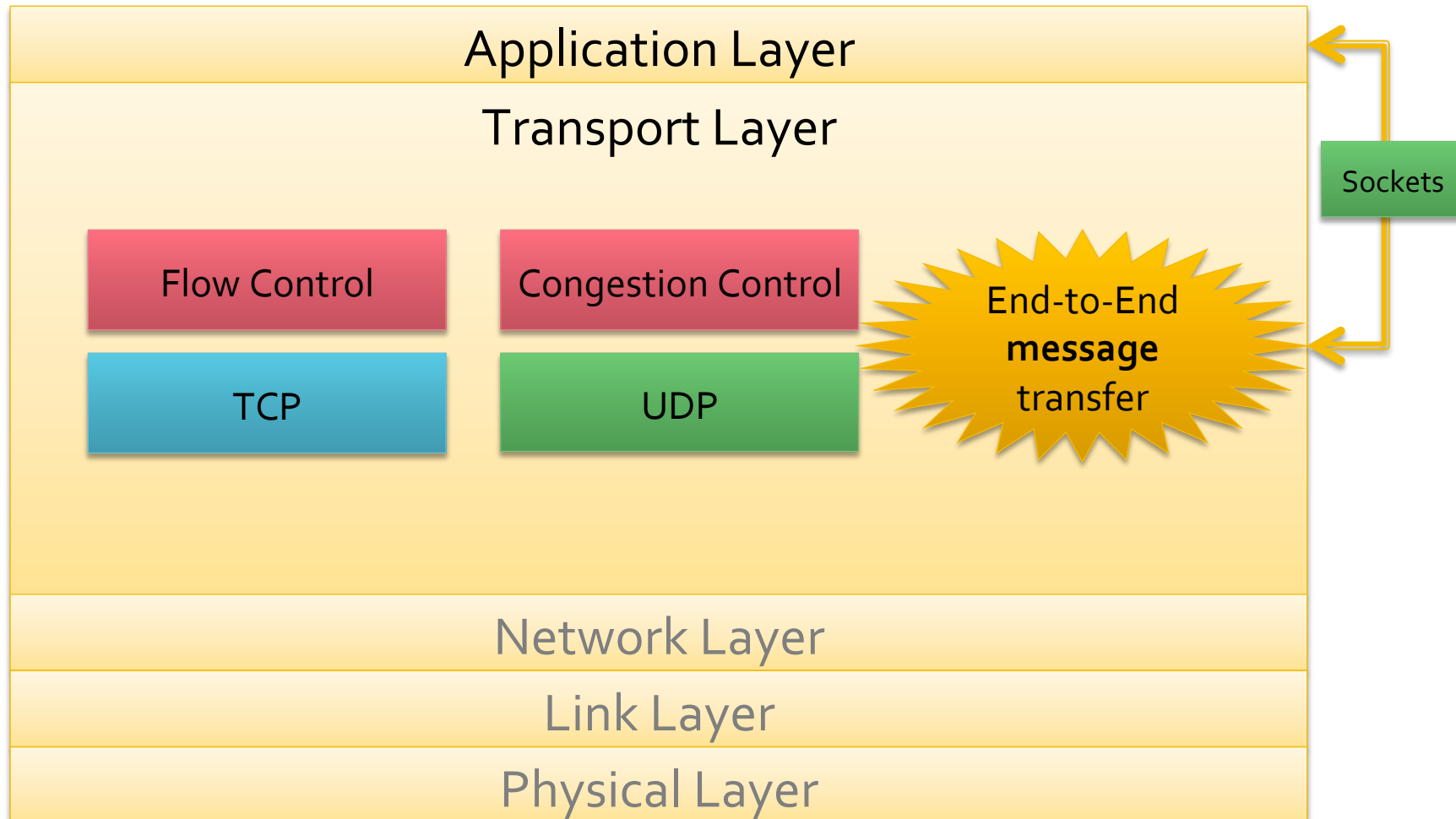
- No guarantee of delivery
- No guarantee of integrity of data

➤ **Best effort**

- Only drop packets when necessary
- No time guarantee for delivery

Ethernet networks provide the same “guarantees”

Transport Layer



“Magic” of the Internet

- **IP**: Un-reliable, order not guaranteed, delivery of **individual messages**
- **TCP**: Reliable, in-order delivery of data **stream**
- Magic
 - TCP is built on top of IP!
- Great clown analogy by Joel Spolsky
<http://www.joelonsoftware.com/articles/LeakyAbstractions.html>

Clown Delivery



Need to move clowns from Broadway to Hollywood for a new job



Broadway, NYC



Clown Delivery – Problems?



Many cars, many clowns
Bad things are guaranteed to happen to at least *some* of them

Car crash / lost



Shaved head / too ugly to work!

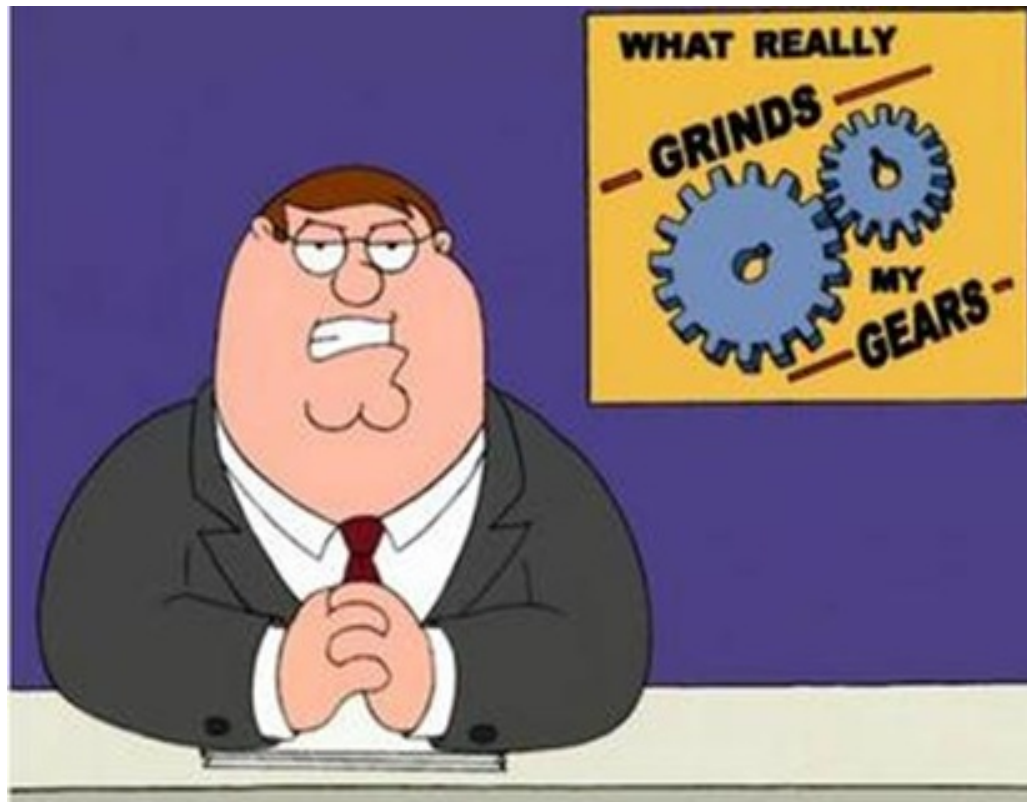


Different routes



Clown Delivery – Problems?

People in Hollywood get frustrated –
It's hard to make movies with clowns in this condition!



Clown Delivery - Solution

- New company
 - **Hollywood Express**
- Guarantees that all clowns
 - (1) Arrive
 - (2) In Order
 - (3) In Perfect Condition

- Mishap? Call and request clown's twin brother be sent immediately



- UFO crash in Nevada blocks highway?



- Clowns re-routed via Arizona
 - Director never even *hears* about the UFO crash
 - Clowns arrive a little more slowly

Networking Abstraction

- TCP provides a similar reliable delivery service for IP
- Abstraction has its limits
 - Ethernet cable chewed through by cat?
 - No useful error message for that problem!
 - The abstraction is “leaky” – it couldn’t save the user from learning about the chewed cable



Demos



Demos

1. Impersonate web browser via Telnet
2. Run `download.py` with example image
3. Monitor `download.py` with *Wireshark* and examine packet trace