

Assignment 2

Part-1

1. By default TCP traffic is green. But black highlighting in Wireshark identifies TCP packets with problems. For example, it might suggest that the packets were delivered out-of-order, which could be an indication of network issues or problems in the transmission of data. It's a way for Wireshark to visually flag packets that might require further investigation due to potential anomalies or errors.

2. To filter for all outgoing HTTP traffic, you can use the following filter command:

```
http.request.method == "GET" || http.request.method == requestname
```

This filter looks for HTTP requests with methods "GET" or any other request like "PUT, POST, DELETE, HEAD, OPTIONS, CONNECT, and TRACE" which are commonly used for fetching resources from a server or in general outgoing HTTP requests.

3. In the DNS world, we are trying to cut the resolving time as much as possible

- DNS (Domain Name System):
 - DNS primarily uses UDP (User Datagram Protocol) for its communication.
 - DNS requests are typically short, so they have no problems fitting into the UDP segments.
 - It doesn't use a time-consuming three-way hand-shake procedure to start the data transfer like TCP does. The UDP just transmits the data and saves plenty of time.
 - UDP can support many more clients at the same time thanks to the lack of connection state (UDP is connectionless and doesn't require a response). While the TCP has Receive and Send buffers, Sequence and Acknowledge Number Parameters and congestion-control parameters.
 - DNS has a huge workload so it doesn't prefer TCP which takes a long time.
- HTTP (Hypertext Transfer Protocol):
 - HTTP uses TCP (Transmission Control Protocol) for its communication.
 - HTTP messages, especially those containing web page content, can be larger and may span multiple TCP packets.
 - Among the two most common transport protocols on the Internet, TCP is reliable and UDP isn't. HTTP therefore relies on the TCP standard, which is connection-based.
 - Before a client and server can exchange an HTTP request/response pair, they must establish a TCP connection, a process which requires several round-trips.

In conclusion, DNS uses UDP due to its speed, efficiency, and suitability for most DNS operations. UDP allows fast DNS resolution of domain names, quick delivery of DNS queries and responses, and efficient processing of small, time-sensitive data transfers. While TCP is employed in specific cases such as zone transfers, larger responses, and encrypted communication, UDP remains the preferred choice for its lightweight nature and low resource usage.

Part-2

1. The following different protocols appeared in the protocol column in the unfiltered packet-listing window in Wireshark GUI in one session:

TCP, UDP, HTTP, DNS, ARP, IMAP, TLSv1.2, TLSv1.3, OCSP, QUIC, SSDP.

2.

30718	14:17:30.680985	10.240.205.5	34.107.221.82	HTTP	357 GET /canonical.html HTTP/1.1
30721	14:17:30.696573	34.107.221.82	10.240.205.5	HTTP	352 HTTP/1.1 200 OK (text/html)
30722	14:17:30.709285	10.240.205.5	34.107.221.82	HTTP	359 GET /success.txt?ipv4 HTTP/1.1

According to the screenshot, the time interval between the HTTP GET message and HTTP OK message is $14:17:30.680985 - 14:17:30.696573 = 0.015588\text{s}$.

3. **www.amazon.com:** 10.240.205.131
my computer: 10.240.205.5

4.

13908	14:45:41.182679	10.240.205.5	10.240.205.131	HTTP	303 GET /nservice/ HTTP/1.1
13912	14:45:41.189714	10.240.205.131	10.240.205.5	HTTP/X...	499 HTTP/1.1 200 OK
13928	14:45:42.195702	10.240.205.5	10.240.205.131	HTTP	289 GET /nservice/ HTTP/1.1
13931	14:45:42.199390	10.240.205.131	10.240.205.5	HTTP/X...	499 HTTP/1.1 200 OK

The above screenshot shows 2 GET requests from my computer to the URL “www.amazon.com” and the subsequent OK responses.

Screenshot of GET request:

13928	14:45:42.195702	10.240.205.5	10.240.205.131	HTTP	289 GET /nservice/ HTTP/1.1
13931	14:45:42.199390	10.240.205.131	10.240.205.5	HTTP/X...	499 HTTP/1.1 200 OK

> Frame 13928: 289 bytes on wire (2312 bits), 289 bytes captured (2312 bits) on interface \Device\NPF_{27A30D6E-F35B-4C87-AAD4-4BFA9668CF4E}, id 0

> Ethernet II, Src: AzureWaveTec_c7:3e:39 (14:13:33:c7:3e:39), Dst: SamsungElect_0c:87:ca (d0:03:df:0c:87:ca)

> Internet Protocol Version 4, Src: 10.240.205.5, Dst: 10.240.205.131

> Transmission Control Protocol, Src Port: 60569, Dst Port: 7678, Seq: 1, Ack: 1, Len: 235

> Hypertext Transfer Protocol

Screenshot of OK response:

13928	14:45:42.195702	10.240.205.5	10.240.205.131	HTTP	289 GET /nservice/ HTTP/1.1
13931	14:45:42.199390	10.240.205.131	10.240.205.5	HTTP/X...	499 HTTP/1.1 200 OK

> Frame 13931: 499 bytes on wire (3992 bits), 499 bytes captured (3992 bits) on interface \Device\NPF_{27A30D6E-F35B-4C87-AAD4-4BFA9668CF4E}, id 0

> Ethernet II, Src: SamsungElect_0c:87:ca (d0:03:df:0c:87:ca), Dst: AzureWaveTec_c7:3e:39 (14:13:33:c7:3e:39)

> Internet Protocol Version 4, Src: 10.240.205.131, Dst: 10.240.205.5

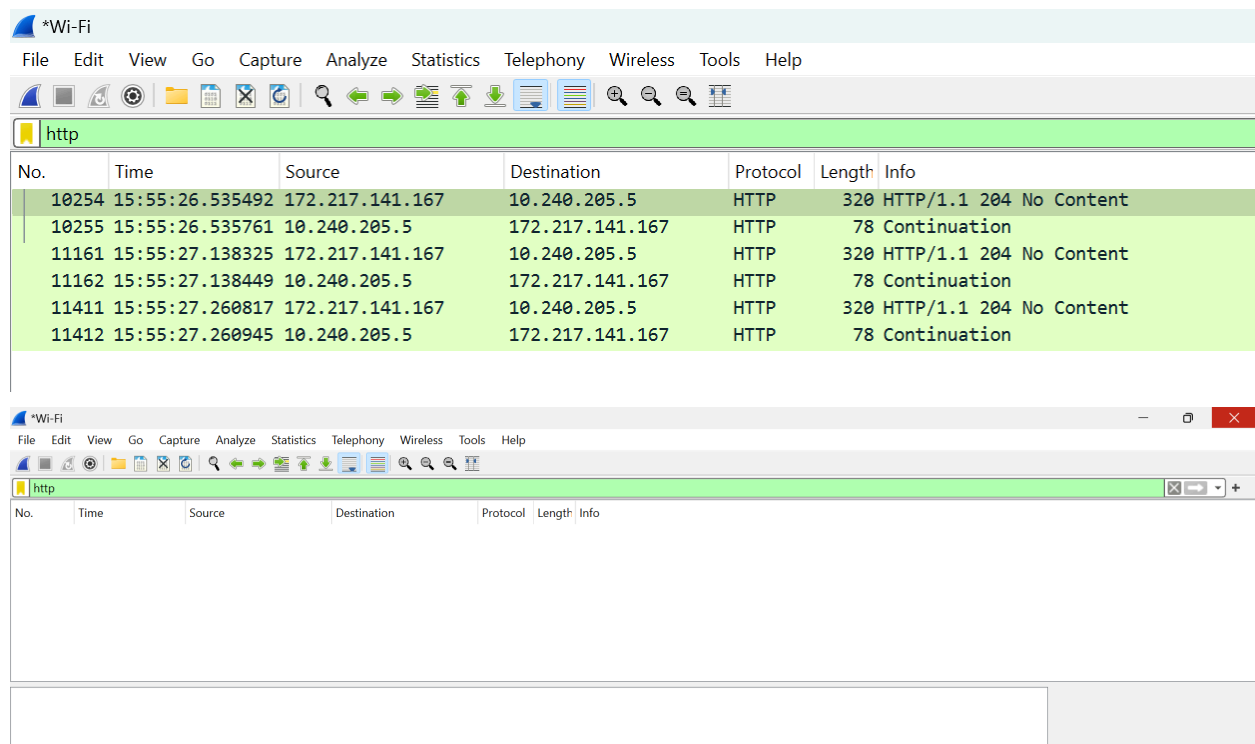
> Transmission Control Protocol, Src Port: 7678, Dst Port: 60569, Seq: 1251, Ack: 236, Len: 445

> [2 Reassembled TCP Segments (1695 bytes): #13930(1250), #13931(445)]

> Hypertext Transfer Protocol

> eXtensible Markup Language

5.



The top screenshot shows a Wireshark capture of HTTP traffic. The packet list contains six entries, all of which are HTTP 204 No Content responses. The details pane shows the structure of an HTTP message, including the status line and body.

No.	Time	Source	Destination	Protocol	Length	Info
10254	15:55:26.535492	172.217.141.167	10.240.205.5	HTTP	320	HTTP/1.1 204 No Content
10255	15:55:26.535761	10.240.205.5	172.217.141.167	HTTP	78	Continuation
11161	15:55:27.138325	172.217.141.167	10.240.205.5	HTTP	320	HTTP/1.1 204 No Content
11162	15:55:27.138449	10.240.205.5	172.217.141.167	HTTP	78	Continuation
11411	15:55:27.260817	172.217.141.167	10.240.205.5	HTTP	320	HTTP/1.1 204 No Content
11412	15:55:27.260945	10.240.205.5	172.217.141.167	HTTP	78	Continuation

The bottom screenshot shows the same Wireshark interface, but the packet list is empty, indicating that no packets were captured during this session.

The reason for not seeing HTTP GET and OK messages in Wireshark when using Google Chrome could be due to how modern browsers handle connections and load resources. Browsers are designed to optimize performance, and they often use various techniques to minimize the number of network requests and enhance page loading speed. There can be numerous reasons for this, some of them are:

HTTP/2 and Multiplexing:

- Modern browsers, including Google Chrome, often use the HTTP/2 protocol, which supports multiplexing. In HTTP/2, multiple requests and responses can be sent and received in parallel over a single connection.

Caching:

- Browsers aggressively cache resources to improve page load times. If a resource is already cached, the browser might not send an explicit HTTP GET request for it. Cached resources may not be visible in Wireshark captures, as the browser can use locally stored copies without making network requests.

Browser Security Features:

- Browsers might employ security features like HTTPS (secure http), which encrypts the content of the communication between the browser and the server, then "POST/GET" can't be seen as they'll be encrypted inside a TLS packet. Gmail, for instance, uses https.
- While you will still see the initial handshake and encrypted traffic, the actual content of the HTTP requests and responses might not be visible in plain text.

Firewall or Antivirus:

- Firewall or antivirus software may be blocking Wireshark from capturing packets.