Lab 3 - DNS and TCP

Total Points: 100

Platform: This entire lab is to be run on the Mininet VM. The lab computers are available for all of these problems.

Before getting started: We highly recommend reading Section 2.4: Domain Name Service (DNS) in the textbook before beginning this assignment.

Submission:

Important Announcement about Gradescope submissions: When submissions are made on Gradescope, the page/region for each problem MUST be marked. We will not grade submissions that are not marked. If you do not know how to do this, ask for help. If a submission does not have the region/page marked, you will be asked to resubmit. The recorded date for the assignment will be the resubmission date and late days will be counted.

One PDF file for this assignment is to be submitted directly on Gradescope. Naming convention of the file: [YourCruzID].pdf.

Screenshots:

For all questions that require a screenshot, make sure that a **date timestamp** is visible next to your results. No credit will be given for screenshots without a timestamp. **Make sure to highlight as necessary in the screenshots to get full credit.**

References: Chapter 2, Computer Networking: A Top Down Approach

- Section 2.2: The Web and HTTP
- Section 2.4: Domain Name Service (DNS)
- Chapter 3: Transport Layer
- RFC 2616: Hypertext Transfer Protocol -- HTTP/1.1
- https://www.w3.org/Protocols/rfc2616/rfc2616-sec5.html
- https://www.ietf.org/rfc/rfc1035.txt

[50 pts] Domain Name Service

In this lab, we'll make extensive use of the command line tool ***`dig`***. In its most basic operation, the `dig` tool allows the host running the tool to query any Nameserver for a DNS record. The queried DNS server can be a root DNS server, a top-level-domain DNS server, an authoritative DNS server, or an intermediate DNS server (see the textbook for definitions of these terms).

Example commands (Read man page for more information):

```
dig [domain name]  [record type:optional]
dig www.google.com
dig www.google.com AAAA
```

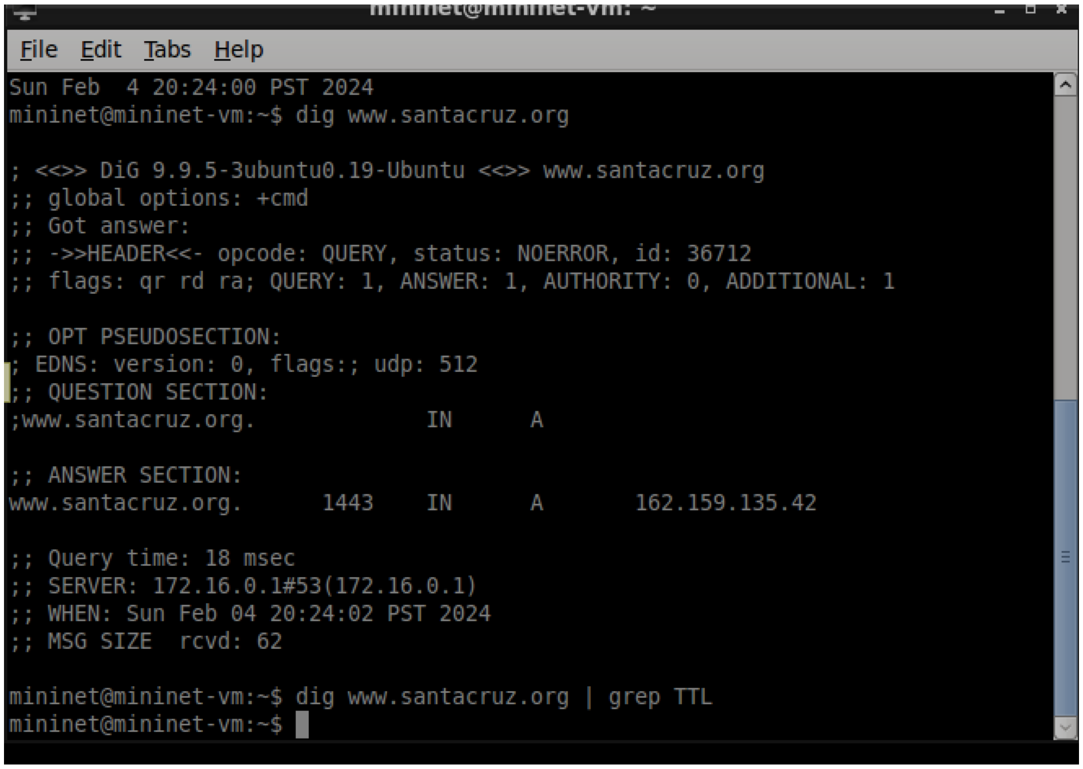
[35 pts] DNS Warmups - Resource Records:

1. Think about the role of the **Resource Records** (RR) in the **DNS** protocol and the role of a **web object** in the **HTTP** protocol.
   What **similarities** do they share in their respective protocols?
   Web object and RR both map hostnames however web objects have URLs and RR map to IP addresses, they both carry info about what their protocol is doing, and they both get cached for faster access

2. Run the command **dig www.santacruz.org**
   a. What **type** of Resource Record(s) are returned and **what is their Functin?**
      By default (according to the man page in the mininet vm, it will run an A Querry. This will return the address record and domain name

   b. What is the purpose of the **Time to Live (TTL)** field? To reflect the changes of the of the RR's since the last time you cached, and just caching data in general
      What is the **TTL** value of the returned Resource Record(s)? 18msec

      Take a **timestamped screenshot** and highlight your answers **for 2(a) and 2(b)**



```
Sun Feb  4 20:24:00 PST 2024
mininet@mininet-vm:~$ dig www.santacruz.org

; <<>> DiG 9.9.5-3ubuntu0.19-Ubuntu <<>> www.santacruz.org
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 36712
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;www.santacruz.org.             IN      A

;; ANSWER SECTION:
www.santacruz.org.      1443    IN      A       162.159.135.42

;; Query time: 18 msec
;; SERVER: 172.16.0.1#53(172.16.0.1)
;; WHEN: Sun Feb 04 20:24:02 PST 2024
;; MSG SIZE  rcvd: 62

mininet@mininet-vm:~$ dig www.santacruz.org | grep TTL
mininet@mininet-vm:~$
```

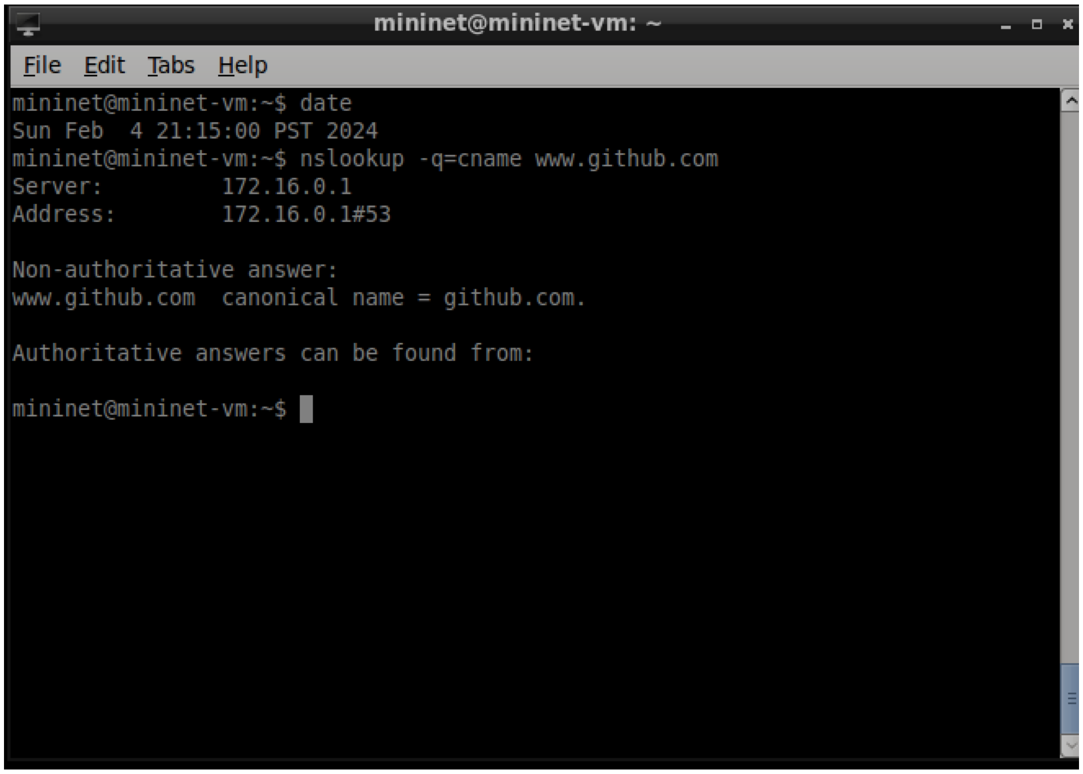
3. **CNAME** records are used to map domains to aliases.
   a. Why are domain **aliases** used on the Internet?

Because it makes owning a domain much easier because no matter which TLD someone puts in you can reroute them to your website without reloading.

   b. Run a command to find the **CNAME** record for **www.github.com**. Which name is the ***alias***? Which name is the **canonical** name?

Www.google.com is the alias name and github.com is the canonical name

Take a **timestamped screenshot** of the command and output – highlight the **alias** and **canonical** name to support your answer.



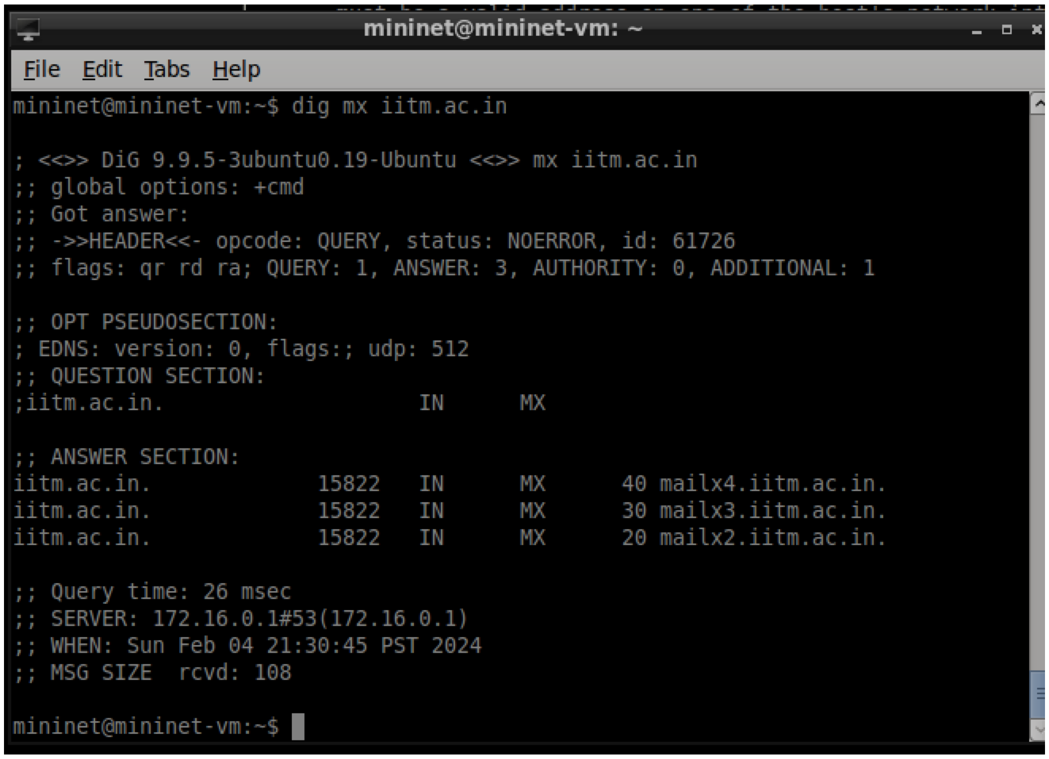
```
mininet@mininet-vm:~$ date
Sun Feb  4 21:15:00 PST 2024
mininet@mininet-vm:~$ nslookup -q=cname www.github.com
Server:         172.16.0.1
Address:        172.16.0.1#53

Non-authoritative answer:
www.github.com  canonical name = github.com.

Authoritative answers can be found from:

mininet@mininet-vm:~$ 
```

4. Run **`dig`** to find the ***MX*** resource record of **iitm.ac.in**
    a. What information does an **MX** resource record provide?
    An MX or Mail Exchange record lets you know what mail server is chosen for that domain

    b. What **command** did you use to obtain the **MX** resource records for the given domain? Dig mx **iitm.ac.in**

    c. Based on the output of the MX query, which mail server do you think your computer would contact when sending an email to someone@iitm.ac.in? Explain your answer. Mailx4.iitm.ac.in as that seems to be the first one responding that it is ready to receive.

Take a **timestamped screenshot** of the command and output – answers in (b) and (c) to support your answer.



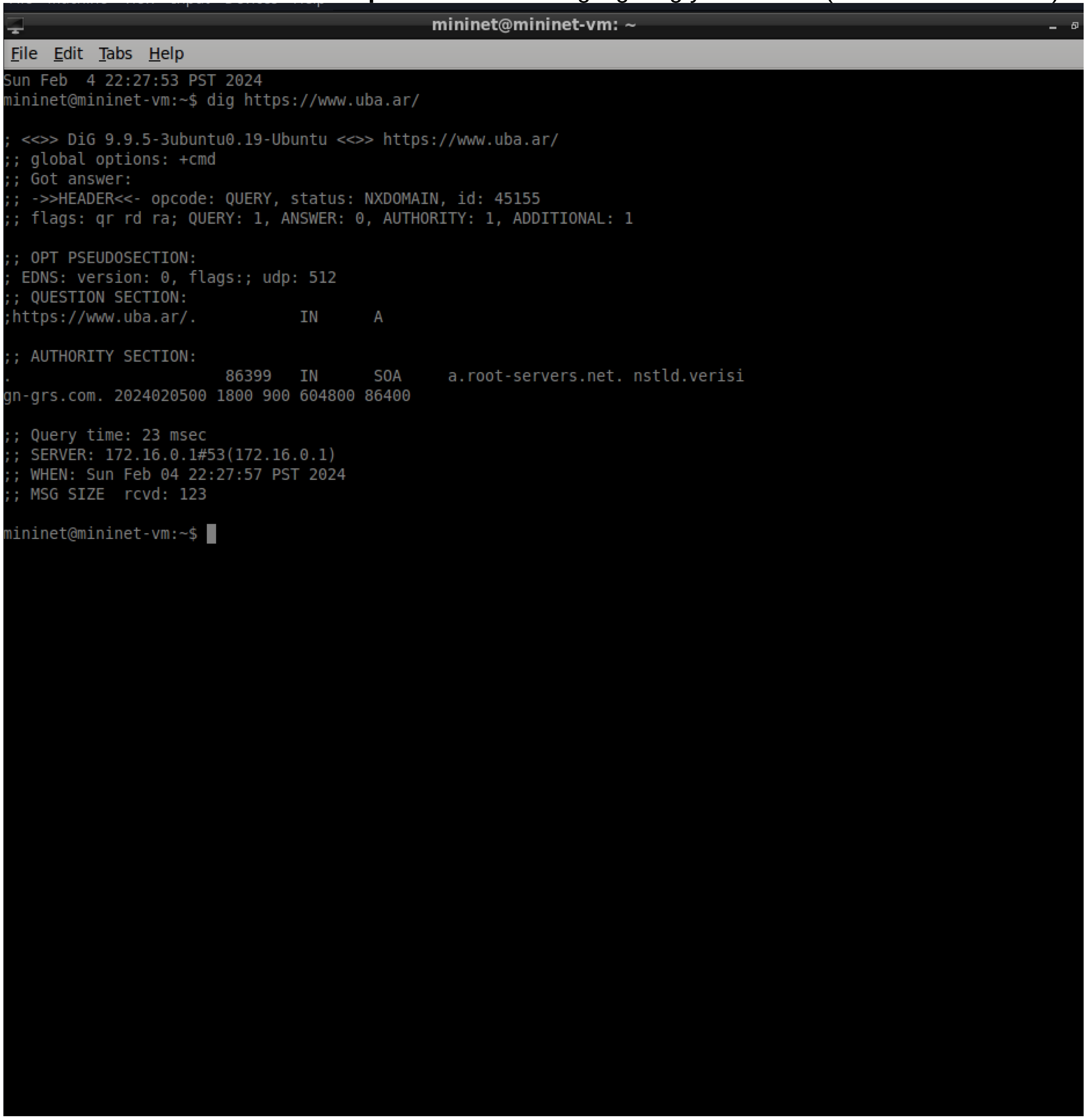
```
mininet@mininet-vm:~$ dig mx iitm.ac.in

; <<>> DiG 9.9.5-3ubuntu0.19-Ubuntu <<>> mx iitm.ac.in
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 61726
;; flags: qr rd ra; QUERY: 1, ANSWER: 3, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;iitm.ac.in.                    IN      MX

;; ANSWER SECTION:
iitm.ac.in.             15822   IN      MX      40 mailx4.iitm.ac.in.
iitm.ac.in.             15822   IN      MX      30 mailx3.iitm.ac.in.
iitm.ac.in.             15822   IN      MX      20 mailx2.iitm.ac.in.

;; Query time: 26 msec
;; SERVER: 172.16.0.1#53(172.16.0.1)
;; WHEN: Sun Feb 04 21:30:45 PST 2024
;; MSG SIZE  rcvd: 108

mininet@mininet-vm:~$
```

d. Think about accessing the domain in (a) in your browser compared to sending an email to a person with an email account in the domain (e.g., www.ucsc.edu in your browser versus sammyslug@ucsc.edu).
What corresponding **DNS** queries are made, how are they **different**?
When accessing the website you only use A or AAAA records but mail also uses an MX on top of that, also websites querry static content meanwhile email uses dynamic messages which means that continuous queries happen while the email is being sent rather then just at the beginning like a website.
What **DNS** mechanisms/services are being used in this example?
Records, queries, caching,

5. **Authoritative** name servers
   a. What is an Authoritative name server and why is it **needed**?
   It is a source of data that has ip addresses for domains. It is run by an official source that keeps people from adding fake ip addresses to domains in order to trick you. It is needed for a multitude of reasons, one being that remembering the IP address to everything is too much for most people, second IP addresses don't always stay the same, so this allows you to just put in the web address and it will point you to the correct IP. Lastly it makes it much harder to spoof domains.

   b. Run **`dig`** to determine the authoritative **DNS** servers for a university in **South America.** What is the name of the university you chose?
   Universidad de Buenos Aires

Take a **timestamped screenshot** highlighting your results (the Authoritative NS).



```
Sun Feb  4 22:27:53 PST 2024
mininet@mininet-vm:~$ dig https://www.uba.ar/

; <<>> DiG 9.9.5-3ubuntu0.19-Ubuntu <<>> https://www.uba.ar/
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN, id: 45155
;; flags: qr rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 1, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;https://www.uba.ar/.           IN      A

;; AUTHORITY SECTION:
.                       86399   IN      SOA     a.root-servers.net. nstld.verisi
gn-grs.com. 2024020500 1800 900 604800 86400

;; Query time: 23 msec
;; SERVER: 172.16.0.1#53(172.16.0.1)
;; WHEN: Sun Feb 04 22:27:57 PST 2024
;; MSG SIZE  rcvd: 123

mininet@mininet-vm:~$ 
```

c. Suppose the university wanted to further partition the **DNS** namespace. For example, the **engineering** and **biology** departments want to be in charge of their own **namespace** (e.g.,names of computers in their respective departments). How could **Authoritative nameservers** be used to accomplish this task? You could easily do this with subdomains, you would just tell the authoritative nameserver the new IP of the subdomain (being engineering.soe.uba.ar).

6. Run 'dig x.com ANY'

Make sure you are doing this from the ucsc domain or 'ANY' may not work as expected.

a) What is the purpose of ANY?
It returns any and all responses it gets when it sends out every check

b) Why are multiple A records returned? Explain and discuss the additional service DNS is performing by providing these multiple records and explain its importance.
It seems that they have multiple Ips, this could be because they load balance or because they have multiple domain aliases, This would help with popular websites or if a website has multiple servers in different parts of the world and they want to send you to one thats closer to you.

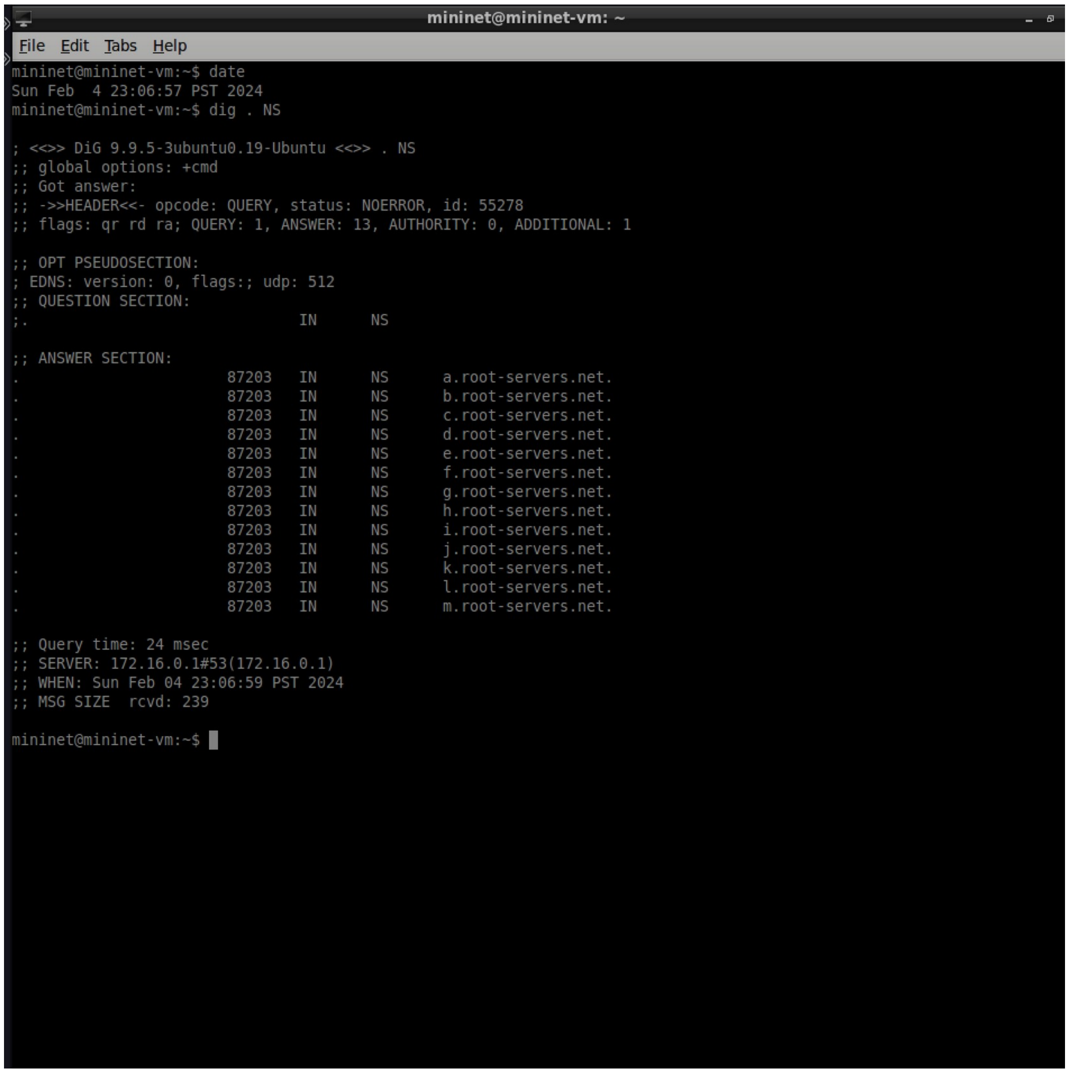
7. Open Wireshark and listen on the `‘any’` interface. Open a terminal, and clear your dns cache using the command `systemd-resolve --flush-caches` (new VM) or `sudo /etc/init.d/dns-clean restart` (old VM).
   Then, use `dig` to perform a DNS lookup to kicker.de **two consecutive times**.

   a) Does the DNS query take the same amount of time to run both times?
   No it doesn’t take very long for the second one compared to the first which took significantly longer

   b) With respect to what you know about DNS, provide an explanation for the difference in lookup time - i.e. what is different between the two queries? Most likely the second one had a bunch of data cached already so it could easiy find the info without asking the Authoratative name server. Which would make sense as you had us clear our cache earlier, the first needed to pull all new data and then cached it, the second however already had it all.

8. Root Name Servers
   a. What are root name servers?
   They are the servers that hold all records for .com, .org, .net, etc
   b. What command is used to find the root name servers?
   Dig . NS

c. Run the command and include a **timestamped screenshot of** the results and discuss what you see.



```
mininet@mininet-vm:~$ date
Sun Feb  4 23:06:57 PST 2024
mininet@mininet-vm:~$ dig . NS

; <<>> DiG 9.9.5-3ubuntu0.19-Ubuntu <<>> . NS
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 55278
;; flags: qr rd ra; QUERY: 1, ANSWER: 13, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;.                              IN      NS

;; ANSWER SECTION:
.                       87203   IN      NS      a.root-servers.net.
.                       87203   IN      NS      b.root-servers.net.
.                       87203   IN      NS      c.root-servers.net.
.                       87203   IN      NS      d.root-servers.net.
.                       87203   IN      NS      e.root-servers.net.
.                       87203   IN      NS      f.root-servers.net.
.                       87203   IN      NS      g.root-servers.net.
.                       87203   IN      NS      h.root-servers.net.
.                       87203   IN      NS      i.root-servers.net.
.                       87203   IN      NS      j.root-servers.net.
.                       87203   IN      NS      k.root-servers.net.
.                       87203   IN      NS      l.root-servers.net.
.                       87203   IN      NS      m.root-servers.net.

;; Query time: 24 msec
;; SERVER: 172.16.0.1#53(172.16.0.1)
;; WHEN: Sun Feb 04 23:06:59 PST 2024
;; MSG SIZE  rcvd: 239

mininet@mininet-vm:~$
```

I see 13 different name servers, it also seems like it only needed to querry once to get all the data on the name servers, and doing so took a very short amount of time.

d. Is your access network physically close to a root name server? Explain how you have come to your conclusion.Yes, I tracerouted to one of them and because of both low ping time and few hops, I think I am

9. Treasure hunt! di:
What command would you use to run a reverse DNS lookup on IP Address 171.67.215.200? What domain is associated with the address 171.67.215.200?
Dig -x, web.stanford.edu

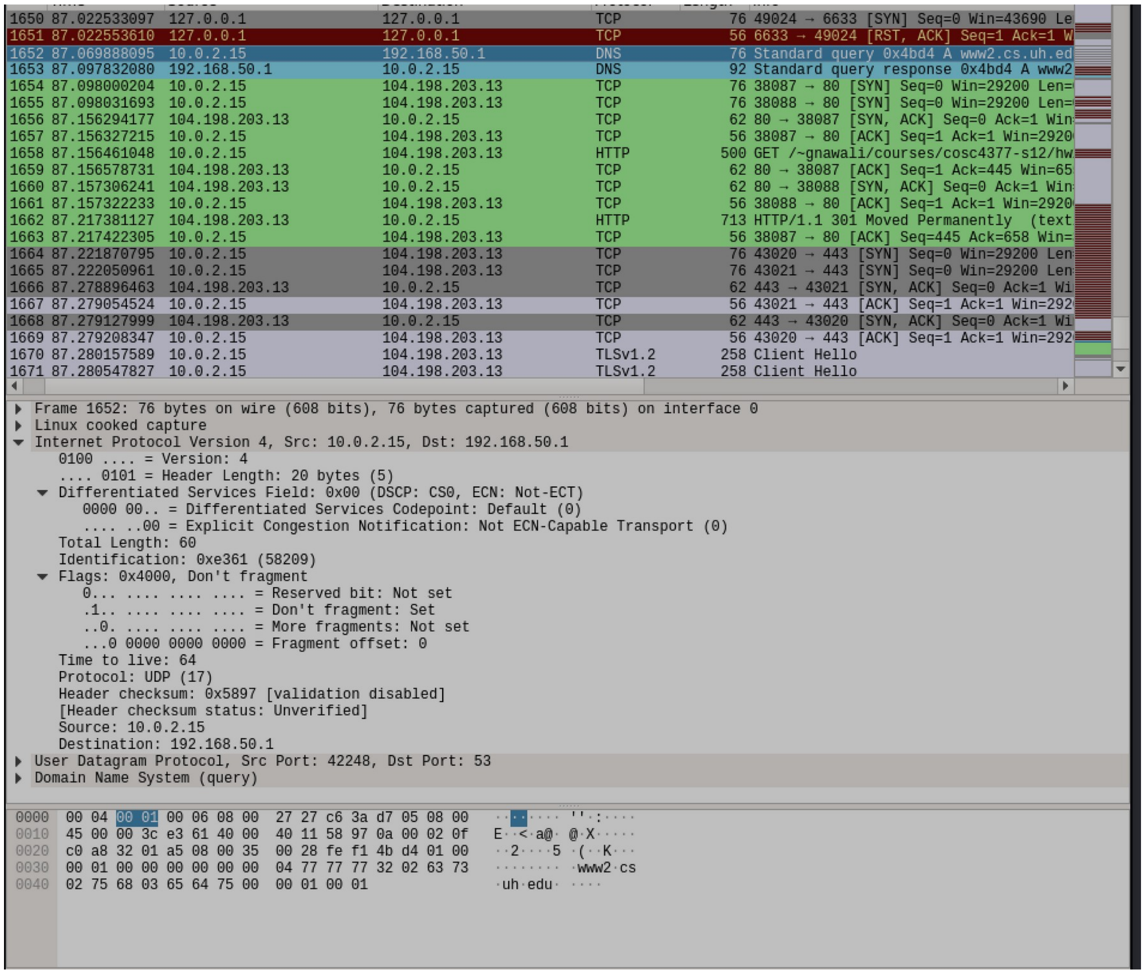
[15 pts] Digging in with Wireshark - the DNS Messages

For the next two questions, we will observe how the DNS protocol operates at the packet level by using the Mininet VM and Wireshark.
Begin by doing the following:

- Open Wireshark and listen on the 'any' interface without any filter
- Open Chromium, clear your web cache and navigate to

http://www2.cs.uh.edu/~gnawali/courses/cosc4377-s12/hw2/http.html

10. **Wireshark capture of DNS Messages:**
Show a **timestamped screenshot** in Wireshark and highlight the areas in your answers to the questions below:



```
1650 87.022533097 127.0.0.1       127.0.0.1       TCP       76 49024 → 6633 [SYN] Seq=0 Win=43690 Le
1651 87.022553610 127.0.0.1       127.0.0.1       TCP       56 6633 → 49024 [RST, ACK] Seq=1 Ack=1 W
1652 87.069888095 10.0.2.15       192.168.50.1    DNS       76 Standard query 0x4bd4 A www2.cs.uh.ed
1653 87.097832080 192.168.50.1    10.0.2.15       DNS       92 Standard query response 0x4bd4 A www2
1654 87.098000204 10.0.2.15       104.198.203.13  TCP       76 38087 → 80 [SYN] Seq=0 Win=29200 Len=
1655 87.098031693 10.0.2.15       104.198.203.13  TCP       76 38088 → 80 [SYN] Seq=0 Win=29200 Len=
1656 87.156294177 104.198.203.13  10.0.2.15       TCP       62 80 → 38087 [SYN, ACK] Seq=0 Ack=1 Win
1657 87.156327215 10.0.2.15       104.198.203.13  TCP       56 38087 → 80 [ACK] Seq=1 Ack=1 Win=2920
1658 87.156461048 10.0.2.15       104.198.203.13  HTTP     500 GET /~gnawali/courses/cosc4377-s12/hw
1659 87.156578731 104.198.203.13  10.0.2.15       TCP       62 80 → 38087 [ACK] Seq=1 Ack=445 Win=65
1660 87.157306241 104.198.203.13  10.0.2.15       TCP       62 80 → 38088 [SYN, ACK] Seq=0 Ack=1 Win
1661 87.157322233 10.0.2.15       104.198.203.13  TCP       56 38088 → 80 [ACK] Seq=1 Ack=1 Win=2920
1662 87.217381127 104.198.203.13  10.0.2.15       HTTP     713 HTTP/1.1 301 Moved Permanently  (text
1663 87.217422305 10.0.2.15       104.198.203.13  TCP       56 38087 → 80 [ACK] Seq=445 Ack=658 Win=
1664 87.221870795 10.0.2.15       104.198.203.13  TCP       76 43020 → 443 [SYN] Seq=0 Win=29200 Len
1665 87.222050961 10.0.2.15       104.198.203.13  TCP       76 43021 → 443 [SYN] Seq=0 Win=29200 Len
1666 87.278896463 104.198.203.13  10.0.2.15       TCP       62 443 → 43021 [SYN, ACK] Seq=0 Ack=1 Wi
1667 87.279054524 10.0.2.15       104.198.203.13  TCP       56 43021 → 443 [ACK] Seq=1 Ack=1 Win=292
1668 87.279127999 104.198.203.13  10.0.2.15       TCP       62 443 → 43020 [SYN, ACK] Seq=0 Ack=1 Wi
1669 87.279208347 10.0.2.15       104.198.203.13  TCP       56 43020 → 443 [ACK] Seq=1 Ack=1 Win=292
1670 87.280157589 10.0.2.15       104.198.203.13  TLSv1.2  258 Client Hello
1671 87.280547827 10.0.2.15       104.198.203.13  TLSv1.2  258 Client Hello

▶ Frame 1652: 76 bytes on wire (608 bits), 76 bytes captured (608 bits) on interface 0
▶ Linux cooked capture
▼ Internet Protocol Version 4, Src: 10.0.2.15, Dst: 192.168.50.1
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  ▼ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
      0000 00.. = Differentiated Services Codepoint: Default (0)
      .... ..00 = Explicit Congestion Notification: Not ECN-Capable Transport (0)
    Total Length: 60
    Identification: 0xe361 (58209)
  ▼ Flags: 0x4000, Don't fragment
      0... .... .... .... = Reserved bit: Not set
      .1.. .... .... .... = Don't fragment: Set
      ..0. .... .... .... = More fragments: Not set
      ...0 0000 0000 0000 = Fragment offset: 0
    Time to live: 64
    Protocol: UDP (17)
    Header checksum: 0x5897 [validation disabled]
    [Header checksum status: Unverified]
    Source: 10.0.2.15
    Destination: 192.168.50.1
▶ User Datagram Protocol, Src Port: 42248, Dst Port: 53
▶ Domain Name System (query)

0000  00 04 00 01 00 06 08 00  27 27 c6 3a d7 05 08 00   ········ '':·····
0010  45 00 00 3c e3 61 40 00  40 11 58 97 0a 00 02 0f   E··<·a@· @·X·····
0020  c0 a8 32 01 a5 08 00 35  00 28 fe f1 4b d4 01 00   ··2····5 ·(··K···
0030  00 01 00 00 00 00 00 00  04 77 77 77 32 02 63 73   ········ ·www2·cs
0040  02 75 68 03 65 64 75 00  00 01 00 01               ·uh·edu· ····
```

a. Find the DNS query and response messages. Are these messages sent with UDP or TCP? Circle the transport layer protocol in your screenshot.

UDP

b. Discuss the transport layer protocol observed in (a) - why do you think the one you observed is used? Highlight in your screenshot. It most likely uses it because it doesn't need to keep a stable connection like tcp and because it can send multiple at a time its safer.

c. What is the IP address of the local nameserver contacted by your client? Show in the screenshot.192.168.50.1

Look for the source and destination ports of the **DNS queries**:

		Well known port? yes/no
Source Port Number	42248	no
Destination Port Number	53	yes

d. Is this destination port number what you expected? Why or why not? Highlight this port in your screenshot. Yes it is the DNS port

e. If you were to close your browser and then access the site again, do you expect your source port number to stay the same? Explain your answer. Highlight the port number in your screenshot.
Probably not because chrome is just finding a random port.

f. Examine the DNS response message. How many "answers" are provided? What information is provided in each of these "answers"? Do you see a final answer or only referrals? What is the "final answer"? i.e. what is the IP address returned by DNS? 104.198.203.13

11. **Continuation of Q10** – **Loading the webpage and Wireshark capture:**

a. After DNS is finally resolved, we expect that a TCP connection should be opened <u>before</u> the HTTP request is sent. <u>Look at a TCP packet and determine the source and destination addresses and ports.</u> Fill in the table below.

IP Addresses:		
Source IP Address	10.0.2.15	
Destination IP Address	104.198.203.13	
Ports:		
		Well known port? yes/no
Source Port Number	38087	No
Destination Port Number	80	yes

b. <u>Web Server IP Address</u>: Referring to the table above, Is the IP Address of the web server the same as what was returned in Q10(f)? Explain why or why not.
Yes, because the web server must stay the same to download the file

c. Why is the destination IP address for the HTTP request different from the destination IP address used by DNS queries? Because of DNS resolution

d. What kind of file is downloaded from the web server? How do you know?
A .zip file, because when I ls the file it says .zip

[50 pts] Transport Control Protocol - TCP

In this section we look at a few different ways to transfer files and observe the transfers over Wireshark. We will give an express TCP overview in class, but we won't officially reach this topic until we study the Transport Layer in Chapter 3 (a good reference if you have questions).

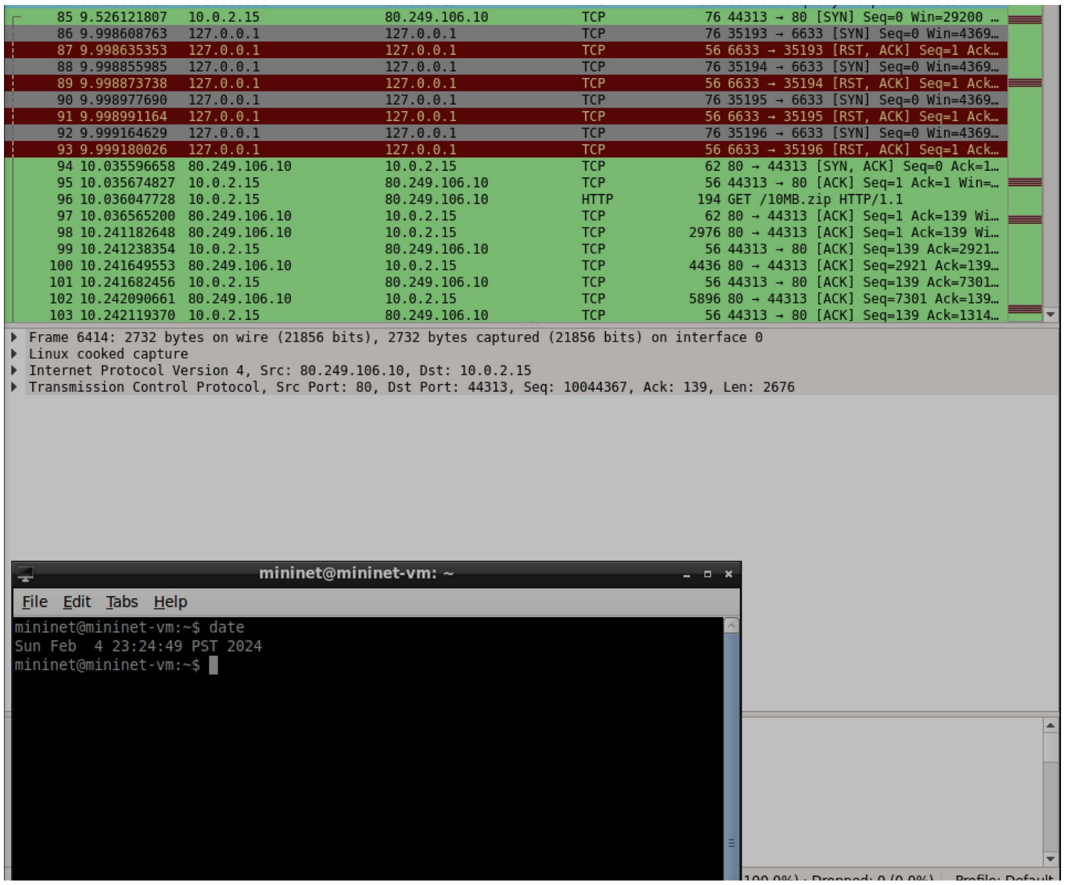
12. File Transfer using **wget** from an origin server

Open Wireshark and listen on the 'any' interface. Then using **`wget,`** download http://ipv4.download.thinkbroadband.com/10MB.zip in a terminal window.

a. Enumerate and describe all of the different protocols you see in your Wireshark capture from the time you enter the URL leading up to and including the file download (Ignore NTP packets).

- DNS
- Tcp
- http

b. Find the **TCP three-way handshake** for connection establishment. Take a **<u>timestamped screenshot</u>** of these TCP segments and highlight all of them in the screenshot.



```
 85 9.526121807   10.0.2.15       80.249.106.10   TCP     76 44313 → 80 [SYN] Seq=0 Win=29200 …
 86 9.998608763   127.0.0.1       127.0.0.1       TCP     76 35193 → 6633 [SYN] Seq=0 Win=4369…
 87 9.998635353   127.0.0.1       127.0.0.1       TCP     56 6633 → 35193 [RST, ACK] Seq=1 Ack…
 88 9.998855985   127.0.0.1       127.0.0.1       TCP     76 35194 → 6633 [SYN] Seq=0 Win=4369…
 89 9.998873738   127.0.0.1       127.0.0.1       TCP     56 6633 → 35194 [RST, ACK] Seq=1 Ack…
 90 9.998977690   127.0.0.1       127.0.0.1       TCP     76 35195 → 6633 [SYN] Seq=0 Win=4369…
 91 9.998991164   127.0.0.1       127.0.0.1       TCP     56 6633 → 35195 [RST, ACK] Seq=1 Ack…
 92 9.999164629   127.0.0.1       127.0.0.1       TCP     76 35196 → 6633 [SYN] Seq=0 Win=4369…
 93 9.999180026   127.0.0.1       127.0.0.1       TCP     56 6633 → 35196 [RST, ACK] Seq=1 Ack…
 94 10.035596658  80.249.106.10   10.0.2.15       TCP     62 80 → 44313 [SYN, ACK] Seq=0 Ack=1…
 95 10.035674827  10.0.2.15       80.249.106.10   TCP     56 44313 → 80 [ACK] Seq=1 Ack=1 Win=…
 96 10.036047728  10.0.2.15       80.249.106.10   HTTP   194 GET /10MB.zip HTTP/1.1
 97 10.036565200  80.249.106.10   10.0.2.15       TCP     62 80 → 44313 [ACK] Seq=1 Ack=139 Wi…
 98 10.241182648  80.249.106.10   10.0.2.15       TCP   2976 80 → 44313 [ACK] Seq=1 Ack=139 Wi…
 99 10.241238354  10.0.2.15       80.249.106.10   TCP     56 44313 → 80 [ACK] Seq=139 Ack=2921…
100 10.241649553  80.249.106.10   10.0.2.15       TCP   4436 80 → 44313 [ACK] Seq=2921 Ack=139…
101 10.241682456  10.0.2.15       80.249.106.10   TCP     56 44313 → 80 [ACK] Seq=139 Ack=7301…
102 10.242090661  80.249.106.10   10.0.2.15       TCP   5896 80 → 44313 [ACK] Seq=7301 Ack=139…
103 10.242119370  10.0.2.15       80.249.106.10   TCP     56 44313 → 80 [ACK] Seq=139 Ack=1314…

▸ Frame 6414: 2732 bytes on wire (21856 bits), 2732 bytes captured (21856 bits) on interface 0
▸ Linux cooked capture
▸ Internet Protocol Version 4, Src: 80.249.106.10, Dst: 10.0.2.15
▸ Transmission Control Protocol, Src Port: 80, Dst Port: 44313, Seq: 10044367, Ack: 139, Len: 2676
```

```
mininet@mininet-vm:~$ date
Sun Feb  4 23:24:49 PST 2024
mininet@mininet-vm:~$
```

c. **TCP Segments:** After the handshake is complete, find two TCP segments (packets) – one indicating the start of the file download and then the final packet in the transfer.

- How many different TCP segments are transmitted to send the entire file? How did you discover this in Wireshark?241, in wiresharks statistics you can access the conversations tools and see unique conversations

- What is the typical segment size? 14,656
- What value do you expect if you multiply the number segments transmitted by the typical number of bytes in a segment? Try it! Show your work and explain.Multiply the number of segments transmitted by the typical segment size to find the total amount of data transferred. 241*14656=3532096

d. **File Distribution Time according to Wireshark:** Using the two packets in (c), calculate the download time using the Time column in Wireshark.
3.27

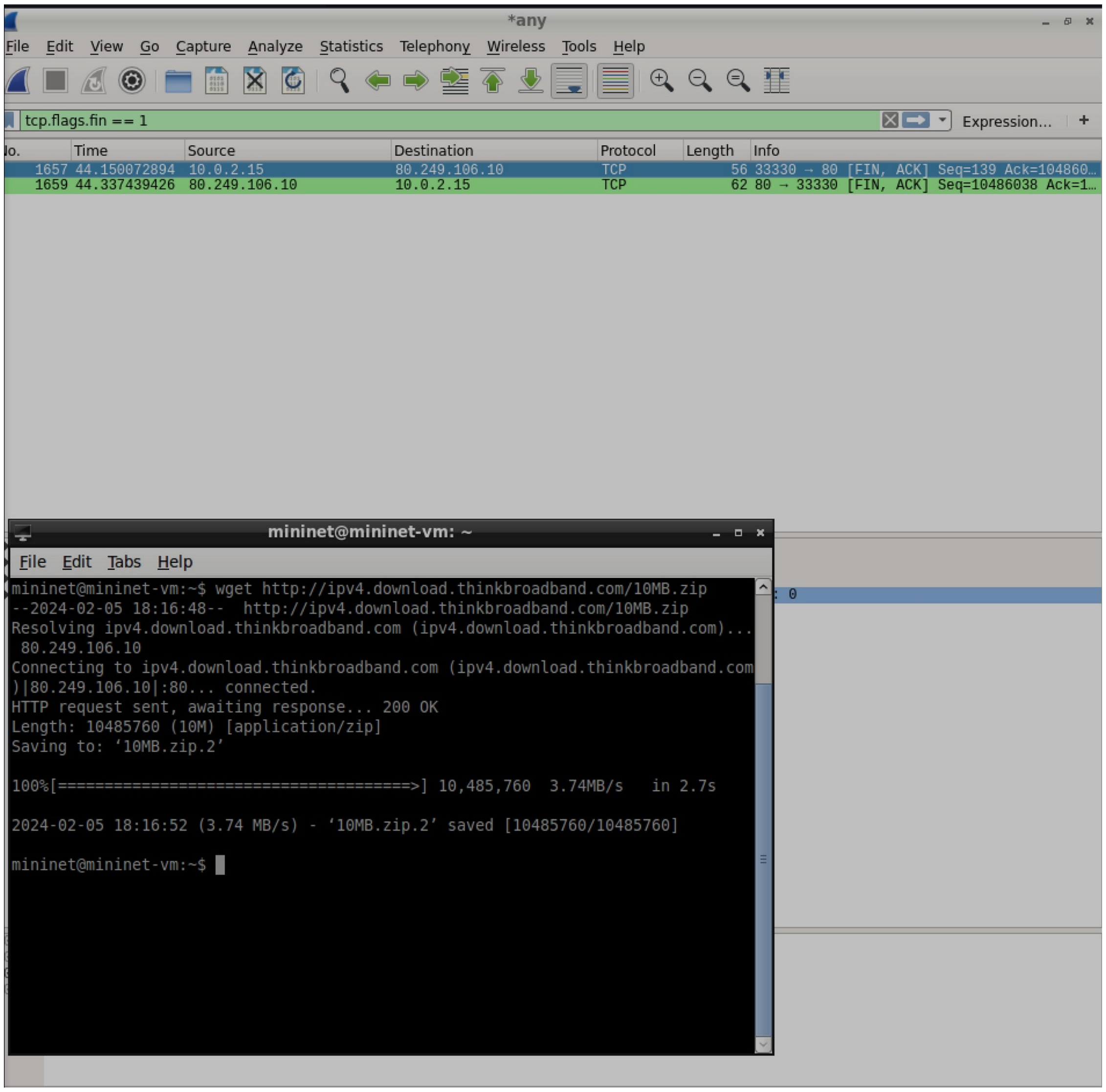
e. **Throughput according to Wireshark:** Using (d), calculate the average throughput seen by the Client and enter it in the table below. Show all of your work. Enter the throughput in the table below. 14,656/3.27=4,482

f. Enter the throughput as reported in the wget terminal in the table below. How does the value compare to the throughput calculation in (e)? Is it close? Discuss and account for any differences.

Application	Throughput	Notes (if any)
Wireshark capture of wget	4482	
Terminal output of wget	3740	

<u>Include two timestamped screenshots:</u>

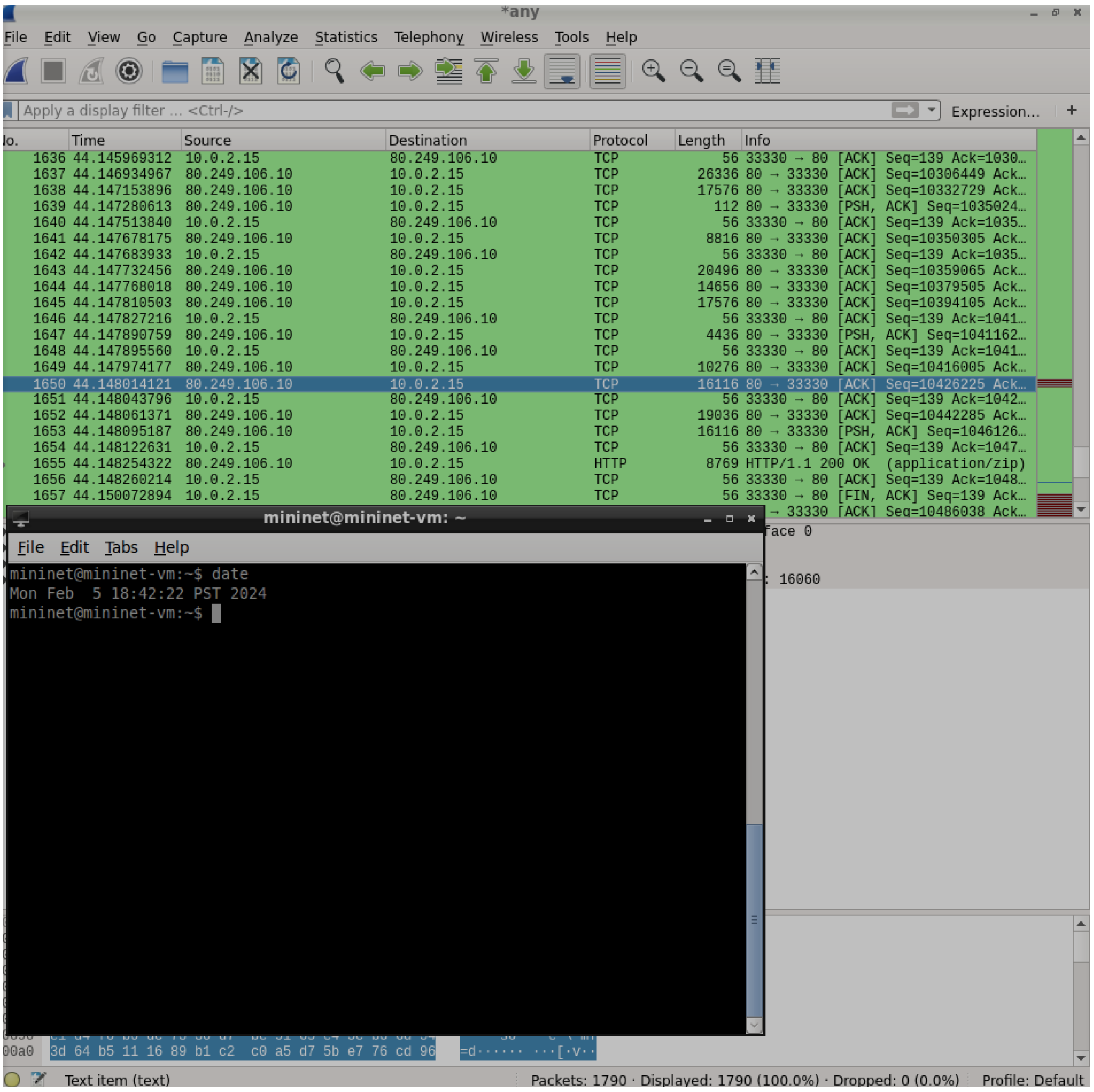
- Wireshark output highlighting packets and values used to calculate (d) and (e)
- Terminal output of `wget` highlighting the reported download speed



```
tcp.flags.fin == 1
```

```
No.   Time          Source          Destination     Protocol  Length Info
 1657 44.150072894  10.0.2.15       80.249.106.10   TCP       56 33330 → 80 [FIN, ACK] Seq=139 Ack=104860…
 1659 44.337439426  80.249.106.10   10.0.2.15       TCP       62 80 → 33330 [FIN, ACK] Seq=10486038 Ack=1…
```

```
mininet@mininet-vm:~$ wget http://ipv4.download.thinkbroadband.com/10MB.zip
--2024-02-05 18:16:48--  http://ipv4.download.thinkbroadband.com/10MB.zip
Resolving ipv4.download.thinkbroadband.com (ipv4.download.thinkbroadband.com)...
 80.249.106.10
Connecting to ipv4.download.thinkbroadband.com (ipv4.download.thinkbroadband.com
)|80.249.106.10|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 10485760 (10M) [application/zip]
Saving to: ‘10MB.zip.2’

100%[=====================================>] 10,485,760  3.74MB/s   in 2.7s

2024-02-05 18:16:52 (3.74 MB/s) - ‘10MB.zip.2’ saved [10485760/10485760]

mininet@mininet-vm:~$ 
```



```
No.   Time         Source         Destination    Protocol Length Info
 1636 44.145969312 10.0.2.15      80.249.106.10  TCP         56 33330 → 80 [ACK] Seq=139 Ack=1030…
 1637 44.146934967 80.249.106.10  10.0.2.15      TCP      26336 80 → 33330 [ACK] Seq=10306449 Ack…
 1638 44.147153896 80.249.106.10  10.0.2.15      TCP      17576 80 → 33330 [ACK] Seq=10332729 Ack…
 1639 44.147280613 80.249.106.10  10.0.2.15      TCP        112 80 → 33330 [PSH, ACK] Seq=1035024…
 1640 44.147513840 10.0.2.15      80.249.106.10  TCP         56 33330 → 80 [ACK] Seq=139 Ack=1035…
 1641 44.147678175 80.249.106.10  10.0.2.15      TCP       8816 80 → 33330 [ACK] Seq=10350305 Ack…
 1642 44.147683933 10.0.2.15      80.249.106.10  TCP         56 33330 → 80 [ACK] Seq=139 Ack=1035…
 1643 44.147732456 80.249.106.10  10.0.2.15      TCP      20496 80 → 33330 [ACK] Seq=10359065 Ack…
 1644 44.147768018 80.249.106.10  10.0.2.15      TCP      14656 80 → 33330 [ACK] Seq=10379505 Ack…
 1645 44.147810503 80.249.106.10  10.0.2.15      TCP      17576 80 → 33330 [ACK] Seq=10394105 Ack…
 1646 44.147827216 10.0.2.15      80.249.106.10  TCP         56 33330 → 80 [ACK] Seq=139 Ack=1041…
 1647 44.147890759 80.249.106.10  10.0.2.15      TCP       4436 80 → 33330 [PSH, ACK] Seq=1041162…
 1648 44.147895560 10.0.2.15      80.249.106.10  TCP         56 33330 → 80 [ACK] Seq=139 Ack=1041…
 1649 44.147974177 80.249.106.10  10.0.2.15      TCP      10276 80 → 33330 [ACK] Seq=10416005 Ack…
 1650 44.148014121 80.249.106.10  10.0.2.15      TCP      16116 80 → 33330 [ACK] Seq=10426225 Ack…
 1651 44.148043796 10.0.2.15      80.249.106.10  TCP         56 33330 → 80 [ACK] Seq=139 Ack=1042…
 1652 44.148061371 80.249.106.10  10.0.2.15      TCP      19036 80 → 33330 [ACK] Seq=10442285 Ack…
 1653 44.148095187 80.249.106.10  10.0.2.15      TCP      16116 80 → 33330 [PSH, ACK] Seq=1046126…
 1654 44.148122631 10.0.2.15      80.249.106.10  TCP         56 33330 → 80 [ACK] Seq=139 Ack=1047…
 1655 44.148254322 80.249.106.10  10.0.2.15      HTTP      8769 HTTP/1.1 200 OK  (application/zip)
 1656 44.148260214 10.0.2.15      80.249.106.10  TCP         56 33330 → 80 [ACK] Seq=139 Ack=1048…
 1657 44.150072894 10.0.2.15      80.249.106.10  TCP         56 33330 → 80 [FIN, ACK] Seq=139 Ack…
                                                                 → 33330 [ACK] Seq=10486038 Ack…
```

```
mininet@mininet-vm:~$ date
Mon Feb  5 18:42:22 PST 2024
mininet@mininet-vm:~$ 
```

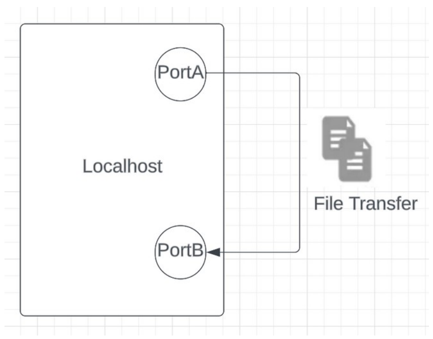
```
face 0
: 16060
00a0  3d 64 b5 11 16 89 b1 c2  c0 a5 d7 5b e7 76 cd 96   =d······ ···[·v··
```

Text item (text)   Packets: 1790 · Displayed: 1790 (100.0%) · Dropped: 0 (0.0%)   Profile: Default

*any
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>
Expression…
mininet@mininet-vm: ~
File Edit Tabs Help

13. File Transfer using netcat locally

In this problem we will use the `netcat` utility to perform a <u>local file transfer</u> of the file 10MB.zip One terminal is set up for listening and the other for sending as shown below:

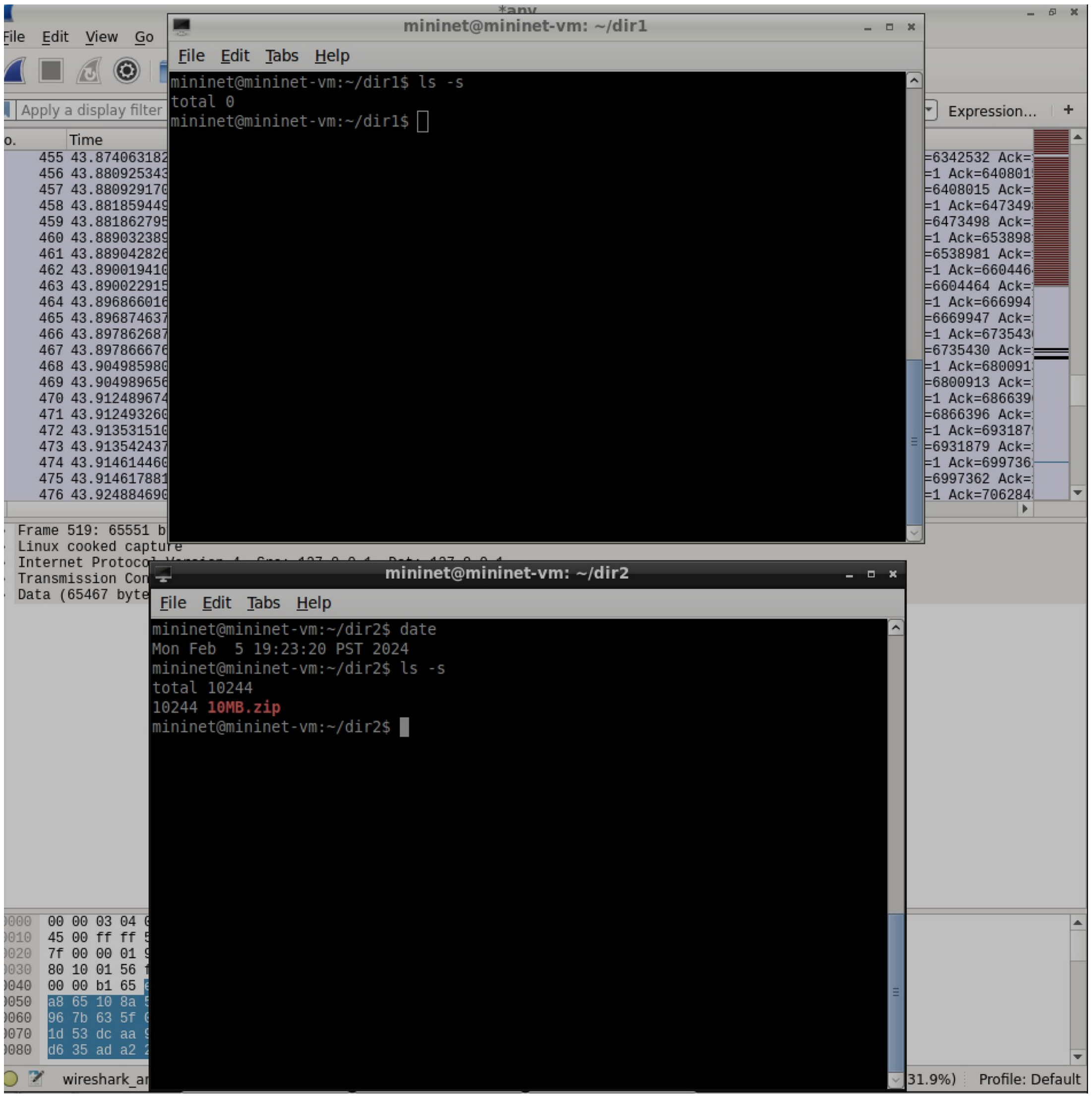


a. Research the `netcat` tool and briefly describe in your own words what it does.

Follow the procedure outlined below to accomplish the transfer of the 10MB.zip file:

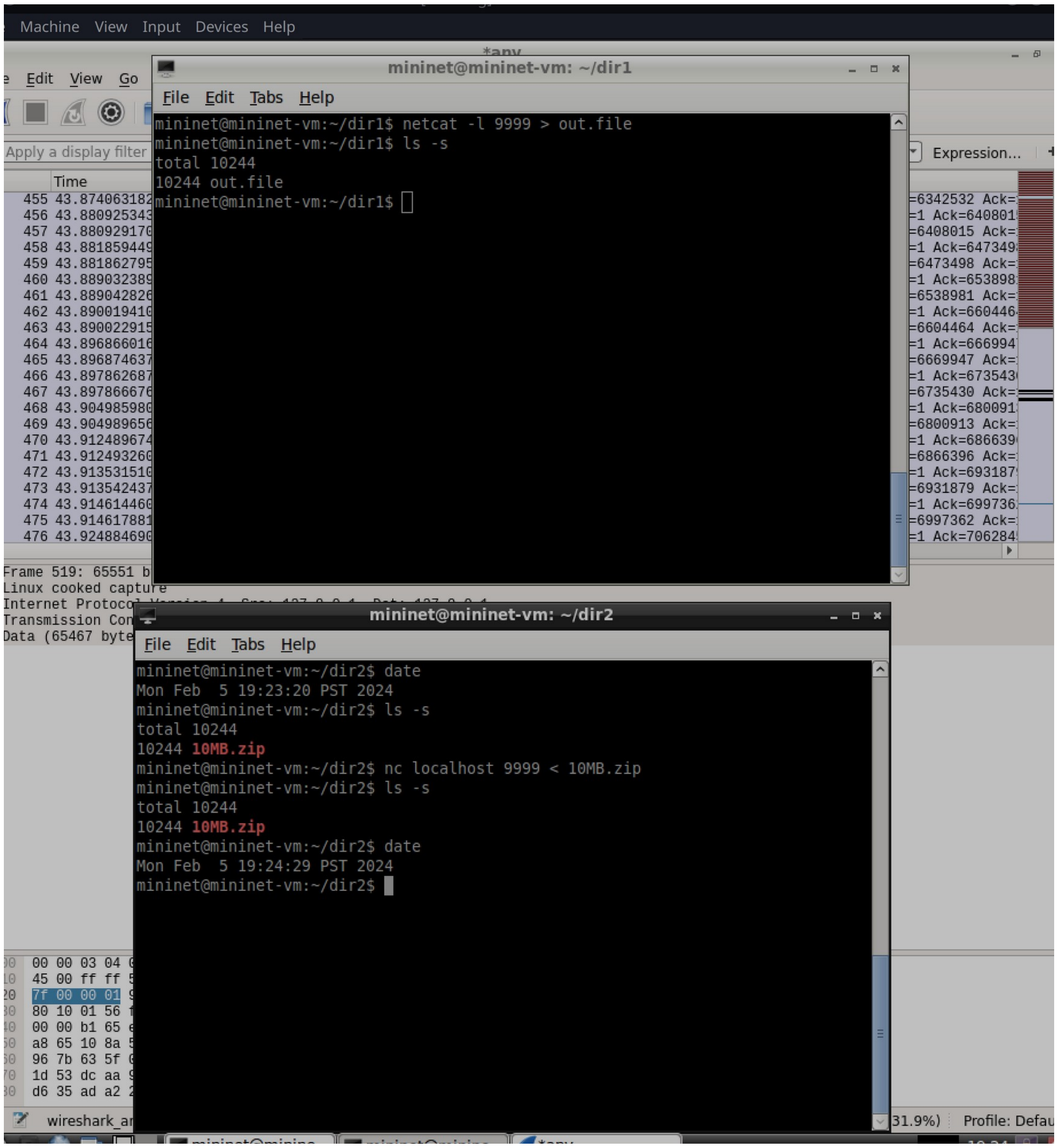
- Open Wireshark to capture the packet transfer
- Create two directories dir1 and dir2
- Open 2 terminals on your VM – one for each of the directories (let's refer to them as terminal 1 with dir1 and terminal 2 with dir2)
- Download http://ipv4.download.thinkbroadband.com/10MB.zip and save it in dir2.
- On terminal 1 inside dir1, run: *`netcat -l 9999 > out.file`*
- On terminal 2 inside dir2, run: *`nc localhost 9999 < 10MB.zip`*

b. Take a **timestamped screenshot** showing the two terminal directory listings before and after the commands are run. Make sure the directory listing displays dates and file size.



```
mininet@mininet-vm:~/dir1$ ls -s
total 0
mininet@mininet-vm:~/dir1$
```

```
mininet@mininet-vm:~/dir2$ date
Mon Feb  5 19:23:20 PST 2024
mininet@mininet-vm:~/dir2$ ls -s
total 10244
10244 10MB.zip
mininet@mininet-vm:~/dir2$
```



```
mininet@mininet-vm:~/dir1$ netcat -l 9999 > out.file
mininet@mininet-vm:~/dir1$ ls -s
total 10244
10244 out.file
mininet@mininet-vm:~/dir1$
```

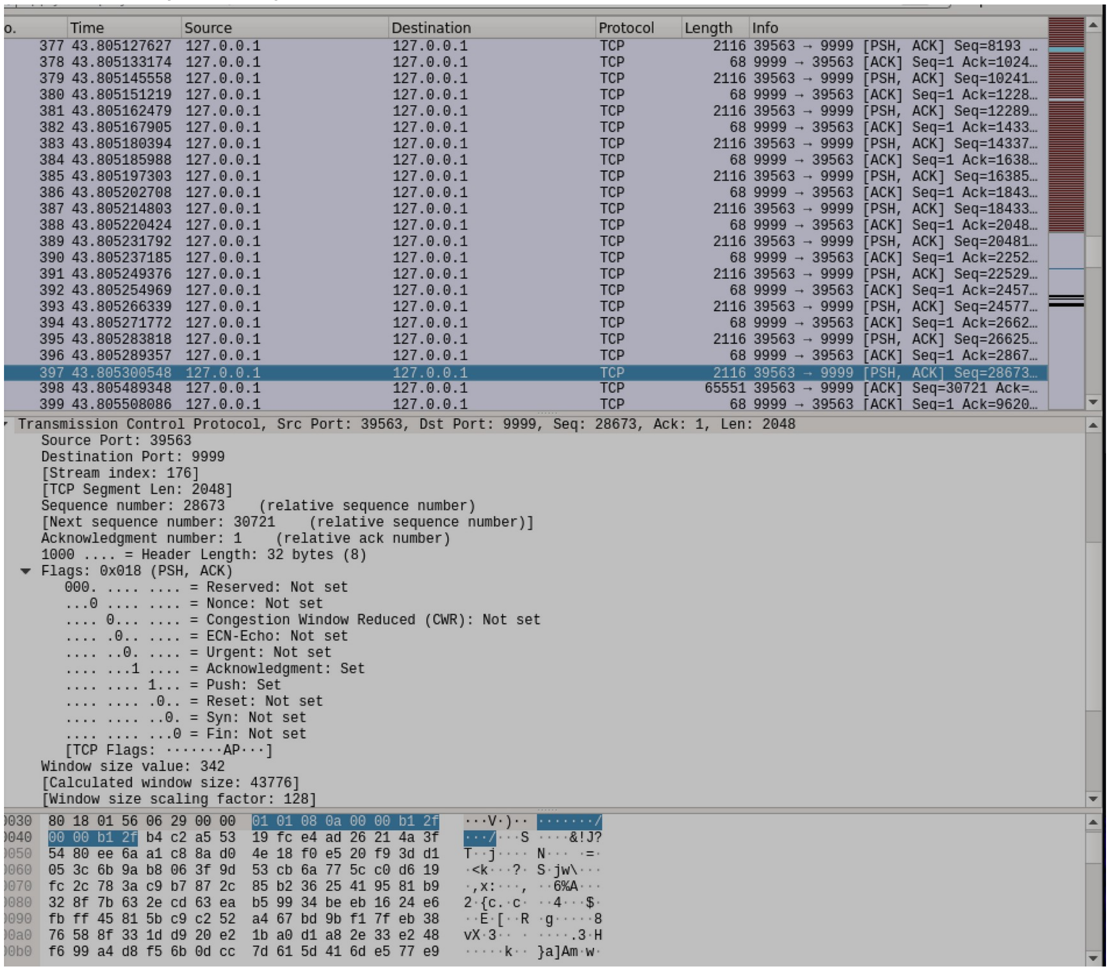
```
mininet@mininet-vm:~/dir2$ date
Mon Feb  5 19:23:20 PST 2024
mininet@mininet-vm:~/dir2$ ls -s
total 10244
10244 10MB.zip
mininet@mininet-vm:~/dir2$ nc localhost 9999 < 10MB.zip
mininet@mininet-vm:~/dir2$ ls -s
total 10244
10244 10MB.zip
mininet@mininet-vm:~/dir2$ date
Mon Feb  5 19:24:29 PST 2024
mininet@mininet-vm:~/dir2$
```

c. These small questions make sure you understand the transfer:
   - Which terminal is sending the file? Terminal 2
   - Which one is receiving it? Identify the Client and the Server. Terminal 1
   - Which command initiates the data transfer? Nc localhost
   - For this transfer to work, which command must be run first? Why?
   - Netcat, so that it opens a listening port for the sender
   - What happens if terminal 2 is executed first?
   - It sends but nothing is recieved

d. Explain this data transfer in your own words
   the NC command opens a listening port and then the netcat connects to the open port 9999 and sends tcp pushes.

e. If the above transfer was done in HTTP, what would be the corresponding HTTP method that accomplishes this transfer? It would use PUT commands to transfer files

f. Use Wireshark to identify the transport layer protocol `netcat` used to transfer the file. Is this what you would expect? Why or why not? Verify in your Wireshark screenshot, markup and explanation. PUSH, I would expect this as its the simplest way to do it



```
o.    Time               Source        Destination   Protocol  Length  Info
  377 43.805127627  127.0.0.1     127.0.0.1     TCP         2116 39563 → 9999 [PSH, ACK] Seq=8193 …
  378 43.805133174  127.0.0.1     127.0.0.1     TCP           68 9999 → 39563 [ACK] Seq=1 Ack=1024…
  379 43.805145558  127.0.0.1     127.0.0.1     TCP         2116 39563 → 9999 [PSH, ACK] Seq=10241…
  380 43.805151219  127.0.0.1     127.0.0.1     TCP           68 9999 → 39563 [ACK] Seq=1 Ack=1228…
  381 43.805162479  127.0.0.1     127.0.0.1     TCP         2116 39563 → 9999 [PSH, ACK] Seq=12289…
  382 43.805167905  127.0.0.1     127.0.0.1     TCP           68 9999 → 39563 [ACK] Seq=1 Ack=1433…
  383 43.805180394  127.0.0.1     127.0.0.1     TCP         2116 39563 → 9999 [PSH, ACK] Seq=14337…
  384 43.805185988  127.0.0.1     127.0.0.1     TCP           68 9999 → 39563 [ACK] Seq=1 Ack=1638…
  385 43.805197303  127.0.0.1     127.0.0.1     TCP         2116 39563 → 9999 [PSH, ACK] Seq=16385…
  386 43.805202708  127.0.0.1     127.0.0.1     TCP           68 9999 → 39563 [ACK] Seq=1 Ack=1843…
  387 43.805214803  127.0.0.1     127.0.0.1     TCP         2116 39563 → 9999 [PSH, ACK] Seq=18433…
  388 43.805220424  127.0.0.1     127.0.0.1     TCP           68 9999 → 39563 [ACK] Seq=1 Ack=2048…
  389 43.805231792  127.0.0.1     127.0.0.1     TCP         2116 39563 → 9999 [PSH, ACK] Seq=20481…
  390 43.805237185  127.0.0.1     127.0.0.1     TCP           68 9999 → 39563 [ACK] Seq=1 Ack=2252…
  391 43.805249376  127.0.0.1     127.0.0.1     TCP         2116 39563 → 9999 [PSH, ACK] Seq=22529…
  392 43.805254969  127.0.0.1     127.0.0.1     TCP           68 9999 → 39563 [ACK] Seq=1 Ack=2457…
  393 43.805266339  127.0.0.1     127.0.0.1     TCP         2116 39563 → 9999 [PSH, ACK] Seq=24577…
  394 43.805271772  127.0.0.1     127.0.0.1     TCP           68 9999 → 39563 [ACK] Seq=1 Ack=2662…
  395 43.805283818  127.0.0.1     127.0.0.1     TCP         2116 39563 → 9999 [PSH, ACK] Seq=26625…
  396 43.805289357  127.0.0.1     127.0.0.1     TCP           68 9999 → 39563 [ACK] Seq=1 Ack=2867…
  397 43.805300548  127.0.0.1     127.0.0.1     TCP         2116 39563 → 9999 [PSH, ACK] Seq=28673…
  398 43.805489348  127.0.0.1     127.0.0.1     TCP        65551 39563 → 9999 [ACK] Seq=30721 Ack=…
  399 43.805508086  127.0.0.1     127.0.0.1     TCP           68 9999 → 39563 [ACK] Seq=1 Ack=9620…
```

```
Transmission Control Protocol, Src Port: 39563, Dst Port: 9999, Seq: 28673, Ack: 1, Len: 2048
   Source Port: 39563
   Destination Port: 9999
   [Stream index: 176]
   [TCP Segment Len: 2048]
   Sequence number: 28673    (relative sequence number)
   [Next sequence number: 30721    (relative sequence number)]
   Acknowledgment number: 1    (relative ack number)
   1000 .... = Header Length: 32 bytes (8)
 ▾ Flags: 0x018 (PSH, ACK)
     000. .... .... = Reserved: Not set
     ...0 .... .... = Nonce: Not set
     .... 0... .... = Congestion Window Reduced (CWR): Not set
     .... .0.. .... = ECN-Echo: Not set
     .... ..0. .... = Urgent: Not set
     .... ...1 .... = Acknowledgment: Set
     .... .... 1... = Push: Set
     .... .... .0.. = Reset: Not set
     .... .... ..0. = Syn: Not set
     .... .... ...0 = Fin: Not set
     [TCP Flags: ·······AP···]
   Window size value: 342
   [Calculated window size: 43776]
   [Window size scaling factor: 128]
```

```
0030  80 18 01 56 06 29 00 00  01 01 08 0a 00 00 b1 2f   ···V·)·· ·······/
0040  00 00 b1 2f b4 c2 a5 53  19 fc e4 ad 26 21 4a 3f   ···/···S ····&!J?
0050  54 80 ee 6a a1 c8 8a d0  4e 18 f0 e5 20 f9 3d d1   T··j···· N··· ·=·
0060  05 3c 6b 9a b8 06 3f 9d  53 cb 6a 77 5c c0 d6 19   ·<k···?· S·jw\···
0070  fc 2c 78 3a c9 b7 87 2c  85 b2 36 25 41 95 81 b9   ·,x:···, ··6%A···
0080  32 8f 7b 63 2e cd 63 ea  b5 99 34 be eb 16 24 e6   2·{c.·c· ··4···$·
0090  fb ff 45 81 5b c9 c2 52  a4 67 bd 9b f1 7f eb 38   ··E·[··R ·g·····8
00a0  76 58 8f 33 1d d9 20 e2  1b a0 d1 a8 2e 33 e2 48   vX·3·· · ····.3·H
00b0  f6 99 a4 d8 f5 6b 0d cc  7d 61 5d 41 6d e5 77 e9   ·····k·· }a]Am·w·
```

g. Choose one of the packets transferred and drill down to find the source and destination IP addresses - circle them in red in the screenshot in (f). Do the addresses make sense to you (you might want to research the loopback address)? Explain. Both are 127.0.01, yes because you are sending it to local host.

14. [15 pts] Compare downloads using `wget` and `netcat` from the origin

server Do the following in your VM:

- Open Wireshark and listen on the `'any'` interface.
- Using `netcat,` download the same file used in question 12) (http://ipv4.download.thinkbroadband.com/10MB.zip)

a. What is the complete command to download the file using `netcat` from the origin server? Explain your command - i.e. what each part does. (Hint: Construct a HTTP 1.1 request to download the file – `Host` header is needed).echo -e "GET /10MB.zip HTTP/1.1\r\nHost:

ipv4.download.thinkbroadband.com\r\nConnection: close\r\n\r\n" | nc ipv4.download.thinkbroadband.com 80 > downloaded_file

you first make an echo of an http header then use netcat to pull from origin.

b. Is `netcat` a useful tool? How is it different from using wget?

Yes, it allows you to send files from one device to another using tcp, however it is not very secure, a system like scp would be better. For dowloading files from a server it is quite inefficient since you need to make an http header. Wget just does everything automatically

c. Would you expect the file distribution time and throughput using `netcat` to be similar to using wget or different? Explain your answer. It should be the same as it uses the same tools to download however it takes longer to write the http header.

Extra Credit (5 points)

15. Networking meets art – Let's draw what's going on!

Netcat transfer: Draw a simple Client-Server architecture overview of the file transfer in Question 13. Remember that the client and server were set up inside your own VM, so start your drawing with a box representing the VM and label it "Virtual Machine". Show the following elements and the interaction of components involved in the transfer in the drawing:

a. The hardware and software components (clients, servers, VM, etc).
b. All IP addresses and ports. (refer to Wireshark capture if needed)
c. Arrows representing the data flow.

Web Transfer: Suppose now the client requests a web page from an web external server with IP address 220.15.34.30.

d. Add this server to your drawing, including the additional intermediate elements (such as Internet connection).Indicate in your drawing the communication between the client and server. Label the IP address and the port number that the server would be listening on.

You can update your drawing above to show the new communication (don't need a second drawing).

Note: refer to your class notes for sample drawings containing a simple network, with clients and server. Additionally, **hand made drawings are not acceptable for Extra Credit-** please take this opportunity to learn how to use drawing software. You can ask your TA for suggestions!

Timestamped Screenshot Check-off: (just for you to keep track of your progress)

Problem #	Screenshot	Problem #	Screenshot
Q2		Q8	
Q3		Q10	
Q4		Q12 b,f	
Q5		Q13 b, f	