

Alex Davila
1465932
CSE150/L
Chen Qian

Introduction to Computer Networks Lab

Pre-lab 2 - HTTP, DNS, and TCP

Sources:

<https://www.ietf.org/rfc/rfc2616.txt>

<https://www.ietf.org/rfc/rfc1035.txt>

<https://linux.die.net/man/>

<http://www.tcpipguide.com/free/>

HTTP Questions

1. [7pts] Choose 5 HTTP status codes and describe each one.

| | |
|--|---|
| 1.1 Informational 1xx - Request received, continuing process | <ul style="list-style-type: none">• 100 // Continue• 101 // Switching Protocols |
| 1.2 Successful 2xx - The action was successfully received, understood, and accepted | <ul style="list-style-type: none">• 200 // OK• 201 // Created• 202 // Accepted• 203 // Non-Authoritative Information• 204 // No Content• 205 // Reset Content• 206 // Partial Content |
| 1.3 Redirection 3xx - Further action must be taken in order to complete the request | <ul style="list-style-type: none">• 300 // Multiple Choices• 301 // Moved Permanently• 302 // Found• 303 // See Other• 304 // Not Modified• 305 // Use Proxy• 307 // Temporary Redirect |
| 1.4 Client Error 4xx | <ul style="list-style-type: none">• 400 // Bad Request• 401 // Unauthorized• 402 // Payment Required |

| | |
|--|---|
| <ul style="list-style-type: none"> - The request contains bad syntax or cannot be fulfilled | <ul style="list-style-type: none"> • 403 // Forbidden • 404 // Not Found • 405 // Method Not Allowed • 406 // Not Acceptable • 407 // Proxy Authentication Required • 408 // Request Time-Out • 409 // Conflict • 410 // Gone • 411 // Length Required • 412 // Precondition Failed • 413 // Request Entity Too Large • 414 // Request-URI Too Large • 415 // Unsupported Media Type • 416 // Request range not satisfiable • 417 // Expectations Failed |
| <p>1.5 Server Error 5xx</p> <ul style="list-style-type: none"> - The server failed to fulfill an apparently valid request | <ul style="list-style-type: none"> • 500 // Internal Server Error • 501 // Not Implemented • 502 // Bad Gateway • 503 // Service Unavailable • 504 // Gateway Time-Out • 505 // HTTP Version Not Supported |

2. [7 pts] List the 8 HTTP 1.1 methods and explain what they do.

SAFE & IDEMPOTENT METHODS – For Retrieval

➤ **GET**

- Retrieves whatever information is identified by the Request-URI in the form of an entity.
- If the Request-URI refers to a data-producing process, it is the produced data which shall be returned as the entity in the response and not the source text of the process, unless that text happens to be the output of the process.
- GET method changes to a “conditional GET” if the request message includes If-Modified-Since, If-Unmodified-Since, If-Match, or If-Range header field.
- A “conditional GET” method request that the entity be transferred only under the circumstances described by the conditional header field(s). It is intended for it to reduce unnecessary network usage by allowing cached entities to be refreshed without requiring multiple request or transferring data already held by the client.
- GET method changes to a “partial GET” if the request message includes a Range header field. It requests that only a part of the entity be transferred. It is intended to reduce unnecessary network usage by allowing partially-retrieved entities to be completed without transferring data already held by the client.

- The response of the `GET` method is cacheable if and only if it meets the requirements for HTTP caching.

➤ **HEAD**

- The head method is almost identical to `GET` except that the server **MUST NOT** return a message-body in the response. The metainformation contained in the HTTP headers in response to a `HEAD` request **SHOULD** be identical to the information sent in response to `GET` request.
- This method can be used for obtaining metainformation about the entity implied by the request without transferring the entity-body itself.
- This method is often used for testing hypertext links for validity, accessibility, and recent modification.
- The response to a `HEAD` request **MAY** be cacheable in the sense that the information contained in the response **MAY** be used to update a previously cached entity from that resource.
- If the new field values indicate that the cached entity differs from the current entity (as would be indicated by a change in Content-Length, Content-MD5, ETag or Last-Modified), then the cache **MUST** treat the cache entry as stale.

IDEMPOTENT METHODS – The side effects of $N > 0$ identical requests is the same as for a single request

- **PUT** – request method creates a new resource or replaces a representation of the target resource with the request payload.
- **DELETE** – The `DELETE` method requests that the origin server delete the resource identified by the Request-URI. This method **MAY** be overridden by human intervention (or other means) on the origin server.
- **POST** – The HTTP `POST` method sends data to the server. The type of the body of the request is indicated by the Content-Type header.
- **CONNECT** – The HTTP `CONNECT` method starts two-way communications with the requested resource. It can be used to open a tunnel.

OTHER IDEMPOTENT – Should not have side effects

➤ **OPTIONS**

- Represents a request for information about the communication options available on the request/response chain identified by the Request-URI allowing the client to determine the options and/or requirements associated with a resource, or the capabilities of a server, without implying or initiating a resource retrieval.
- Response to this method are not cacheable.
- If the `OPTIONS` request includes an entity-body, then the media type **MUST** be indicated by a Content-Type field. Extensions to HTTP might use the `OPTIONS` body to make more detailed queries on the server. A server that does not support such an extension **MAY** discard the request body.
- If the Request-URI is an asterisk ("*"), the `OPTIONS` request is intended to apply to the server in general rather than to a specific resource. Since a server's communication

options typically depend on the resource, the “*” request is only useful as a “ping” or “no-op” type method; it does nothing beyond allowing the client to test the capabilities of the server.

- Ex: This can be used to test a proxy for HTTP/1.1 compliance (or lack thereof).
 - If the Request-URI is not an asterisk, the OPTIONS request applies only to the options that are available when communicating with that resource.
 - A 200 response SHOULD include any header fields that indicate optional features implemented by the server and applicable to that resource (e.g, Allow), possibly including extensions not defined by this specification. The response body, if any, SHOULD also include information about the communication options.
- **TRACE** - The HTTP TRACE method performs a message loop-back test along the path to the target resource, providing a useful debugging mechanism.

`wget` and `telnet` are two commonly known commands for testing and debugging. Answer the following questions by using your mininet VM’s terminal or the Unix timeshare.

wget - Non-interactive network downloader [-Vhbe] [oadq nv iFB] [-tO nc cNST]

- `wget` is a free utility for non-interactive downloading of files from the Web. It supports HTTP, HTTPS, and FTP protocols, as well as retrieval through HTTP proxies.
- It can work in the background (non-interactive) while the user is not logged on.
- Can follow the links in HTML, XHTML, and CSS pages, to create local versions of remote web sites, fully recreating the directory structure of the original site (recursive downloading).
- Respects the Robot Exclusion Standard (/robots.txt).
- Can be restructured to convert the links in downloaded files to point at the local files, for offline viewing.
- Designed for robustness over slow or unstable network connections
 - If a download fails due to a network problem, it will keep retrying until the whole file has been retrieved.
 - If the server supports regetting, it will instruct the server to continue the download from where it left off.

3. [7 pts] Use `wget` on *example.com* to view the last modified date of the webpage. What was the HTTP return status given and what command was used to do this? (The command should not download the file! Hint: Look into the `wget` man page.)

1.1 Last modified date of the webpage:

- i. `Wget -S example.com` //downloads file
- ii. `wget -S --spider example.com` //--spider reads only
- iii. Last-modified: Thu, 17 Oct 2019 07:18:26 GMT

1.2 HTTP return status: 200 OK

- i. (the request has succeeded)
- ii. Get: The resource has been fetched and it is transmitted in the message body

1.3 Command for HTTP status:

- i. `curl -I example.com` //fetch the HTTP header only
- ii. `curl -Is example.com | head -1 | awk '{print $2}'`
 - 1. `-I` //give me just the head info
 - 2. `-s` //don't give me feedback
 - 3. `Head -1` //I only care about the 1st row
 - 4. `Awk '{print $2}'` //I only care about the 2nd column

4. [7 pts] Look up the telnet command. Use `telnet` to connect to `towel.blinkenlights.nl`.
What does the `telnet` server do?

telnet - user interface to the TELNET protocol

- Used to communicate with another host using the TELNET protocol.
- If `telnet` is invoked without the host argument, it enters command mode (`telnet>`)
 - In this mode it accepts and executes the commands listed below
 - `[-468EFKLacdfrx] [-X authtype] [-b hostalias] [-e escapechar] [-k realm] [-l user] [-n tracefile] [host [port]]`
- If it is invoked with arguments, it performs an **open** command with those arguments.
- If hostname is resolved to multiple IP addresses, **telnet** attempts to establish a connection with each address until one of them is successful or until no more addresses are left.
- `telnet>`
 - `close` //close current connection
 - `logout` //forcibly log-out remote user and close the conn.
 - `display` //display operating parameters
 - `mode` //try to enter line or character mode ('mode ?')
 - `open` //connect to a site
 - `quit` //exit telnet
 - `send` //transmit special characters ('send ?')
 - `set` //set operating parameters
 - `unset` //unset operating parameters
 - `status` //print status information
 - `toggle` //toggle operating parameters
 - `slc` //change state of special characters
 - `z` //suspend telnet
 - `!` //invoke a subshell/ shell within
 - `environ` //change environment variables
 - `?` //print help informatio

DNS Questions

5. [7 pts] In your own words describe what a DNS resource record (RR) is. Now using the command line tool *nslookup* find the *MX* resource record of *ucsc.edu*. What does this resource record mean?

- DNS resource records (RR) is the unit of information entry in DNS zone files; RRs are the building blocks of host-name and IP information and are used to resolve all DNS queries.

```
[~bash-4.2$ nslookup ucsc.edu
Server:      128.114.142.6
Address:     128.114.142.6#53
```

```
Name:   ucsc.edu
Address: 128.114.109.5
```

```
[~bash-4.2$ ls
[~bash-4.2$ nslookup
> set q=MX
> ucsc.edu
Server:      128.114.142.6
Address:     128.114.142.6#53
```

```
ucsc.edu      mail exchanger = 5 alt1.aspmx.l.google.com.
ucsc.edu      mail exchanger = 1 aspmx.l.google.com.
ucsc.edu      mail exchanger = 10 alt3.aspmx.l.google.com.
ucsc.edu      mail exchanger = 10 alt4.aspmx.l.google.com.
ucsc.edu      mail exchanger = 5 alt2.aspmx.l.google.com.
> exit
```

```
-bash-4.2$ █
```

- A mail exchanger record (MX record) specifies the mail server responsible for accepting email messages on behalf of a domain name.
- nslookup is a program to query Internet domain name servers.
 - It has two modes: Interactive and non-interactive.
 1. Interactive modes allows the user to query name servers for information about various hosts and domains or to print a list of hosts in a domain.
 2. Non-interactive mode is used to print just the name and requested information for a host or domain.

6. [7 pts] What does the command `nslookup -type=ns`. Do? Explain its output. (Note: the “.” is part of the command)
- This command changes the type of information query to an authoritative name server.

| Type | Value and Meaning |
|-------|--|
| A | A host address |
| NS | An authoritative name server |
| MD | A mail destination (Obsolete - use MX) |
| MF | A mail forwarder (Obsolete - use MX) |
| CNAME | The canonical name for an alias |
| SOA | Marks the start of a zone of authority |
| MB | A mailbox domain name (EXPERIMENTAL) |
| MG | A mail group member (EXPERIMENTAL) |
| MR | A mail rename domain name (EXPERIMENTAL) |
| NULL | A null RR (EXPERIMENTAL) |
| WKS | A well known service description |
| PTR | A domain name pointer |
| HINFO | Host information |
| MINFO | Mailbox or mail list information |
| MX | Mail exchange |
| TXT | Text strings |

TCP Questions

7. [10 pts] How can multiple applications services running on a single machine with a single IP address be uniquely identified?
- By their port address number
8. [9 pts] What is the purpose of the window mechanism in TCP?
- The purpose of the window mechanism in TCP otherwise known as Sliding Windows, is used by the Internet's TCP as a method of controlling the flow of packets between two

computers or network hosts. TCP requires that all data be acknowledged by the receiving host.

- Client and server example: each of the two devices on a connecting must keep track of the data it is sending as well as the data it is receiving from the other device.
- Windowing is done to ensure how many packets are sent at a time that depends on the size of the window and each packet is acknowledged. windowing is used to control the flow of packets(data) between two networks this method is used to get acknowledgment for each packet received at the receiver side

9. [9 pts] What is an MTU? What happens when a packet is larger than the MTU?

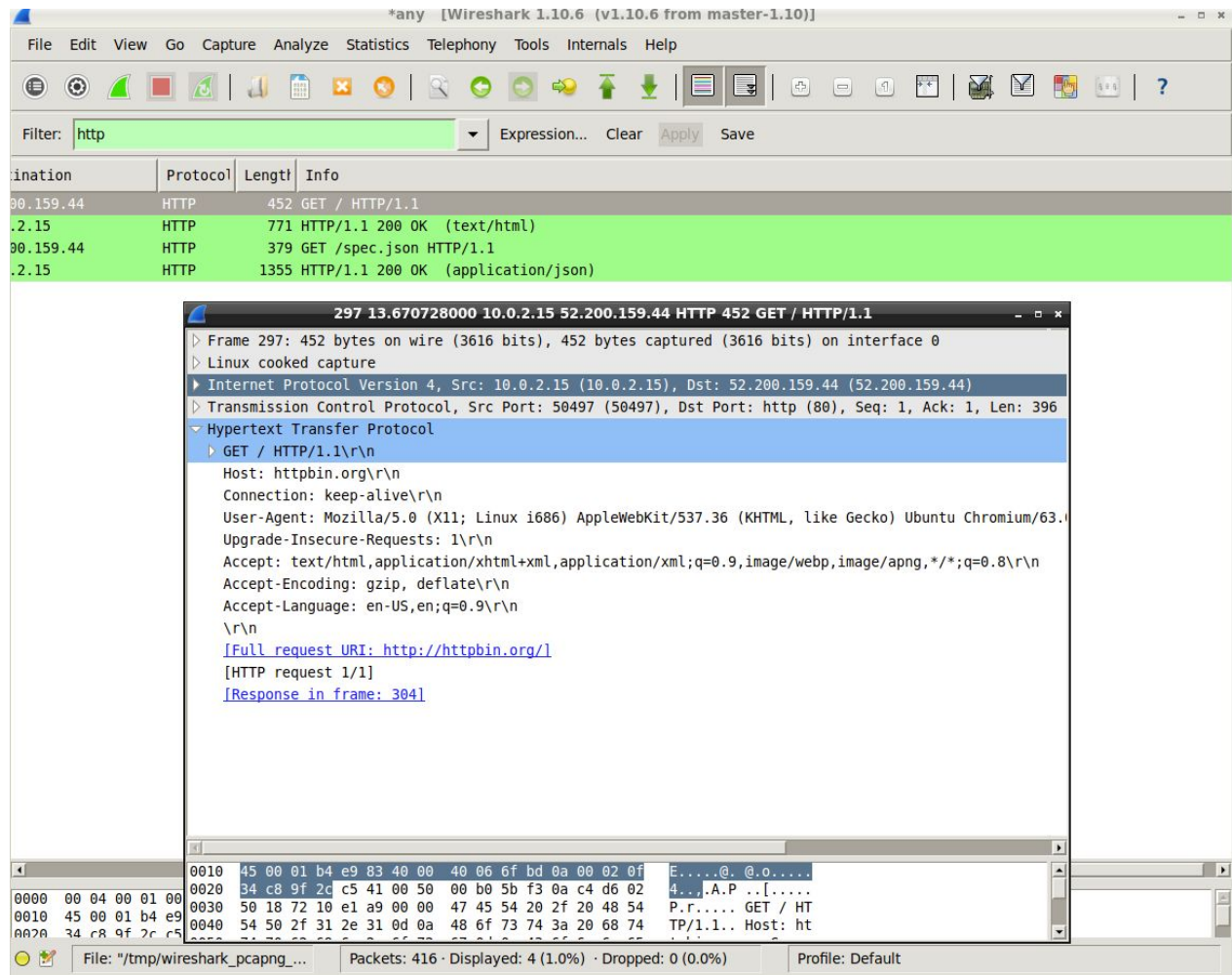
- Network's maximum transmission unit
- MTU : maximum transport unit (size of packet)
- part 3: If the packet size exceeds MTU then fragmentation (dividing the packets into small packets and these packets are reassembled at the receiver side into its original size) is done. piggybacking is acknowledgment sent along with data packet.

Lab 2 - HTTP, DNS, and TCP

Part 1: HTTP

In this section, we will observe how the HTTP protocol operates. We will do this by using the Mininet VM. Begin by opening Wireshark and listening on the 'any' interface. Open Chromium and navigate to <http://httpbin.org>

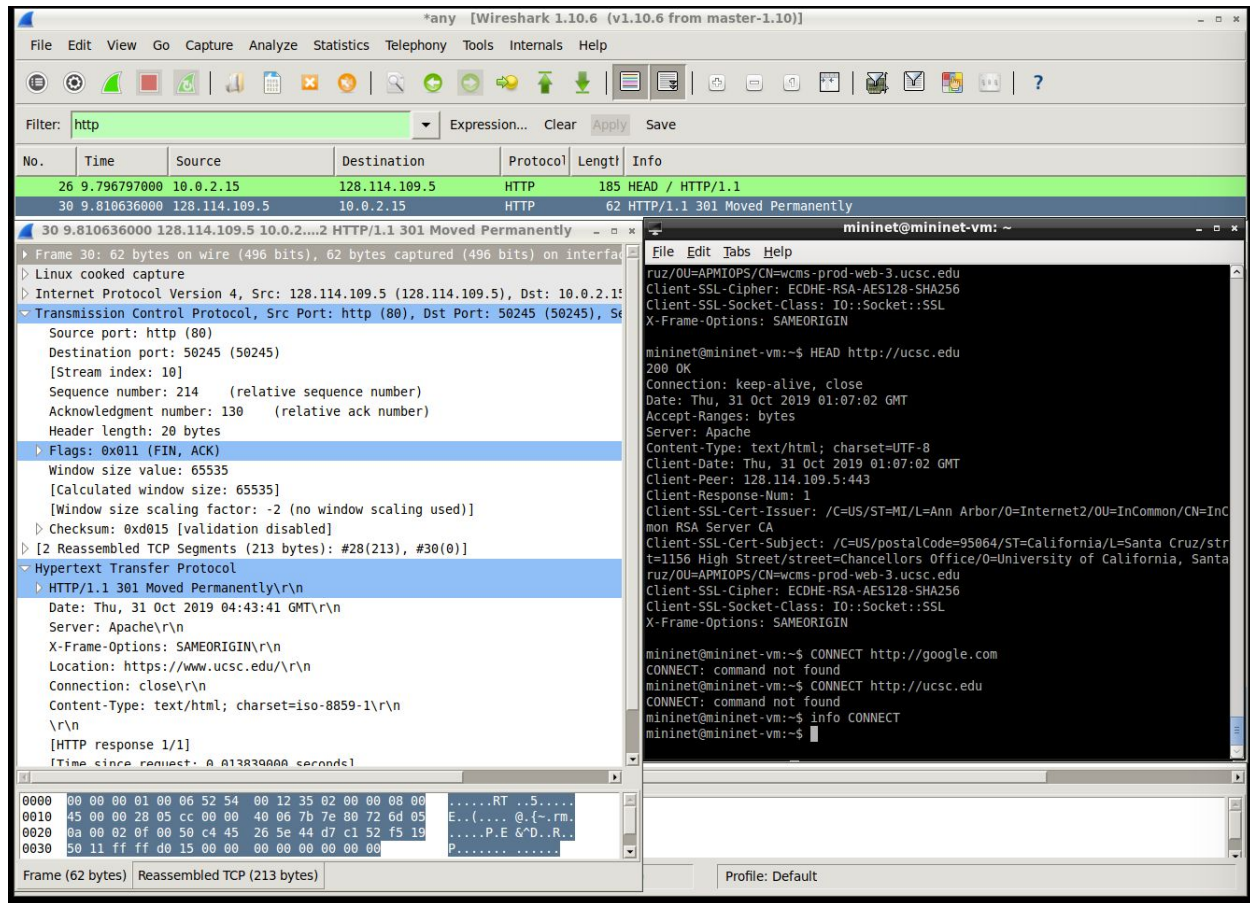
1. [10 pts] Find the HTTP packet that corresponds to the **initial request** that your computer made. Take a screenshot of this packet. What HTTP method did your computer use to make this request?
 - a. HTTP packet:
 - b. Screenshot:



- c. HTTP method used: GET

2. [10 pts] Find the HTTP packet that corresponds to the **initial response** the server made to your request. Take a screenshot of this packet. What HTTP status code did the server return? What is the content type of the response the server is sending back? Using Chromium and navigate to <http://ucsc.edu>

- Response HTTP packet: No.: 30 Time: 9.8106... S:128.114.109.5 D:10.0.2.15 Protocol: HTTP Length: 62
Info: Head /HTTP/1.1 301 Moved Permanently
- Screenshot:



- HTTP Status Code returned: 301 Moved Permanently
- Content type: text/html; charset=UTF-8

3. [10 pts] Find the HTTP packets that correspond to the **initial request and response** that your computer made. Take a screenshot of these packets. What's different? Explain. Using Chromium (or any other Linux utility you are comfortable with), find a way to make a HTTP packet with a method other than GET.

- HTTP request packet: (2) initial request and response

b. Initial Request Screenshot:

Wireshark 1.10.6 (v1.10.6 from master-1.10) interface showing an initial HTTP request. The filter is set to http. The packet list shows a POST request from 10.0.2.15 to 172.217.6.46. The packet details pane shows the request structure, including headers like User-Agent, Host, and Content-Type. The packet bytes pane shows the raw data.

| No. | Time | Source | Destination | Protocol | Length | Info |
|-----|--------------|--------------|--------------|----------|--------|---|
| 38 | 13.736038000 | 10.0.2.15 | 172.217.6.46 | HTTP | 71 | POST / HTTP/1.1 (application/x-www-form-urlencoded) |
| 40 | 13.772591000 | 172.217.6.46 | 10.0.2.15 | HTTP | 1860 | HTTP/1.1 405 Method Not Allowed (text/html) |

Packet 38 details:

- Frame 38: 71 bytes on wire (568 bits), 71 bytes captured (568 bits) on interface
- Linux cooked capture
- Internet Protocol Version 4, Src: 10.0.2.15 (10.0.2.15), Dst: 172.217.6.46 (172.217.6.46)
- Transmission Control Protocol, Src Port: 57562 (57562), Dst Port: http (80), Seq: 306553000, Win: 0, Len: 0
- Hypertext Transfer Protocol
 - POST / HTTP/1.1
 - User-Agent: wget/1.15 (linux-gnu)
 - Accept: */*
 - Host: google.com
 - Connection: Keep-Alive
 - Content-Type: application/x-www-form-urlencoded
 - Content-Length: 15
 - Full request URI: http://google.com/
 - HTTP request 1/1
 - Response in frame: 40
- Line-based text data: application/x-www-form-urlencoded
 - username=Yankee

c. Initial Response Screenshot:

Wireshark 1.10.6 (v1.10.6 from master-1.10) interface showing an initial HTTP response. The filter is set to http. The packet list shows a 405 Method Not Allowed response from 172.217.6.46 to 10.0.2.15. The packet details pane shows the response structure, including headers like Allow, Date, Content-Type, and Server. The packet bytes pane shows the raw data.

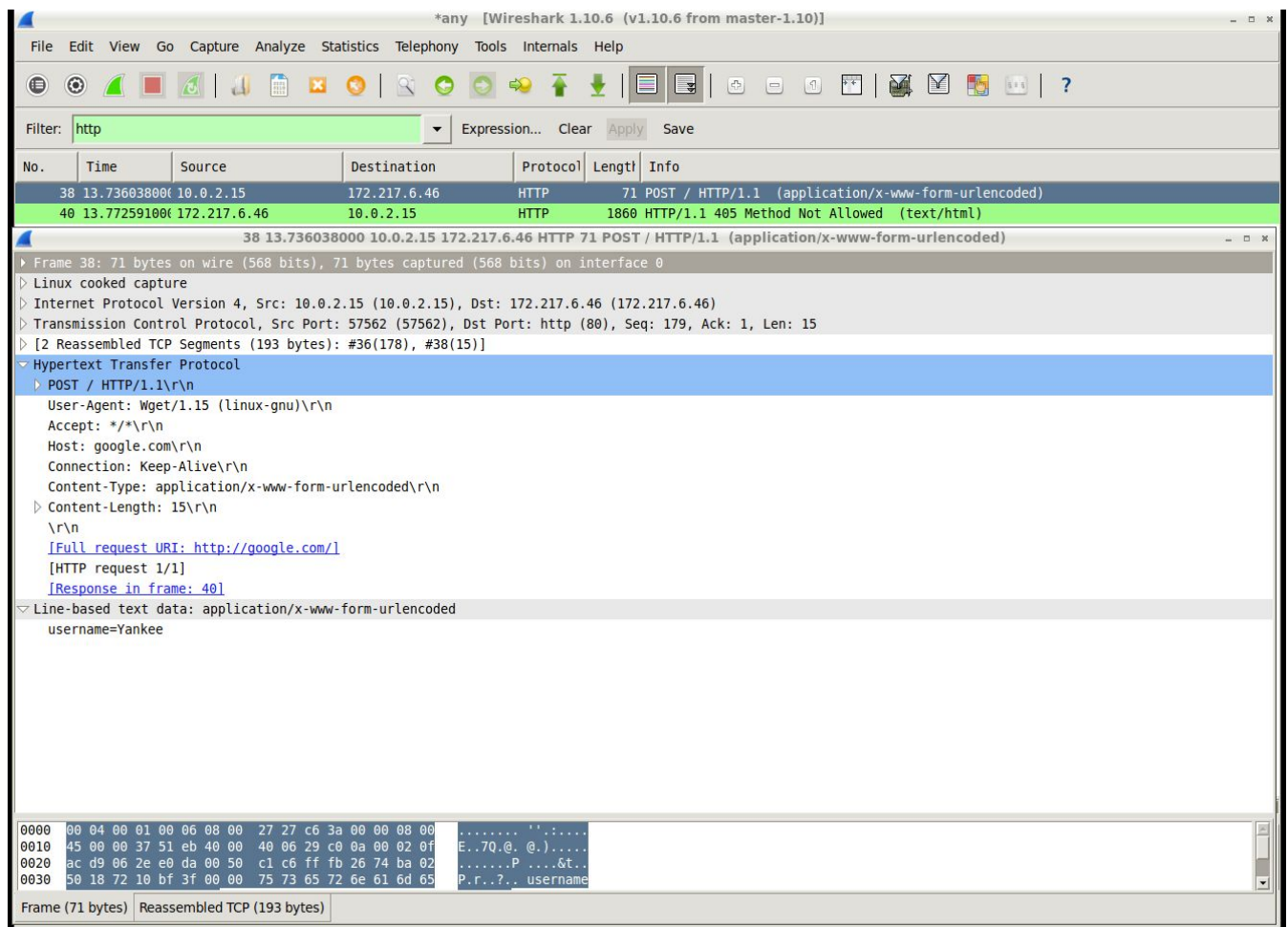
| No. | Time | Source | Destination | Protocol | Length | Info |
|-----|--------------|--------------|--------------|----------|--------|---|
| 38 | 13.736038000 | 10.0.2.15 | 172.217.6.46 | HTTP | 71 | POST / HTTP/1.1 (application/x-www-form-urlencoded) |
| 40 | 13.772591000 | 172.217.6.46 | 10.0.2.15 | HTTP | 1860 | HTTP/1.1 405 Method Not Allowed (text/html) |

Packet 40 details:

- Frame 40: 1860 bytes on wire (14880 bits), 1860 bytes captured (14880 bits) on interface
- Linux cooked capture
- Internet Protocol Version 4, Src: 172.217.6.46 (172.217.6.46), Dst: 10.0.2.15 (10.0.2.15)
- Transmission Control Protocol, Src Port: http (80), Dst Port: 57562 (57562), Seq: 306553001, Win: 0, Len: 0
- Hypertext Transfer Protocol
 - HTTP/1.1 405 Method Not Allowed
 - Allow: GET, HEAD
 - Date: Thu, 31 Oct 2019 04:53:16 GMT
 - Content-Type: text/html; charset=UTF-8
 - Server: gws
 - Content-Length: 1589
 - X-XSS-Protection: 0
 - X-Frame-Options: SAMEORIGIN
 - HTTP response 1/1
 - Time since request: 0.036553000 seconds
 - Request in frame: 38
- Line-based text data: text/html
 - <!DOCTYPE html>
 - <html lang=en>
 - <meta charset=utf-8>
 - <meta name=viewport content=initial-scale=1, minimum-scale=1, width=device-width>
 - <title>Error 405 (Method Not Allowed)</title>
 - <style>
 - [truncated] *{margin:0;padding:0}html,code{font:15px/22px arial,sans-serif}
 - </style>
 - Google
 - <p>405. That's an error.</p>

- d. What is the difference? In the POST request I am submitting a `application/x-www-form-urlencoded` for the content type using the string `username=Yankee`. The difference is that under POST I am actually sending data to the server whereas the response I received a response of a 405 Method not allowed with the content type of `text/html`.

4. [10 pts] Take a screenshot of your packet, and explain what you did to create it



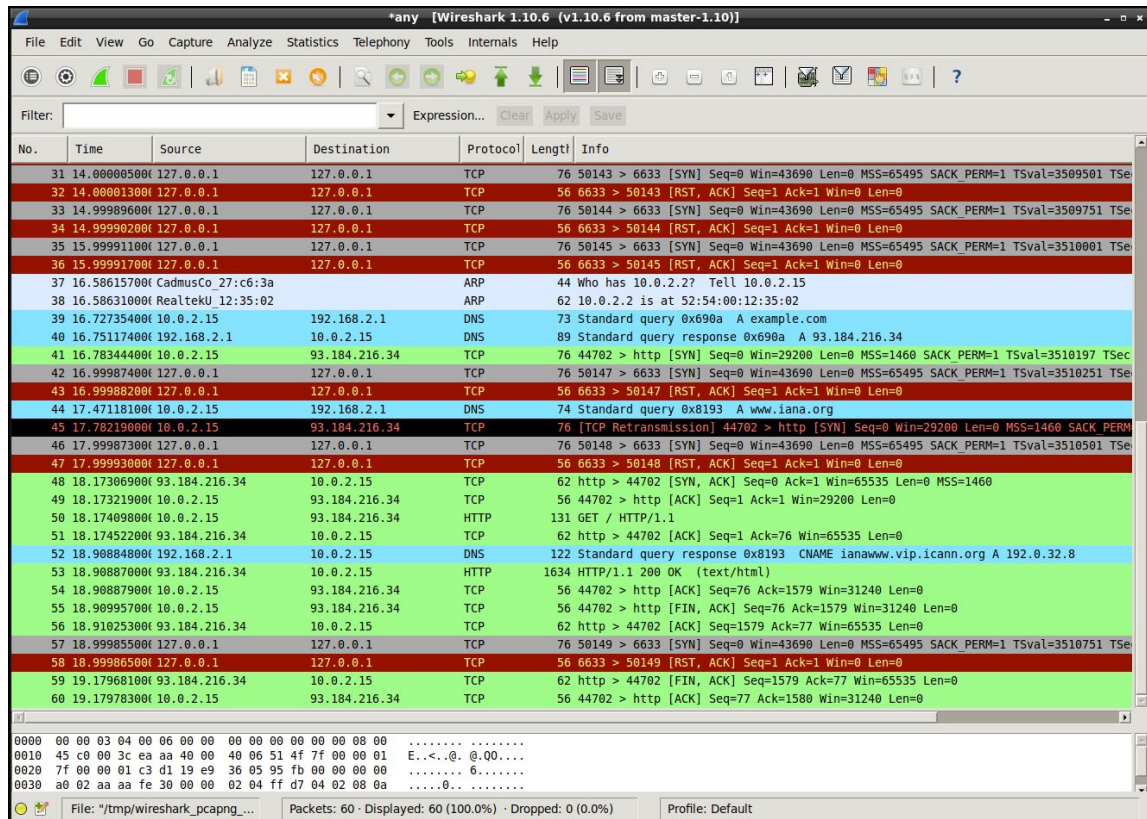
I used the terminal with the Linux utility/command `wget--post-data "username=Yankee"` `http://google.com` to post the username Yankee to the webpage Google.com.

Part 2: DNS

In this section, we will observe how the DNS protocol operates. We will do this by using the Mininet VM. Begin by opening Wireshark and listening on the ‘any’ interface.

Open Chromium and navigate to www.example.com.

5. [10 pts] Were any steps taken by your computer before the web page was loaded? If so, using your captured packets in Wireshark, find the packets that allowed your computer to successfully load <http://www.example.com>. Take a screenshot of these packets, and explain why you think these are the correct packets. What’s the IP address of www.example.com?



| No. | Time | Source | Destination | Protocol | Length | Info |
|-----|-------------|-------------------|---------------|----------|--------|--|
| 31 | 14.00005900 | 127.0.0.1 | 127.0.0.1 | TCP | 76 | 50143 > 6633 [SYN] Seq=0 Win=43690 Len=0 MSS=65495 SACK_PERM=1 TSval=3509501 TSe |
| 32 | 14.00001300 | 127.0.0.1 | 127.0.0.1 | TCP | 56 | 6633 > 50143 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0 |
| 33 | 14.99989600 | 127.0.0.1 | 127.0.0.1 | TCP | 76 | 50144 > 6633 [SYN] Seq=0 Win=43690 Len=0 MSS=65495 SACK_PERM=1 TSval=3509751 TSe |
| 34 | 14.99990200 | 127.0.0.1 | 127.0.0.1 | TCP | 56 | 6633 > 50144 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0 |
| 35 | 15.99991100 | 127.0.0.1 | 127.0.0.1 | TCP | 76 | 50145 > 6633 [SYN] Seq=0 Win=43690 Len=0 MSS=65495 SACK_PERM=1 TSval=3510001 TSe |
| 36 | 15.99991700 | 127.0.0.1 | 127.0.0.1 | TCP | 56 | 6633 > 50145 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0 |
| 37 | 16.58615700 | CadmusCo 27:c6:3a | | ARP | 44 | Who has 10.0.2.2? Tell 10.0.2.15 |
| 38 | 16.58631000 | RealtekU 12:35:02 | | ARP | 62 | 10.0.2.2 is at 52:54:00:12:35:02 |
| 39 | 16.72735400 | 10.0.2.15 | 192.168.2.1 | DNS | 73 | Standard query 0x690a A example.com |
| 40 | 16.75117400 | 192.168.2.1 | 10.0.2.15 | DNS | 89 | Standard query response 0x690a A 93.184.216.34 |
| 41 | 16.78344400 | 10.0.2.15 | 93.184.216.34 | TCP | 76 | 44702 > http [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=3510197 TSec |
| 42 | 16.99987400 | 127.0.0.1 | 127.0.0.1 | TCP | 76 | 50147 > 6633 [SYN] Seq=0 Win=43690 Len=0 MSS=65495 SACK_PERM=1 TSval=3510251 TSe |
| 43 | 16.99988200 | 127.0.0.1 | 127.0.0.1 | TCP | 56 | 6633 > 50147 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0 |
| 44 | 17.47118100 | 10.0.2.15 | 192.168.2.1 | DNS | 74 | Standard query 0x8193 A www.iana.org |
| 45 | 17.78219000 | 10.0.2.15 | 93.184.216.34 | TCP | 76 | [TCP Retransmission] 44702 > http [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM |
| 46 | 17.99987300 | 127.0.0.1 | 127.0.0.1 | TCP | 76 | 50148 > 6633 [SYN] Seq=0 Win=43690 Len=0 MSS=65495 SACK_PERM=1 TSval=3510501 TSe |
| 47 | 17.99993000 | 127.0.0.1 | 127.0.0.1 | TCP | 56 | 6633 > 50148 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0 |
| 48 | 18.17306900 | 93.184.216.34 | 10.0.2.15 | TCP | 62 | http > 44702 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 |
| 49 | 18.17321900 | 10.0.2.15 | 93.184.216.34 | TCP | 56 | 44702 > http [ACK] Seq=1 Ack=1 Win=29200 Len=0 |
| 50 | 18.17409800 | 10.0.2.15 | 93.184.216.34 | HTTP | 131 | GET / HTTP/1.1 |
| 51 | 18.17452200 | 93.184.216.34 | 10.0.2.15 | TCP | 62 | http > 44702 [ACK] Seq=1 Ack=76 Win=65535 Len=0 |
| 52 | 18.90884800 | 192.168.2.1 | 10.0.2.15 | DNS | 122 | Standard query response 0x8193 CNAME ianaww.vip.icann.org A 192.0.32.8 |
| 53 | 18.90887000 | 93.184.216.34 | 10.0.2.15 | HTTP | 1634 | HTTP/1.1 200 OK (text/html) |
| 54 | 18.90887900 | 10.0.2.15 | 93.184.216.34 | TCP | 56 | 44702 > http [ACK] Seq=76 Ack=1579 Win=31240 Len=0 |
| 55 | 18.90995700 | 10.0.2.15 | 93.184.216.34 | TCP | 56 | 44702 > http [FIN, ACK] Seq=76 Ack=1579 Win=31240 Len=0 |
| 56 | 18.91025300 | 93.184.216.34 | 10.0.2.15 | TCP | 62 | http > 44702 [ACK] Seq=1579 Ack=77 Win=65535 Len=0 |
| 57 | 18.99985500 | 127.0.0.1 | 127.0.0.1 | TCP | 76 | 50149 > 6633 [SYN] Seq=0 Win=43690 Len=0 MSS=65495 SACK_PERM=1 TSval=3510751 TSe |
| 58 | 18.99986500 | 127.0.0.1 | 127.0.0.1 | TCP | 56 | 6633 > 50149 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0 |
| 59 | 19.17968100 | 93.184.216.34 | 10.0.2.15 | TCP | 62 | http > 44702 [FIN, ACK] Seq=1579 Ack=77 Win=65535 Len=0 |
| 60 | 19.17978300 | 10.0.2.15 | 93.184.216.34 | TCP | 56 | 44702 > http [ACK] Seq=77 Ack=1580 Win=31240 Len=0 |

Here is what happened before the computer successfully loaded www.example.com.

| | | | | | | |
|----|-------------|-------------------|---------------|-----|----|--|
| 37 | 16.58615700 | CadmusCo 27:c6:3a | | ARP | 44 | Who has 10.0.2.2? Tell 10.0.2.15 |
| 38 | 16.58631000 | RealtekU 12:35:02 | | ARP | 62 | 10.0.2.2 is at 52:54:00:12:35:02 |
| 39 | 16.72735400 | 10.0.2.15 | 192.168.2.1 | DNS | 73 | Standard query 0x690a A example.com |
| 40 | 16.75117400 | 192.168.2.1 | 10.0.2.15 | DNS | 89 | Standard query response 0x690a A 93.184.216.34 |
| 41 | 16.78344400 | 10.0.2.15 | 93.184.216.34 | TCP | 76 | 44702 > http [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=3510197 TSec |
| 42 | 16.99987400 | 127.0.0.1 | 127.0.0.1 | TCP | 76 | 50147 > 6633 [SYN] Seq=0 Win=43690 Len=0 MSS=65495 SACK_PERM=1 TSval=3510251 TSe |
| 43 | 16.99988200 | 127.0.0.1 | 127.0.0.1 | TCP | 56 | 6633 > 50147 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0 |
| 44 | 17.47118100 | 10.0.2.15 | 192.168.2.1 | DNS | 74 | Standard query 0x8193 A www.iana.org |
| 45 | 17.78219000 | 10.0.2.15 | 93.184.216.34 | TCP | 76 | [TCP Retransmission] 44702 > http [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM |
| 46 | 17.99987300 | 127.0.0.1 | 127.0.0.1 | TCP | 76 | 50148 > 6633 [SYN] Seq=0 Win=43690 Len=0 MSS=65495 SACK_PERM=1 TSval=3510501 TSe |
| 47 | 17.99993000 | 127.0.0.1 | 127.0.0.1 | TCP | 56 | 6633 > 50148 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0 |
| 48 | 18.17306900 | 93.184.216.34 | 10.0.2.15 | TCP | 62 | http > 44702 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 |
| 49 | 18.17321900 | 10.0.2.15 | 93.184.216.34 | TCP | 56 | 44702 > http [ACK] Seq=1 Ack=1 Win=29200 Len=0 |

IP Address: 93.184.216.34 93.184.216.34 93.184.216.34

The reason why this happened was because the local host had to perform an ARP request, DNS resolution and perform a TCP three way handshake before loading the actual website.

6. [10 pts] Open a terminal window. Execute the command to flush your DNS cache: `sudo /etc/init.d/networking restart`. Using `wget`, download the same content of `www.example.com` with its IP address you discovered in question 5, without sending DNS requests.

a. What command did you use to accomplish that?

`wget example.com`

b. Take a screenshot of related packets and explain why you think these are the correct packets.

The screenshot displays a network traffic analysis using Wireshark and terminal commands. The top section shows a Wireshark capture of an HTTP GET request to 93.184.216.34. The bottom section shows a terminal window where the user runs `nslookup -type=ns. example.com` and `wget 93.184.216.34`, and a Chromium browser window showing the 'Example Domain'.

Wireshark Packet List:

| No. | Time | Source | Destination | Protocol | Length | Info |
|-----|-------------|---------------|---------------|----------|--------|---|
| 1 | 0.000000000 | 127.0.0.1 | 127.0.0.1 | TCP | 76 | 51082 > 6633 [SYN] Seq=0 Win=43690 Len=0 MSS=65535 |
| 2 | 0.000007000 | 127.0.0.1 | 127.0.0.1 | TCP | 56 | 6633 > 51082 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0 |
| 3 | 0.999589000 | 127.0.0.1 | 127.0.0.1 | TCP | 76 | 51083 > 6633 [SYN] Seq=0 Win=43690 Len=0 MSS=65535 |
| 4 | 0.999595000 | 127.0.0.1 | 127.0.0.1 | TCP | 56 | 6633 > 51083 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0 |
| 5 | 1.999882000 | 127.0.0.1 | 127.0.0.1 | TCP | 76 | 51084 > 6633 [SYN] Seq=0 Win=43690 Len=0 MSS=65535 |
| 6 | 1.999888000 | 127.0.0.1 | 127.0.0.1 | TCP | 56 | 6633 > 51084 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0 |
| 7 | 2.969839000 | 10.0.2.15 | 93.184.216.34 | TCP | 76 | 45641 > http [SYN] Seq=0 Win=29280 Len=0 MSS=65535 |
| 8 | 2.963652000 | 93.184.216.34 | 10.0.2.15 | TCP | 62 | http > 45641 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 |
| 9 | 2.983759000 | 10.0.2.15 | 93.184.216.34 | TCP | 56 | 45641 > http [ACK] Seq=1 Ack=1 Win=29280 Len=0 |
| 10 | 2.984143000 | 10.0.2.15 | 93.184.216.34 | HTTP | 167 | GET / HTTP/1.1 |
| 11 | 2.984458000 | 93.184.216.34 | 10.0.2.15 | TCP | 62 | http > 45641 [ACK] Seq=1 Ack=112 Win=65535 Len=0 |
| 12 | 3.000186000 | 127.0.0.1 | 127.0.0.1 | TCP | 76 | 51086 > 6633 [SYN] Seq=0 Win=43690 Len=0 MSS=65535 |
| 13 | 3.000193000 | 127.0.0.1 | 127.0.0.1 | TCP | 56 | 6633 > 51086 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0 |
| 14 | 3.001409000 | 93.184.216.34 | 10.0.2.15 | HTTP/XML | 334 | HTTP/1.1 404 Not Found |
| 15 | 3.001425000 | 10.0.2.15 | 93.184.216.34 | TCP | 56 | 45641 > http [ACK] Seq=112 Ack=479 Win=38016 Len=0 |
| 16 | 3.001966000 | 10.0.2.15 | 93.184.216.34 | TCP | 56 | 45641 > http [FIN, ACK] Seq=112 Ack=479 Win=38016 Len=0 |
| 17 | 3.002073000 | 93.184.216.34 | 10.0.2.15 | TCP | 62 | http > 45641 [ACK] Seq=479 Ack=113 Win=65535 Len=0 |
| 18 | 3.019367000 | 93.184.216.34 | 10.0.2.15 | TCP | 62 | http > 45641 [FIN, ACK] Seq=479 Ack=113 Win=65535 Len=0 |
| 19 | 3.019398000 | 10.0.2.15 | 93.184.216.34 | TCP | 56 | 45641 > http [ACK] Seq=113 Ack=480 Win=38016 Len=0 |
| 20 | 4.000444000 | 127.0.0.1 | 127.0.0.1 | TCP | 76 | 51087 > 6633 [SYN] Seq=0 Win=43690 Len=0 MSS=65535 |
| 21 | 4.000451000 | 127.0.0.1 | 127.0.0.1 | TCP | 56 | 6633 > 51087 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0 |
| 22 | 5.000087000 | 127.0.0.1 | 127.0.0.1 | TCP | 76 | 51088 > 6633 [SYN] Seq=0 Win=43690 Len=0 MSS=65535 |
| 23 | 5.000094000 | 127.0.0.1 | 127.0.0.1 | TCP | 56 | 6633 > 51088 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0 |
| 24 | 6.000459000 | 127.0.0.1 | 127.0.0.1 | TCP | 76 | 51089 > 6633 [SYN] Seq=0 Win=43690 Len=0 MSS=65535 |
| 25 | 6.000457000 | 127.0.0.1 | 127.0.0.1 | TCP | 56 | 6633 > 51089 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0 |

Terminal Output:

```
mininet@mininet-vm:~$ nslookup -type=ns. example.com
Server: 192.168.2.1
Address: 192.168.2.1#53
Non-authoritative answer:
Name: example.com
Address: 93.184.216.34

mininet@mininet-vm:~$ wget 93.184.216.34
--2019-10-30 22:37:11-- http://93.184.216.34/
Connecting to 93.184.216.34:80... connected.
HTTP request sent, awaiting response... 404 Not Found
2019-10-30 22:37:11 ERROR 404: Not Found.

mininet@mininet-vm:~$
```

Chromium Browser: Example Domain - Chromium. The page content states: "This domain is for use in illustrative examples in documents. You may use this domain in literature without prior coordination or asking for permission. More information..."

The reason why these packets are the correct ones is because when I run `nslookup -type=ns. example.com` it resolves to the same IP that I got using `wget 93.184.216.34`

Open a terminal window. Using nslookup, find the A records for www.google.com.

7. [10 pts] Take a screenshot of the packets corresponding to your request, and the response from the server. If the request was resolved, what is the IP address you were given for www.google.com?

The screenshot shows a Wireshark packet capture window with the filter 'dns'. Two packets are displayed:

| No. | Time | Source | Destination | Protocol | Length | Info |
|-----|-------------|-------------|-------------|----------|--------|--|
| 15 | 6.629117000 | 10.0.2.15 | 192.168.2.1 | DNS | 72 | Standard query 0x797f A google.com |
| 16 | 6.643937000 | 192.168.2.1 | 10.0.2.15 | DNS | 88 | Standard query response 0x797f A 172.217.164.110 |

Below the packet list, a terminal window titled 'mininet@mininet-vm: ~' is open, showing the output of 'nslookup' commands:

```
mininet@mininet-vm: ~  
File Edit Tabs Help  
  
Non-authoritative answer:  
Name:   example.com  
Address: 93.184.216.34  
  
mininet@mininet-vm:~$ nslookup -type=ns. example.com  
unknown query type: ns.  
Server:      192.168.2.1  
Address:     192.168.2.1#53  
  
Non-authoritative answer:  
Name:   example.com  
Address: 93.184.216.34  
  
mininet@mininet-vm:~$ nslookup -type=ns. google.com  
unknown query type: ns.  
Server:      192.168.2.1  
Address:     192.168.2.1#53  
  
Non-authoritative answer:  
Name:   google.com  
Address: 172.217.164.110  
  
mininet@mininet-vm:~$
```

At the bottom of the Wireshark window, the packet details pane shows the raw DNS data for the selected packet:

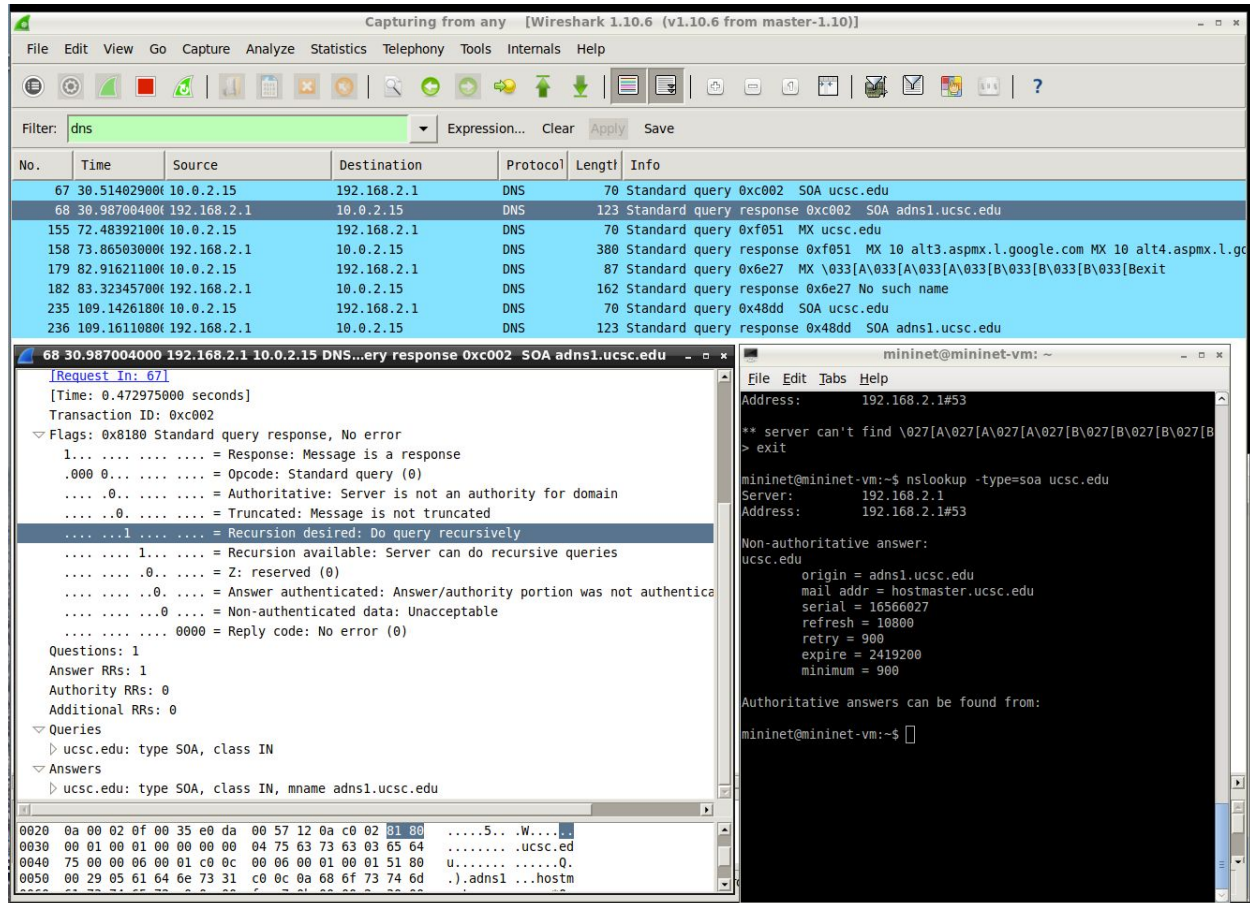
```
0000  00 04 00 01 00 06 08 00 27 27 c6 3a 00 00 08 00  .....':....  
0010  45 00 00 38 b3 c9 00 00 40 11 f8 33 0a 00 02 0f  E..8....@.3...  
0020  c0 a8 02 01 bd b2 00 35 00 24 ce ed 79 7f 01 00  .....5$.y...  
0030  00 01 00 00 00 00 00 00 06 67 6f 6f 67 6c 65 03  .....google.
```

The request was resolved: A 172.217.164.110

*nslookup gives you back host A records unless specified

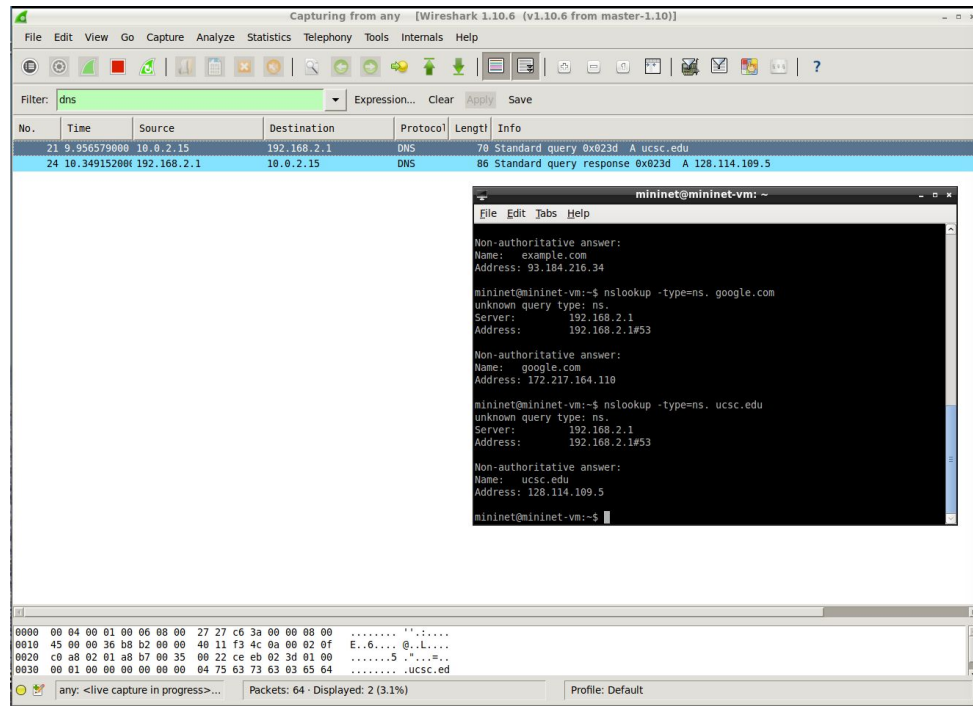
8. [10 pts] Did your computer want to complete the request recursively? How do you know? Take a screenshot proving your answer.

A: My computer completed the request recursively because it does so by default and because recursion was available from the server. Here is the proof:



Using nslookup, find the A records for ucsc.edu.

9. [10 pts] Take a screenshot of the packets corresponding to your request, and the response from the server.



a. If the request was resolved, what is the IP address you were given for ucsc.edu?

A records: A 128.114.109.5

10. [10 pts] What is the authoritative name server for the ucsc.edu domain? adns1.ucsc.edu

How do you know?

- By typing: nslookup -type=ns ucsc.edu

Take a screenshot proving your answer:

```
mininet@mininet-vm: ~  
File Edit Tabs Help  
Authoritative answers can be found from:  
  
mininet@mininet-vm:~$ nslookup -type=ns.  
unknown query type: ns.  
> ucsc.edu  
Server:          192.168.2.1  
Address:         192.168.2.1#53  
  
Non-authoritative answer:  
Name:   ucsc.edu  
Address: 128.114.109.5  
> exit  
  
mininet@mininet-vm:~$ nslookup -type=ns ucsc.edu  
Server:          192.168.2.1  
Address:         192.168.2.1#53  
  
Non-authoritative answer:  
ucsc.edu      nameserver = adns2.ucsc.edu.  
ucsc.edu      nameserver = sns-pb.isc.org.  
ucsc.edu      nameserver = adns1.ucsc.edu.  
ucsc.edu      nameserver = ns.zocalo.net.  
  
Authoritative answers can be found from:  
sns-pb.isc.org internet address = 192.5.4.1  
sns-pb.isc.org has AAAA address 2001:500:2e::1  
adns1.ucsc.edu internet address = 128.114.100.100  
adns1.ucsc.edu has AAAA address 2607:f5f0:2::100  
adns2.ucsc.edu internet address = 128.114.100.200  
adns2.ucsc.edu has AAAA address 2607:f5f0:2::200  
  
mininet@mininet-vm:~$
```

Part 3: TCP

In this section, we will observe how the TCP protocol operates. We will do this by using the Mininet VM. Begin by opening Wireshark and listening on the 'any' interface.

Open a terminal window. Using wget, download the file <http://ipv4.download.thinkbroadband.com/10MB.zip>

11. [10 pts] Find the packets corresponding with the SYN, SYN-ACK, and ACK that initiated the TCP connection for this file transfer.

a. Take a screenshot of these packets.

The screenshot displays a Wireshark network packet capture and a terminal window. The Wireshark interface is set to capture on the 'any' interface. The filter is 'tcp.stream eq 27'. The packet list shows three packets: a SYN packet (No. 61), a SYN-ACK packet (No. 62), and an ACK packet (No. 63). The packet details pane shows the 'Follow TCP Stream' view for stream 27, displaying the HTTP request and response. The terminal window shows the execution of 'nslookup' and 'wget' commands, confirming the download of '10MB.zip' from 'http://ipv4.download.thinkbroadband.com/10MB.zip'.

| No. | Time | Source | Destination | Protocol | Length | Info |
|-----|-------------|---------------|---------------|----------|--------|---|
| 61 | 24.24189600 | 10.0.2.15 | 80.249.99.148 | TCP | 76 | 33727 > http [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK PERM=1 TSval=4419061 TSecr=0 WS= |
| 62 | 24.39766100 | 80.249.99.148 | 10.0.2.15 | TCP | 62 | http > 33727 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 |
| 63 | 24.39769600 | 10.0.2.15 | 80.249.99.148 | TCP | 56 | 33727 > http [ACK] Seq=1 Ack=1 Win=29200 Len=0 |

Stream Content:

```
GET /10MB.zip HTTP/1.1
User-Agent: Wget/1.15 (linux-gnu)
Accept: */*
Host: ipv4.download.thinkbroadband.com
Connection: Keep-Alive

HTTP/1.1 200 OK
Server: nginx
Date: Thu, 31 Oct 2019 06:15:18 GMT
Content-Type: application/zip
Content-Length: 10485760
Last-Modified: Mon, 02 Jun 2008 15:30:33 GMT
Connection: keep-alive
ETag: "48441219-a00000"
Access-Control-Allow-Origin: *
Accept-Ranges: bytes

D.i.j..Pn.(;.....E...).5
[.dPI..7n....s...n...z.K.v.`7..~.....X...4.Y.Z.....2su...)}...k...t.x..n.S
B....(.:I[.m..D.af.>...n.iN..Z..~cgo"...+.....nS..g.X...:'.W....I.....m..
\X...5.+6...^.....[W.....b4..0.L='b.t0....q...5.F.#a,Y.....
```

b. What was the initial window size that your computer advertised to the server?

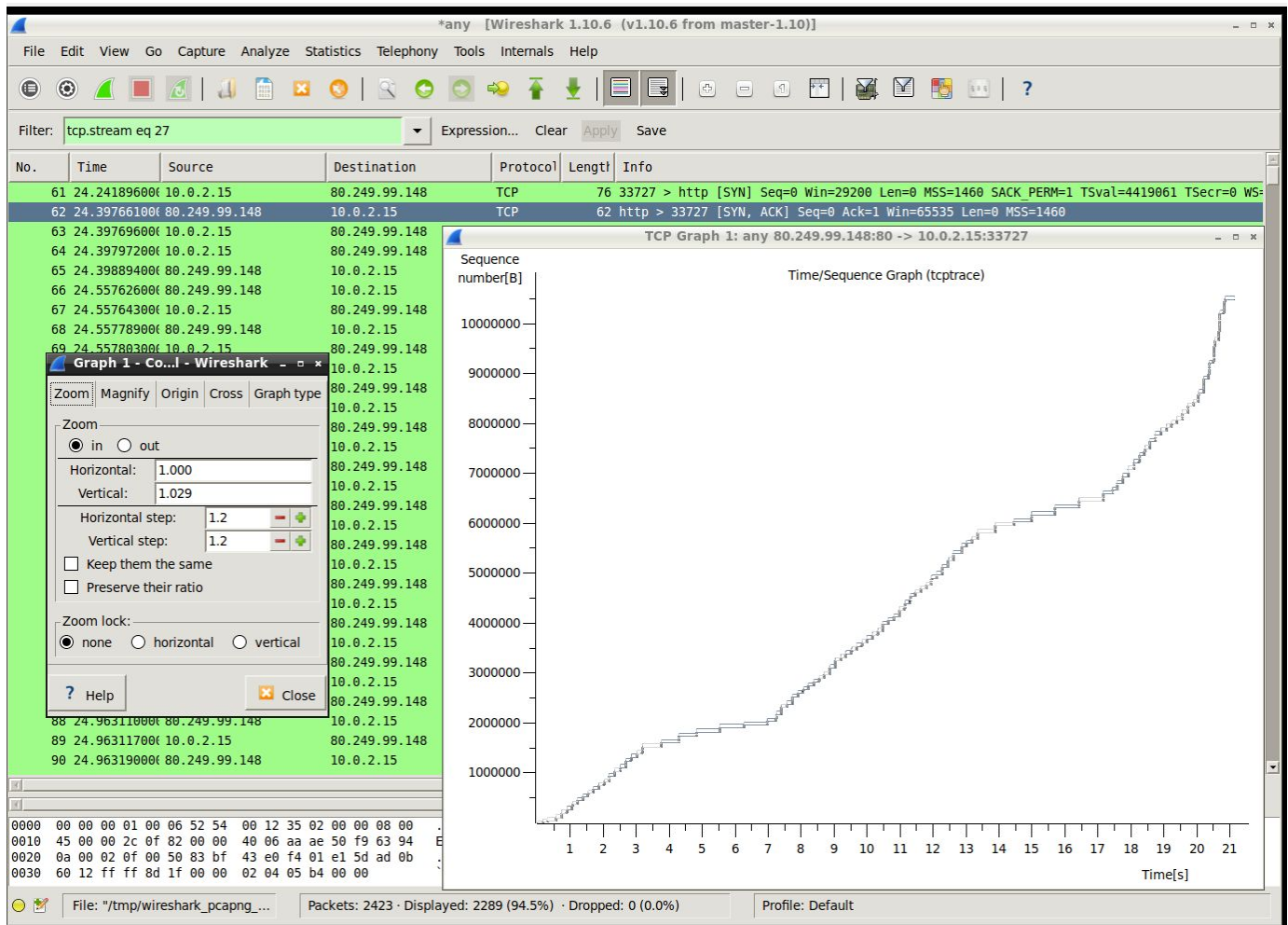
- The initial window size is 29200

c. What was the initial window size that the server advertised to you?

- The initial window size is 65535

12. [10 pts] Find a packet from the download with a source of the server and the destination of your computer.

- Create a tcptrace graph with this packet selected. Take a screenshot of the graph and explain what it is showing. Look into the Wireshark documentation if you need assistance making this graph.



Explain: This graph shows the Bytes of the file downloaded over time, and essentially the total time it took for the download to take place.

In the next section, we will be simulating loss, the command `tc qdisc` will be needed. When you first use the command you should use `add dev` for the device you plan on changing. It only needs to be set on the sender's side. After adding the device use `change dev`. Read through the following paragraph before starting the next step.

Open 2 terminals and have the commands typed and ready before you begin.

- In one terminal, download the 10MB.zip file again. While the download is in progress, change loss to 100%. After a few seconds, change loss to 0%.

Commands:

Terminal 1:

➤ `wget http://ipv4.download.thinkbroadband.com/10MB.zip`

Terminal 2:

➤ `mininet@mininet-vm:~$ sudo tc qdisc add dev eth0 root netem loss`

`100%`

➤ `mininet@mininet-vm:~$ sudo tc qdisc change dev eth0 root netem loss`

`0%`

13. [10 pts] Find a packet from the download with a source of the server and the destination of your computer.

- Create a tcptrace graph with this packet selected. Take a screenshot of the graph and explain what it is showing.
- Using an image editing program, circle the areas where the 0% loss is shown, as well as where TCP is in slow-start and congestion-avoidance.

When changed to 100% loss the pink circle shows where the TCP is in slow start and congestion avoidance. The smaller dark circle shows 0% loss until we change the congestion back to 0% loss where we see a recovery in TCP traffic shown in the larger dark circle.

