Computer Networks Assignment 2

Navneel Singhal

October 29, 2020

Contents

1 Part 1 — 1

2 Part 2 — 2

3 Part 3 — 4

1 Part 1

Problem: Use wireshark to grab all packets on your wireless interface, while visiting the website `http://www.cse.iitd.ac.in` from your browser. Do an `ipconfig /flushdns`[1] before you do this activity to clear your local DNS cache. And also clear your browser cache. Report the following:

1. Apply a "dns" filter on the packet trace, and see if you can find DNS queries and responses for `www.cse.iitd.ac.in`. What DNS server was used? How long did it take for the DNS request-response to complete?

   Answer: Yes, I can find one query and one response for the website. It seems as if my router itself was behaving like the DNS server, since the destination in the query and the source in the response was the IP address of my router, 192.168.1.1. It took 351ms to carry out the DNS query.

2. Apply an "http" filter on the packet trace and report the approximate number of HTTP requests that were generated to download all the objects on the home-page. What can you tell from this observation about how web-pages are structured, and how browsers render complex pages with multiple images and files?

   Answer: There were 82 HTTP packets in total, with 41 requests from my IP (192.168.1.204) to the IP of the web-server (103.27.9.152) and 41 packets sent in the other direction. From the number of packets, we get to know that web-pages are split into multiple components like html content, css content, js scripts, images, and other media. From the packets received, we can see that webpages are in the form of HTML and CSS files, with embedded JS. From here it seems that the browser starts processing the main HTML file, and then for each piece of content (multimedia or script), it sends out an HTML request to procure that file from the web server and so on (while recursively parsing the webpage and also sending out HTML requests when certain actions get triggered in the scripts).

3. Apply a filter such as `((ip.src==192.168.1.3 && ip.dst==10.7.174.111) || (ip.src==10.7.174.111 && ip.dst==192.168.1.3)) && tcp`. As would be self-explanatory, this will filter for TCP packets moving between your browser and the web-server. Recall that the source and destination IP addresses are a part of the network layer header, which is also called the IP layer since IP (Internet Protocol) is the most common network layer protocol in use. Find the number of TCP connections that were opened between your browser and the web-server. Recall that a TCP connection is identified by the 4-tuple (source IP, destination IP, source port, destination port).

   Answer: The number of TCP connections opened between the browser and the web-server was 23, as was found by the following three methods after filtering by `((ip.src == 192.168.1.204 && ip.dst == 103.27.9.152) || (ip.dst == 192.168.1.204 && ip.src == 103.27.9.152)) && tcp`:

   (a) Finding the number of distinct 4-tuples as mentioned above.

[1] For Ubuntu, I did `sudo systemd-resolve --flush-caches` since the suggested commands for Ubuntu in the document didn't work

(b) (Cross-verification) Finding the number of appearances of `[SYN, ACK]`, since any `[SYN, ACK]` happens only at the start of a TCP connection.

(c) (Cross-verification) Going to the `Statistics` menu → `Conversations` and going to the `TCP` tab and checking the `Limit to display filter`.

4. In the previous part, do you find that several content objects are fetched over the same TCP connection?

   Answer: Yes, there are several such examples. For example, the first CSS file is fetched over the same TCP connection which fetched the HTML file.

5. Notice that before an HTTP message is sent on a new TCP connection, a 3-way handshake is first performed to establish the TCP connection. The client sends a SYN message to the server, the server replies with a SYN-ACK message, and the client then sends an ACK. You will find that several TCP connections were opened between your browser and the web-server. How much time does it take for this handshake, before the connection can be used to send/receive data? Given this latency, what kind of optimizations do you think the browser might want to follow to minimize the overall page-load time?

   Answer: The latency for this handshake is about 5-15ms. To minimize the overall page-load time, the browser should try to use the keep-alive optimization (keeping TCP connections open for multiple packets) as well as opening multiple TCP connections in parallel, both of which seem to be done in my wireshark trace.

6. Report the total time taken for download of the entire webpage measured as the time at which the first DNS request was sent and the time when the last content object was received. This is called the page load time. Another useful metric that browsers try to optimize is called the above fold time, which is the time taken to first download objects that are sufficient to render the part of the web-page that's visible on the screen, i.e. above the fold.

   Answer: The total time taken for the download of the entire webpage is 1.4528s, which is the time when the last content object was received (the first DNS request was sent at 0.0000s).

7. Try doing a trace for `http://www.indianexpress.com` and filter for "http". What do you find, is there any HTTP traffic? Browse through the entire trace without any filters, are you able to see the contents of any HTML and Javascript files being transferred? Why is that, while you were able to do it easily earlier for `http://www.cse.iitd.ac.in`?

   Answer: There was an attempt to connect via HTTP, but there was a returned code 301, which says `Moved Permanently`. On searching for this error code, I found that this is considered to be a best practice for upgrading users from HTTP to HTTPS, and the lack of subsequent HTTP packets even though the whole website loaded properly indicates that this is indeed the case.

   While browsing through the entire trace, it was noticed that there are TLS packets where TLS means transport layer security, and this means that these packets are securely encrypted at the transport layer, indicating the presence of a secure communication channel, which indicates HTTPS again. In the case for `http://cse.iitd.ac.in`, we saw that the connection was done via HTTP and not HTTPS (there is a `Not Secure` warning in the address bar in my browser as well), so the content of the packets was unencrypted and hence easily readable when sniffed by Wireshark.

2 Part 2

Problem: Now open the Chrome browser and go to Developer Tools. Open the Network tab. Go to http://www.indianexpress.com. Report the following:

1. Why are you able to see the different content objects in the browser, which you were earlier not able to see through Wireshark?

   Answer: The reason for this is that while the packets intercepted by Wireshark have TLS, the browser already has the decrypted version with it, so we are able to see the content objects in the Chrome browser.

2. How many content objects were downloaded to render the home-page of www.indianexpress.com? You will see that many of these objects are not from the indianexpress.com domain. Where are they from? What do you think is the purpose of these objects?

   Answer: There were 385 objects that were downloaded to render the home-page. Some of the objects not from the domain are for the purposes of advertising, some for rendering fonts (from google api), some are probably for IP geolocation (ip-api.com), some for web push notifications (from izooto), some for accelerated mobile pages (from ampproject) and so on. Not all of these objects are necessary to render the webpage and some are solely for collecting usage analytics, advertising etc.

3. Look at the timing information for any object. Try finding one of the larger objects and look at the Timing sub-tab for this object to see the breakup of the time taken to download the object. What average throughput was observed during the content download period, ie. when the content was actually being downloaded, not counting other latencies like DNS lookup or TCP connection establishment delay?

   Answer: The largest object is ima3.js and has size 111kB and content download takes 604.34ms. So the throughput during the content download period is 184kB/s.

4. Now do the same thing for http://www.nytimes.com and compare the total amount of content downloaded to render the NY Times home-page, with the content downloaded to render the Indian Express home-page. What does this tell you about creating websites?

   Answer: There were about 600 packets received for http://www.nytimes.com, and the largest files are also much larger (of the order of a few MBs).

   It was noticed that the nytimes website took more than 3s to load, while the indianexpress website took less than half the time, which means that smaller websites (which have lesser number of requests) are generally faster to load. It can also be inferred that nytimes is more network-hungry, and indianexpress is not so much, maybe due to keeping in mind their demographics and average network speeds and availability. Both these websites have dynamic home webpages, and the content is fetched only when the user scrolls down to that part of the webpage, making initial loading faster and the whole process more economical.

5. Do you agree that from a user-experience point of view, since web-pages are constituted of many small objects and which could be hosted on multiple domains, factors like the roundtrip delay and optimizations by the browser to pipeline downloads of multiple objects, are more important than the network throughput that is obtained? Explain your answer.

   Answer: Yes, I agree that this is more important. As seen in the case of http://www.indianexpress.com, even the largest packet had the delay in the setting up of connections etc. comparable to the actual download time, leave alone the smaller packets. This shows that a major chunk of the time elapsed in loading the webpage is spent in establishing connections and in RTT delays, which substantiates the claim in the statement.

6. Chrome allows you to test websites by emulating different network conditions. Look at the Throttling dropdown in the Network tab and change the network to Fast 3G, Slow 3G, etc. You can also build your own custom network profile by specifying the mean downlink throughput, uplink throughput, and latency. Experiment with different values, such as the following:

   (technology, downlink throughput in kbps, uplink throughput in kbps, latency in ms)

   Regular 2G, 250, 50, 300

   Good 2G, 450, 150, 150

   Regular 3G, 750, 250, 100

   Good 3G, 1000, 750, 40

   Regular 4G, 4000, 3000, 20

   DSL, 2000, 1000, 5

   How do you think Chrome is able to emulate such different networks? Chrome also allows you to emulate different devices which is particularly useful when the computation capabilities of the device may begin to affect the user experience. Why do you think the device computation capabilities can have this effect?

   Answer: I think Chrome emulates such different networks by introducing delays at the browser level after receiving a response from the web-server, because it doesn't simulate it by changing things at the hardware level. The latency and the speed are both simulated by adjusting the delay as mentioned above.

   The device computation capabilities have an effect if the network is very fast but the computation needed to decrypt and render content like media etc is slow and noticeable.

7. Go to http://www.indianexpress.com and export the Chrome trace as a HAR file. Open this file using a text editor understand the information it contains, look up on the web if needed to understand the structure of HAR files. Examine the requests going to ad networks like Double Click or analytics services like Google Analytics and report what third-party domains do you see being accessed? What user-specific

information seems is being sent to these websites? What kind of information do you think these third-party domains are requesting to be saved as cookies locally? Go to your Chrome privacy settings, then to the cookie settings. Do you have third party cookies blocked?

Answer: The third-party domains being accessed are doubleclick.net, google.com, fonts.googleapis.com, imasdk.googleapis.com, amazonaws.com, fonts.gstatic.com, cloudfront.net, ip-api.com, izooto.com, stats.wp.com, sb.scorecardresearch.com, adservice.google.com, google-analytics.com, googlesyndication.com, googletagservices.com, wzrkt.com, adomega-d.openx.net

Some of the cookies sent to these domains are

(a) scorecardresearch: UID, UIDR
(b) adservice.google.com: 1P_JAR, NID, ANID
(c) doubleclick.net: IDE, DSID
(d) wzrkt.com: WZRK_G
(e) adomega-d.openx.net: i, pd

From a search on cookiepedia, the following are the reasons for the cookies:

(a) UID: assigns user ID and gathers data about activity on the website, and this data can be sent to a 3rd party for analysis (for performance)
(b) UIDR: to measure number and behavior of website visitors coming and going. It helps the website send out surveys to better understand user preferences. Used for targeting/advertising.
(c) 1P_JAR: carries out information about how the end user uses the website and any advertising that the end user may have seen before visiting the said website. The main purpose of this cookie is: Targeting/Advertising
(d) NID: uses tracking data to align advertisements etc on the website.
(e) ANID: unknown but seems like it's for advertising
(f) IDE: used for advertising purposes
(g) DSID: set for note down specific user identity, and contains a hashed/encrypted unique ID.
(h) WZRK_G: unknown but seems like it's for advertising
(i) i: unknown but seems like it's for advertising
(j) pd: generally provided by openx.net and is used for advertising purposes

The data requested by these domains to be saved on the local file-system is probably for user identification and maintaining some sort of state to track activity and give targeted ads.

No, third party cookies are not blocked in this new installation of Chromium browser.

3 Part 3

Problem: Let's go back to Wireshark now. Disconnect from the network, release the DHCP address by invoking ipconfig /release[2], then start Wireshark, and connect back to the network.

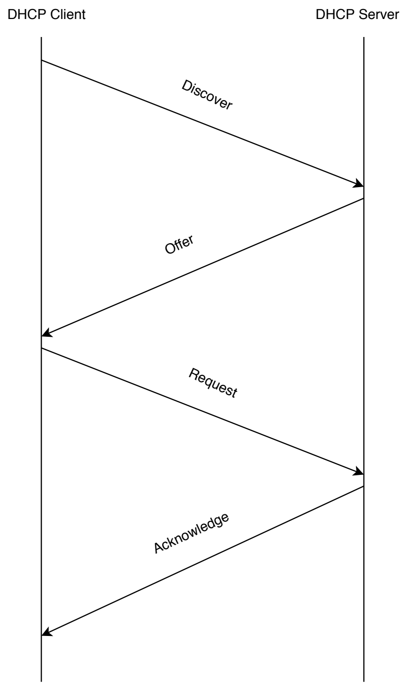
1. Filter for DHCP packets and describe what seems to be the protocol through which DHCP is operating. Draw a transaction diagram of the DHCP messages that you see being sent and received by your device. Report what underlying transport layer protocol is being used.

   From the filtering, I found out that there are 4 DHCP packets that are being used in communication.

   (a) Discover - This packet is broadcasted from my machine, in search of a DHCP server.
   (b) Offer - This packet is broadcasted from the DHCP server (in this case, my router) with the client MAC address as the MAC address of my machine, and apart from offering an IP address, also gives some details like DNS server, subnet mask, router IP, IP address lease time etc.
   (c) Request - This packet is broadcasted from my machine, asking for the offered IP address, to a DHCP server IP address mentioned in the packet (which is the one that offered in the previous packet).
   (d) Ack - This packet is broadcasted from the DHCP server to acknowledge the request.

   The transport layer protocol being used here is UDP.

[2]On Linux, use: `sudo dhclient -r; sudo dhclient`



2. Run a traceroute to www.google.com and similarly report for DNS messages, of what messages are sent and received by your device, and the underlying transport layer protocol.

   There are 16 DNS messages sent and received, and apart from the first two request-response pairs (pertaining to google), all others failed. Upon a careful look, the ones that failed are reverse-DNS lookups (since the type is PTR). The underlying transport layer protocol is UDP.

3. Now filter for ICMP messages and similarly report what messages traceroute seems to be sending.

   There are 68 ICMP messages, and all of these correspond to responses and replies involved in the traceroute. The ping messages have type 8, a sequence number corresponding to which packet number this is, a ttl value (3 packets for each ttl value) and a 32 byte data packet (HIJKLMNOPQRSTUVWXYZ[\]^_`abcdefg). All packets from the middle routers have a response that says TTL exceeded, and this is how traceroute gets the IP address of the router at that hop. However, for all packets with a sufficient TTL to reach google.com, we notice an echo reply from google.com.

4. Run different streaming applications while capturing their packets, like when watching a video on youtube, or talking on Skype, or on Zoom. Report what underlying protocols seem to be in use for the data streams.

   For the data streams, whether YouTube or Skype or Zoom, the predominant transport layer protocol seems to be UDP (even though it might be good for YouTube to use TCP instead).