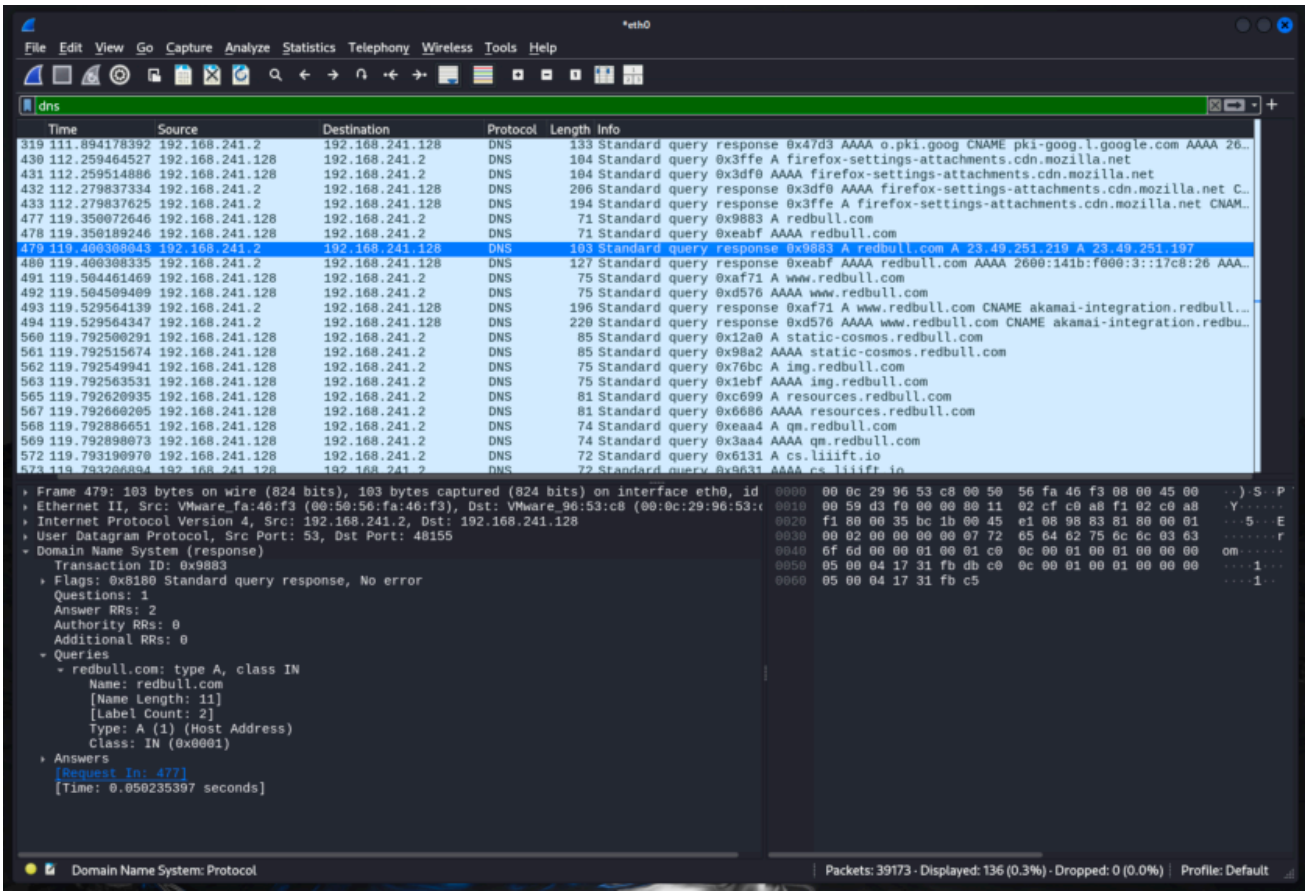
hw3

1. a) False, each request has one response so there will be 4 requests for 4 responses
   b) False, with nonpersistent each time a new request is created, a new TCP connection needs to be established
   c) False it represents the date of the request, the last modified has that info
   d) False, 304 not modified has empty message body
2. a) Finding the IP address via a DNS is needed
   b) It can be done via cookies where each user has a unique cookie id then each purchase can be logged with that specific cookie id.
   c) It reduces the delay by storing frequently requested objects "closer" to the user so that every time the user wants it, it is already available (either in the browser of a local CDN). It only reduces it for objects that have been requested before (either by the same user or different user as well) so if the object has not been requested it won't reduce the time.
3. What you can do is use the dig command on the external website you think was accessed and then compare it with another commonly used website as well as a not commonly used website. If the query time is closer to a commonly used website, that means the IP was cached at it was likely searched up earlier. If it is closer to a less common website, then the IP was not cached and was not searched
4. a) Not reliably no. The way torrenting works is that you receive the most data from the people that are sending you the most data. So if you are not sending anything you will also not get anything back. You may get randomly selected but if you are not sending anything the peer will stop sending to you. Purely theoretically speaking, it is possible by luck to get it by getting randomly selected but it is unlikely and would take a while.
   b) One idea that I had was sending the tracker a log of how much you sent each user and how much you have received from each user. If there is a user that is an outlier and is receiving much more than sending, then kick them out of the torrent
   Another idea shared here https://pub.tik.ee.ethz.ch/students/2006-So/MA-2006-26.pdf was sending proof which gave me the idea of a proof of work scheme kind of like the blockchain has
5. a) So assuming this is a recursive DNS, the user first goes to its own DNS (if it has one) which then requests the root DNS. The root requests the TLD. The TLD sends a request to the authoritative which send the requested IP address back which propagates back up the chain until the user recieves it. Then the user establishes the HTTP connection via TCP and then requests for the object itself
   b) $x_1 + x_2 + \ldots x_N + 2x_0$ (2 to establish HTTP)

6. 

a) redbull.com

b) Theres one thats A and another thats AAAA

c) 23.49.251.197 and 23.49.251.219

d) Theres one thats A and another thats AAAA

e) Since we didn't know the IP associated with redbull.com, we instead ask the DNS. The DNS is a database with the domain name and the IP registered with it so it searches the name, finds its respective IP and sends it to us so we can connect to it

7. a) nslookup seemed more simple and easy to use while dig seemed a bit more complex but providing much more information. Interestingly, I tried the steam website and it didn't work for nslookup suggesting they might have their own database or something.

b) -debug can be used is an option for getting more detailed information -port= {port_number} specifies a specific port number to use and -timeout={time} is for setting a max time before exiting

c) +short displays only the essential info, +x is a reverse DNS lookup, +stats shows statistics

I ran it on Google and there were pretty much no differences. I also ran it on 1337x.to and there were actually more domains listed but only like one or 2

d) Trace starts at the root and iterates through each level showing the response from root to TLD to authoritative