CS471 – Web Technologies (Laboratory)	Qassim University College of Computer | كلية الحاسب جامعة القصيم	Lab 1 The Internet Protocols

This lab session covers the usage of the Wireshark application to monitor and capture the outgoing and incoming packets from a network connection (WIFI, ethernet, etc.). Specifically, students should be able to analyze HTTP, HTTPS, TCP/IP, and UDP protocols using Wireshark, a network protocol analyzer, and draw conclusions.

Pre-lab Preparation:

1. Review the basics and the structure of HTTP, TCP/IP, and UDP protocols,
2. Install Wireshark and ensure it is running on your computer,
3. Create an online, *publically accessible* Git repository to host and upload your work in the labs. We recommend you use GitHub or GitLab.

Lab Activities:

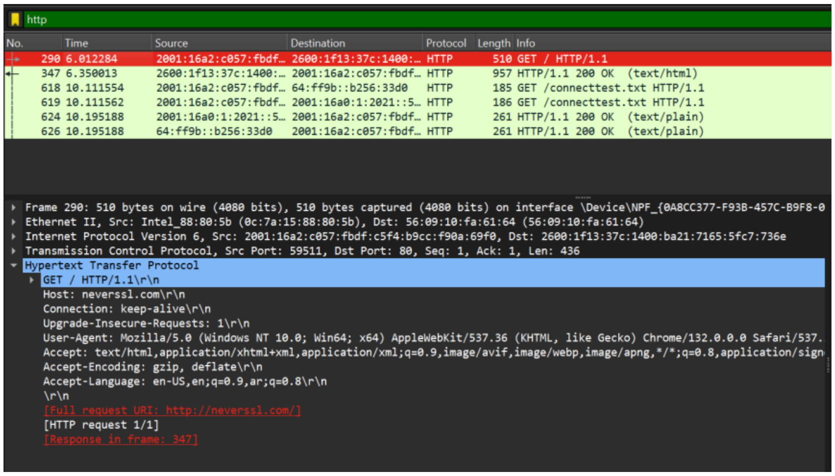
Part 1: Capturing HTTP Traffic.

Task 1: Start Wireshark and capture packets.

Step 1: Open Wireshark.
Step 2: Select the network interface connected to the internet (e.g., Ethernet or Wi-Fi).
Step 3: Click the "Start Capturing Packets" button (the shark fin icon).
Step 4: Open your favorite web browser and navigate to (http://neverssl.com/) website.
Step 5: After the website has fully loaded, stop capturing packets by clicking the red stop button in Wireshark.

Task 2: Filter HTTP packets and analyze them.

Step 1: In the filter bar, type http and press Enter. This filters out only the HTTP packets from the capture.
Step 2: Select any HTTP packet to view its details.
Step 3: Observe the HTTP request and response messages. Note the method (GET, POST), URL, response codes (200 OK, 404 Not Found), etc.

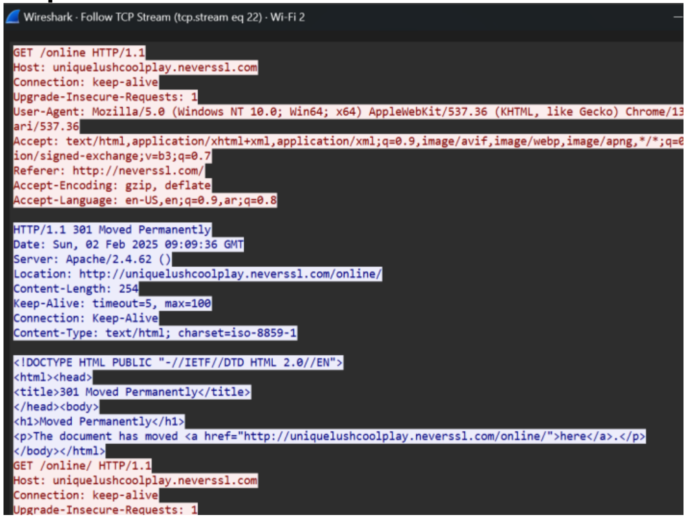


```
http
No.   Time       Source                Destination           Protocol  Length Info
  290 6.012284   2001:16a2:c057:fbdf… 2600:1f13:37c:1400:… HTTP         510 GET / HTTP/1.1
  347 6.350013   2600:1f13:37c:1400:… 2001:16a2:c057:fbdf… HTTP         957 HTTP/1.1 200 OK  (text/html)
  618 10.111554  2001:16a2:c057:fbdf… 64:ff9b::b256:33d0   HTTP         185 GET /connecttest.txt HTTP/1.1
  619 10.111562  2001:16a2:c057:fbdf… 2001:16a0:1:2021::5… HTTP         186 GET /connecttest.txt HTTP/1.1
  624 10.195188  2001:16a0:1:2021::5… 2001:16a2:c057:fbdf… HTTP         261 HTTP/1.1 200 OK  (text/plain)
  626 10.195188  64:ff9b::b256:33d0   2001:16a2:c057:fbdf… HTTP         261 HTTP/1.1 200 OK  (text/plain)

> Frame 290: 510 bytes on wire (4080 bits), 510 bytes captured (4080 bits) on interface \Device\NPF_{0A8CC377-F93B-457C-B9F8-0
> Ethernet II, Src: Intel_88:80:5b (0c:7a:15:88:80:5b), Dst: 56:09:10:fa:61:64 (56:09:10:fa:61:64)
> Internet Protocol Version 6, Src: 2001:16a2:c057:fbdf:c5f4:b9cc:f90a:69f0, Dst: 2600:1f13:37c:1400:ba21:7165:5fc7:736e
> Transmission Control Protocol, Src Port: 59511, Dst Port: 80, Seq: 1, Ack: 1, Len: 436
v Hypertext Transfer Protocol
  > GET / HTTP/1.1\r\n
    Host: neverssl.com\r\n
    Connection: keep-alive\r\n
    Upgrade-Insecure-Requests: 1\r\n
    User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/132.0.0.0 Safari/537.
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/sign
    Accept-Encoding: gzip, deflate\r\n
    Accept-Language: en-US,en;q=0.9,ar;q=0.8\r\n
    \r\n
    [Full request URI: http://neverssl.com/]
    [HTTP request 1/1]
    [Response in frame: 347]
```

Part 2: Analyzing TCP/IP Traffic.

Task 1: Filter TCP packets

Step 1: Clear the previous filter and type TCP to focus on TCP packets.
Step 2: Select a TCP packet related to your HTTP request/response.
Step 3: Right-click on the packet and select "Follow" -> "TCP Stream".
Step 4: This shows the entire conversation between the client and server.



Wireshark · Follow TCP Stream (tcp.stream eq 22) · Wi-Fi 2

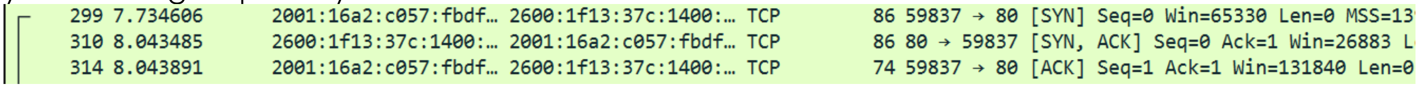
```
GET /online HTTP/1.1
Host: uniquelushcoolplay.neverssl.com
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/13
ari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0
ion/signed-exchange;v=b3;q=0.7
Referer: http://neverssl.com/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9,ar;q=0.8

HTTP/1.1 301 Moved Permanently
Date: Sun, 02 Feb 2025 09:09:36 GMT
Server: Apache/2.4.62 ()
Location: http://uniquelushcoolplay.neverssl.com/online/
Content-Length: 254
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>301 Moved Permanently</title>
</head><body>
<h1>Moved Permanently</h1>
<p>The document has moved <a href="http://uniquelushcoolplay.neverssl.com/online/">here</a>.</p>
</body></html>
GET /online/ HTTP/1.1
Host: uniquelushcoolplay.neverssl.com
Connection: keep-alive
Upgrade-Insecure-Requests: 1
```

Task 2: Analyze TCP handshake and investigate Data Transfer and Termination

Step 1: Find and select packets related to the TCP three-way handshake:

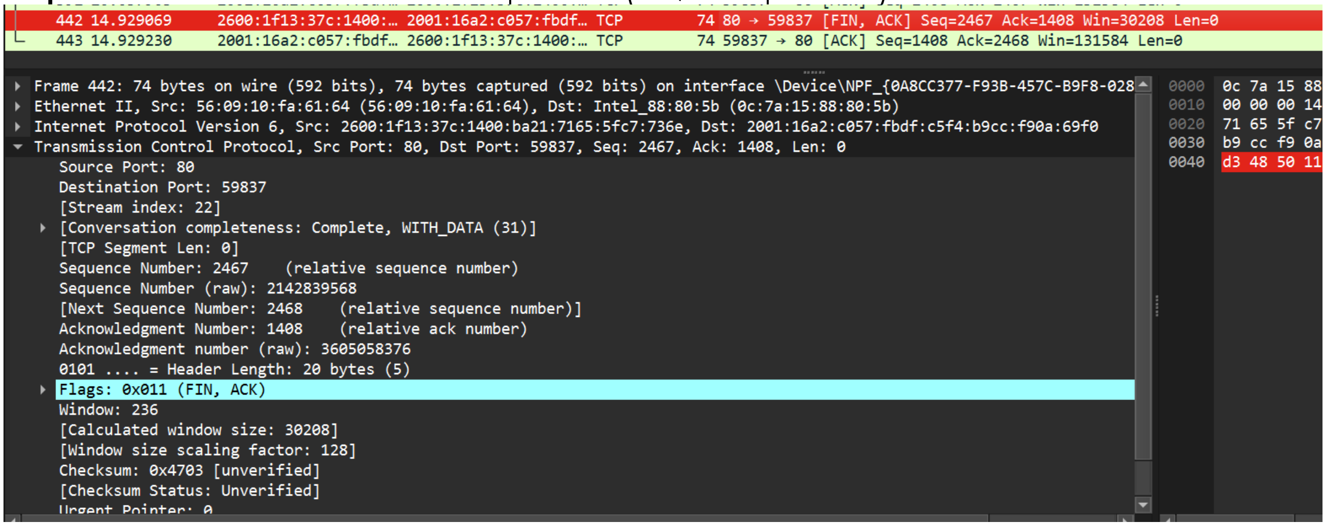
- SYN: Initiates a connection.
- SYN-ACK: Acknowledges and responds to the SYN.
- ACK: Acknowledges the SYN-ACK and establishes the connection.

Step 2: Note the sequence and acknowledgment numbers. Screenshot and upload your image to your online git repository.



```
┌  299 7.734606     2001:16a2:c057:fbdf… 2600:1f13:37c:1400:… TCP      86 59837 → 80 [SYN] Seq=0 Win=65330 Len=0 MSS=13
   310 8.043485     2600:1f13:37c:1400:… 2001:16a2:c057:fbdf… TCP      86 80 → 59837 [SYN, ACK] Seq=0 Ack=1 Win=26883 L
   314 8.043891     2001:16a2:c057:fbdf… 2600:1f13:37c:1400:… TCP      74 59837 → 80 [ACK] Seq=1 Ack=1 Win=131840 Len=0
```

Step 3: Observe the data packets exchanged between the client and server. Take a screenshot and upload it to your online git repo.
Step 4: Look at the TCP termination process (FIN, ACK packets).



```
   442 14.929069    2600:1f13:37c:1400:… 2001:16a2:c057:fbdf… TCP      74 80 → 59837 [FIN, ACK] Seq=2467 Ack=1408 Win=30208 Len=0
└  443 14.929230    2001:16a2:c057:fbdf… 2600:1f13:37c:1400:… TCP      74 59837 → 80 [ACK] Seq=1408 Ack=2468 Win=131584 Len=0

▸ Frame 442: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface \Device\NPF_{0A8CC377-F93B-457C-B9F8-028
▸ Ethernet II, Src: 56:09:10:fa:61:64 (56:09:10:fa:61:64), Dst: Intel_88:80:5b (0c:7a:15:88:80:5b)
▸ Internet Protocol Version 6, Src: 2600:1f13:37c:1400:ba21:7165:5fc7:736e, Dst: 2001:16a2:c057:fbdf:c5f4:b9cc:f90a:69f0
▾ Transmission Control Protocol, Src Port: 80, Dst Port: 59837, Seq: 2467, Ack: 1408, Len: 0
    Source Port: 80
    Destination Port: 59837
    [Stream index: 22]
  ▸ [Conversation completeness: Complete, WITH_DATA (31)]
    [TCP Segment Len: 0]
    Sequence Number: 2467    (relative sequence number)
    Sequence Number (raw): 2142839568
    [Next Sequence Number: 2468    (relative sequence number)]
    Acknowledgment Number: 1408    (relative ack number)
    Acknowledgment number (raw): 3605058376
    0101 .... = Header Length: 20 bytes (5)
  ▸ Flags: 0x011 (FIN, ACK)
    Window: 236
    [Calculated window size: 30208]
    [Window size scaling factor: 128]
    Checksum: 0x4703 [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0

0000  0c 7a 15 88
0010  00 00 00 14
0020  71 65 5f c7
0030  b9 cc f9 0a
0040  d3 48 50 11
```

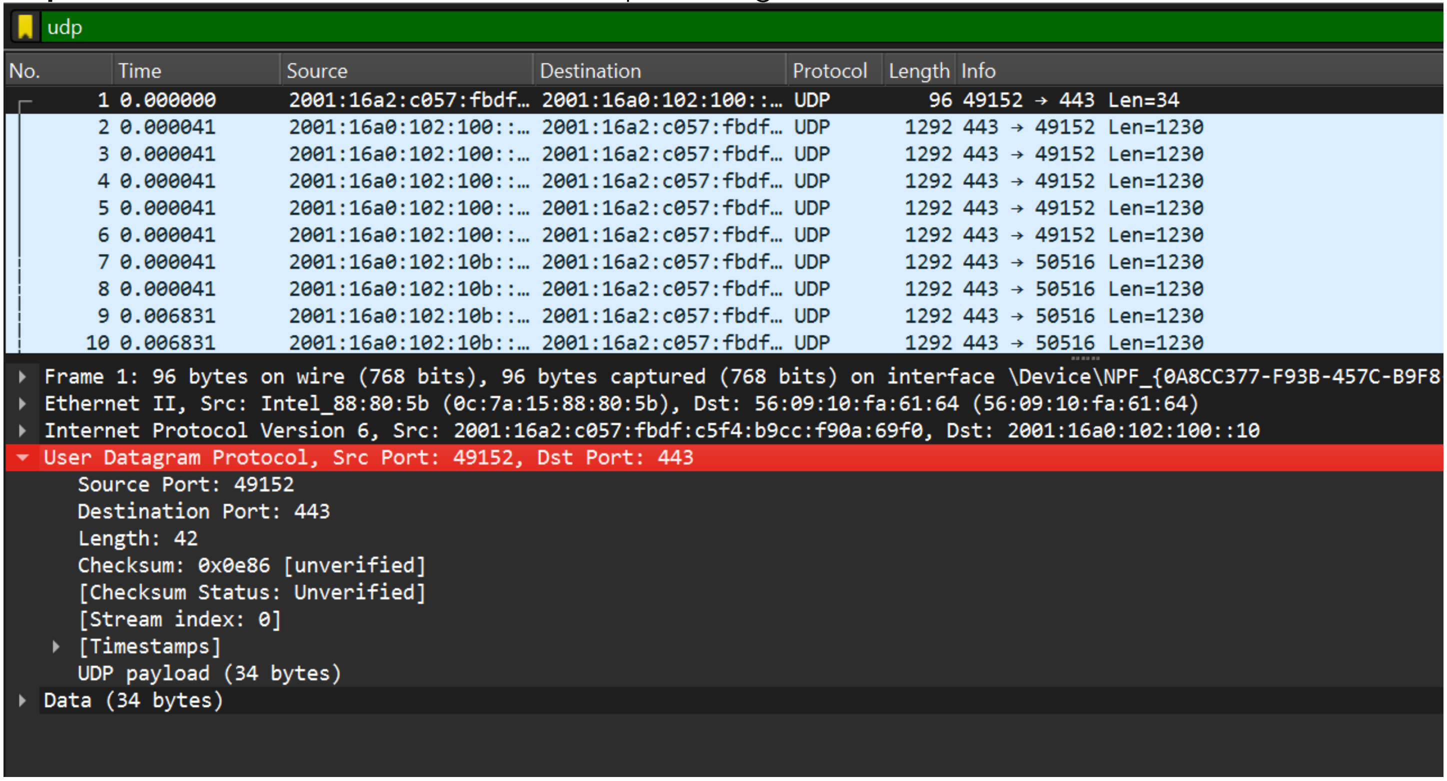
Part 3: Capturing and Analyzing UDP Traffic

Task 1: Generate UDP traffic and capture packets

Step 1: Open a network application that uses UDP (e.g., streaming video, VoIP software, or custom script).
Step 2: Start the application to generate UDP traffic.
Step 3: Start capturing packets in Wireshark while the UDP application is running.
Step 4: After sufficient traffic is generated, stop capturing packets.

Task 2: Filter and analysis UDP Packets

Step 1: In the filter bar, type UDP and press Enter.
Step 2: This filters out only the UDP packets from the capture.
Step 3: Select any UDP packet to view its details.
Step 4: Observe the source and destination ports, length, and data.



```
udp
No.   Time      Source                   Destination              Protocol  Length Info
   1 0.000000   2001:16a2:c057:fbdf…    2001:16a0:102:100::…    UDP          96 49152 → 443 Len=34
   2 0.000041   2001:16a0:102:100::…    2001:16a2:c057:fbdf…    UDP        1292 443 → 49152 Len=1230
   3 0.000041   2001:16a0:102:100::…    2001:16a2:c057:fbdf…    UDP        1292 443 → 49152 Len=1230
   4 0.000041   2001:16a0:102:100::…    2001:16a2:c057:fbdf…    UDP        1292 443 → 49152 Len=1230
   5 0.000041   2001:16a0:102:100::…    2001:16a2:c057:fbdf…    UDP        1292 443 → 49152 Len=1230
   6 0.000041   2001:16a0:102:100::…    2001:16a2:c057:fbdf…    UDP        1292 443 → 49152 Len=1230
   7 0.000041   2001:16a0:102:10b::…    2001:16a2:c057:fbdf…    UDP        1292 443 → 50516 Len=1230
   8 0.000041   2001:16a0:102:10b::…    2001:16a2:c057:fbdf…    UDP        1292 443 → 50516 Len=1230
   9 0.006831   2001:16a0:102:10b::…    2001:16a2:c057:fbdf…    UDP        1292 443 → 50516 Len=1230
  10 0.006831   2001:16a0:102:10b::…    2001:16a2:c057:fbdf…    UDP        1292 443 → 50516 Len=1230

▸ Frame 1: 96 bytes on wire (768 bits), 96 bytes captured (768 bits) on interface \Device\NPF_{0A8CC377-F93B-457C-B9F8
▸ Ethernet II, Src: Intel_88:80:5b (0c:7a:15:88:80:5b), Dst: 56:09:10:fa:61:64 (56:09:10:fa:61:64)
▸ Internet Protocol Version 6, Src: 2001:16a2:c057:fbdf:c5f4:b9cc:f90a:69f0, Dst: 2001:16a0:102:100::10
▾ User Datagram Protocol, Src Port: 49152, Dst Port: 443
     Source Port: 49152
     Destination Port: 443
     Length: 42
     Checksum: 0x0e86 [unverified]
     [Checksum Status: Unverified]
     [Stream index: 0]
   ▸ [Timestamps]
     UDP payload (34 bytes)
▸ Data (34 bytes)
```

Step 5: Compare the simplicity of UDP headers with TCP headers.

Part 4: Comparing TCP and UDP by filling in the following tables. Save your work (e.g., in an MS Word document), and upload it to your online git repo.

Task 1: Fill in the following table and provide reasons.

	TCP or UDP	Reasons
Reliability and Connection Establishment	TCP	TCP provides reliability through connection establishment before data transfer (three-way handshake) and error recovery mechanisms
Data Integrity and Ordering	TCP	TCP ensures data integrity and maintains the order of packets. UDP does not guarantee the order of packets or data integrity

Task 2: Identify the use Cases and Performance of TCP and UDP.

	TCP	UDP
Use cases	Web – email – file transportation	Video Streaming, Online Gaming, VoIP
Performance	reliable, but higher latency	Faster, but less reliable

Ghadi Alsuhaibani