CS 315: Computer Networks Lab
Spring 2024-25, IIT Dharwad
Assignment-5
Chidurala Tejaswini
(220010012/CS22BT012)

Wireshark Lab: TCP

February 9, 2025

Part 0: Paste a screenshot of your system IP address, using `ipconfig` (on Windows) or `ifconfig` (on Mac and Linux), and fill out [this Google form]() to submit the details of your system. The same system must be used to attempt all exercises of this lab.

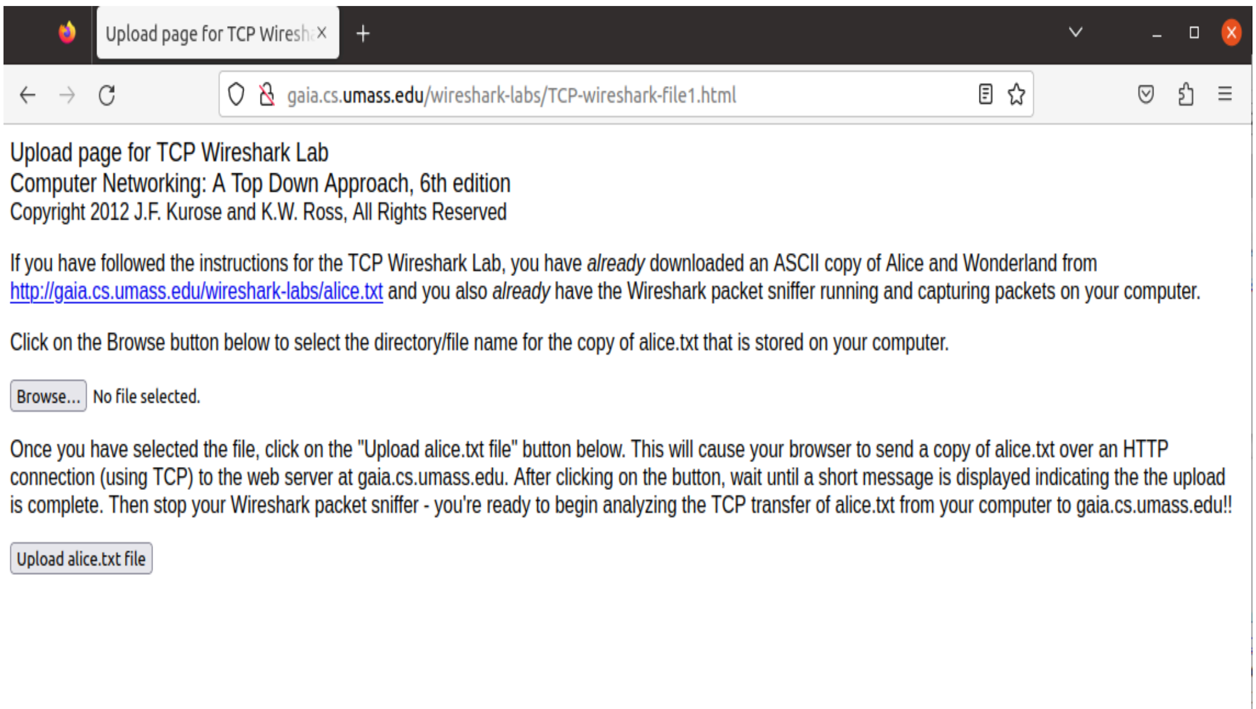
```
user@sysad:~$ ifconfig
eno1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.240.118.96  netmask 255.255.248.0  broadcast 10.240.119.255
        inet6 fe80::ca54:a358:d65:74b7  prefixlen 64  scopeid 0x20<link>
        ether e0:73:e7:0a:79:aa  txqueuelen 1000  (Ethernet)
        RX packets 57380  bytes 62791741 (62.7 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 18420  bytes 4798574 (4.7 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 19  memory 0x80900000-80920000

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 1488  bytes 140600 (140.6 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1488  bytes 140600 (140.6 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

wlp0s20f3: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether b0:dc:ef:d8:78:9f  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Part 1: Capturing a bulk TCP transfer from your computer to a remote server

Before beginning our exploration of TCP, we'll need to use Wireshark to obtain a packet trace of the TCP transfer of a file from your computer to a remote server. You'll do so by accessing a Web page that will allow you to enter the name of a file stored on your computer (which contains the ASCII text of *Alice in Wonderland*), and then transfer the file to a Web server using the HTTP POST method. We're using the POST rather than the GET method as we'd like to transfer a large amount of data from your computer to another. Of course, we'll be running Wireshark during this time to obtain the trace of the TCP segments sent and received from your computer.

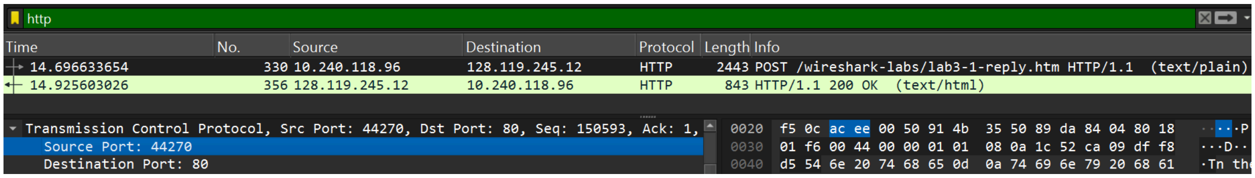
Do the following:

- Start up your web browser. Go the http://gaia.cs.umass.edu/wireshark-labs/alice.txt and retrieve an ASCII copy of *Alice in Wonderland.* Store this as a .txt file somewhere on your computer.
- Next go to http://gaia.cs.umass.edu/wireshark-labs/TCP-wireshark-file1.html
- You should see a screen that looks like the Figure below.



- Use the ***Browse*** button in this form to the file on your computer that you just created containing *Alice in Wonderland*. Don't press the "*Upload alice.txt file*" button yet.
- Now start up Wireshark and begin packet capture (see the earlier Wireshark labs if you need a refresher on how to do this).
- Returning to your browser, press the "*Upload alice.txt file*" button to upload the file to the gaia.cs.umass.edu server. Once the file has been uploaded, a short congratulations message will be displayed in your browser window.
- Stop Wireshark packet capture.

Answer the following questions

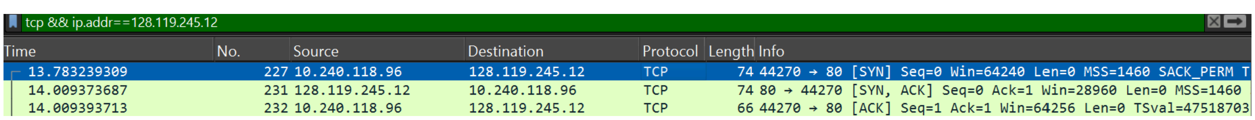


1. **State the number of `GET` and `POST` requests to the `gaia.cs.umass` website.**
   number of GET request to the gaia.cs.umass website=0
   number of POST request to the gaia.cs.umass website=1
2. **State the different type(s) of `http` status code(s) observed in this connection to `gaia.cs.umass` website.**
   The different type of `http` status code(s) observed in this connection to `gaia.cs.umass` website is the response status code is **200 OK**, which indicates that the **POST request** was processed successfully and the expected response was received.
3. **Expand the TCP conversation stream for this connection to `gaia.cs.umass` website and answer the following questions.**
   a. **What are the source and destination IP addresses in the HTTP GET requests?**

Source IP address	10.240.118.96
Destination IP address	128.119.245.12

   b. **What are the source and destination port numbers used in this TCP stream?**

Source port number	44270
Destination port number	80

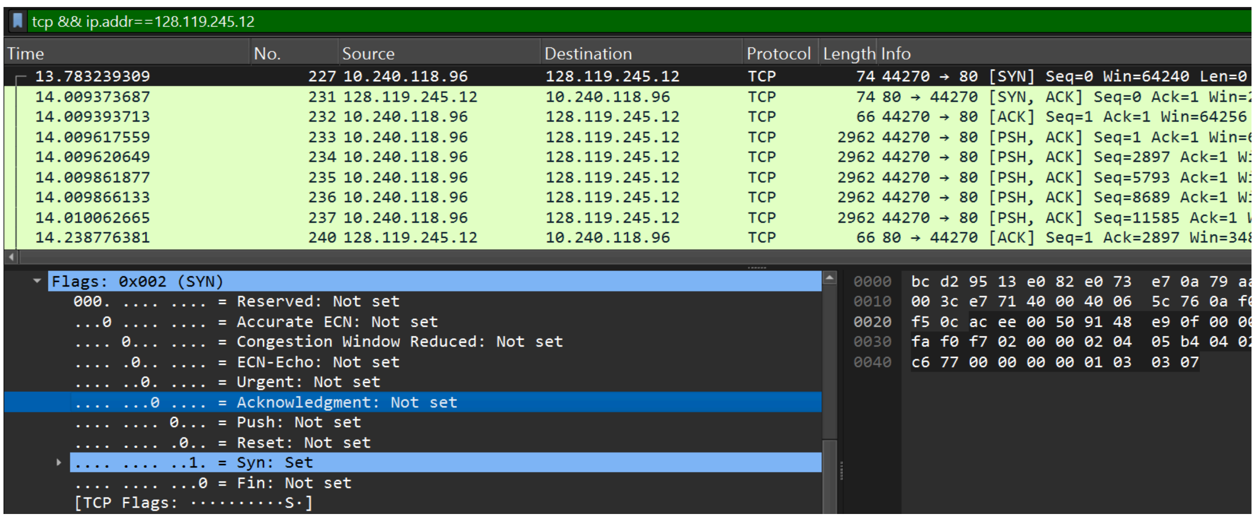


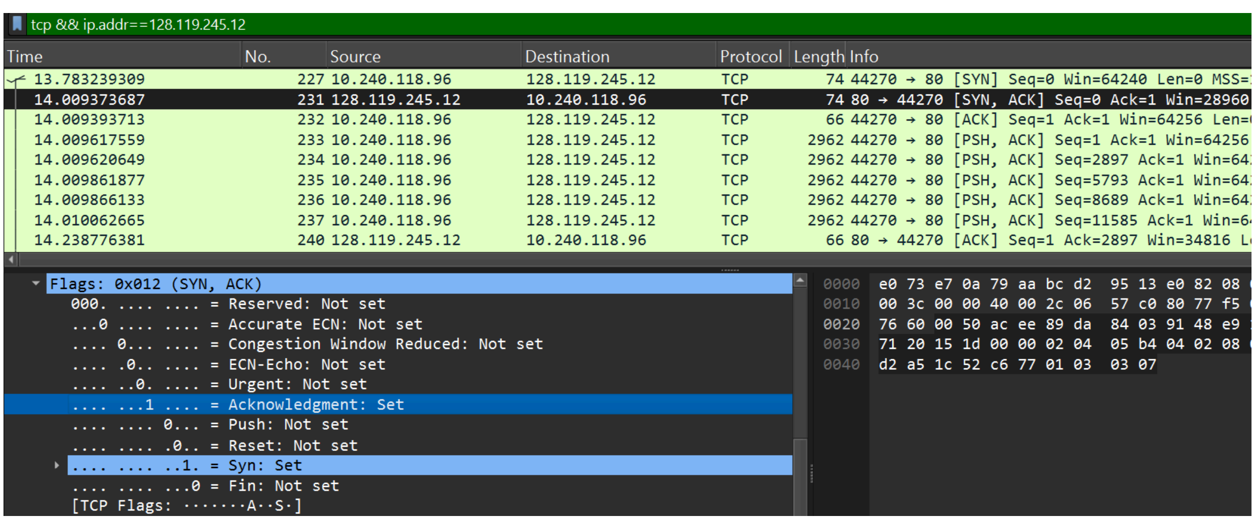
   c. **How many packets are exchanged to establish a TCP connection between your system and the `gaia.cs.umass` website, and what are these packets' specifications?**
   3 packets are exchanged to establish a TCP connection between your system and the `gaia.cs.umass` website.And these packets' specifications are **[SYN], [SYN,ACK], [ACK].**
   d. **What are the `Sequence Number` and `Acknowledgment Number` for the `SYN`, `SYN-ACK`, and `ACK` packets?**

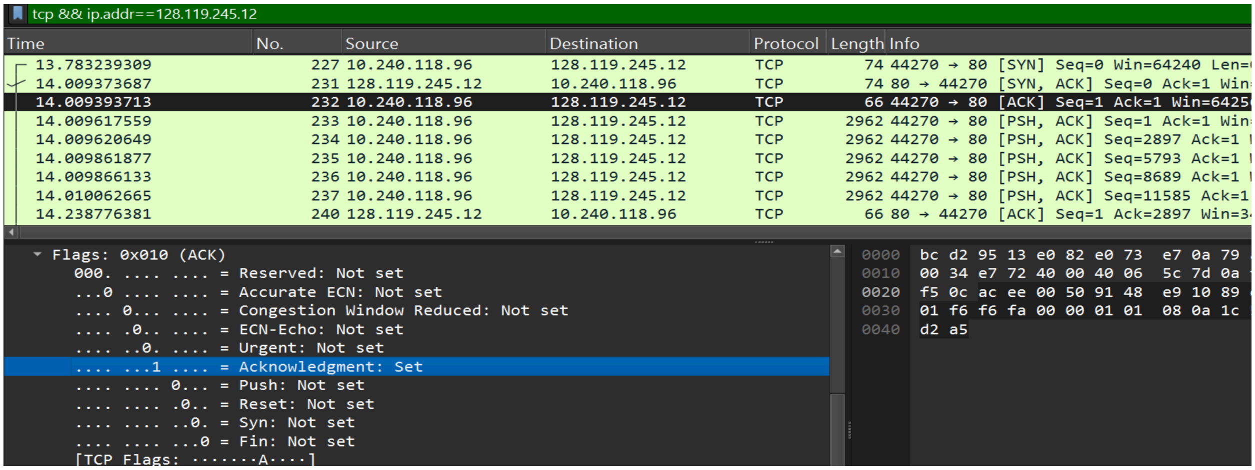
Packet Specification	Sequence Number	Acknowledgment Number
[SYN]	0	0
[SYN-ACK]	0	1
[ACK]	1	1

e. **Which field in the 3 TCP packets – used to establish a connection with the `gaia.cs.umass` website – confirms that the packets are actually the `SYN`, `SYN-ACK`, and `ACK` packets? Provide the screenshots for the same.**

"Flags" field in the 3 TCP packets – used to establish a connection with the **`gaia.cs.umass`** website – confirms that the packets are actually the **`SYN`**, **`SYN-ACK`**, and **`ACK`** packets.In Wireshark, this is shown as Flags under TCP Protocol details.

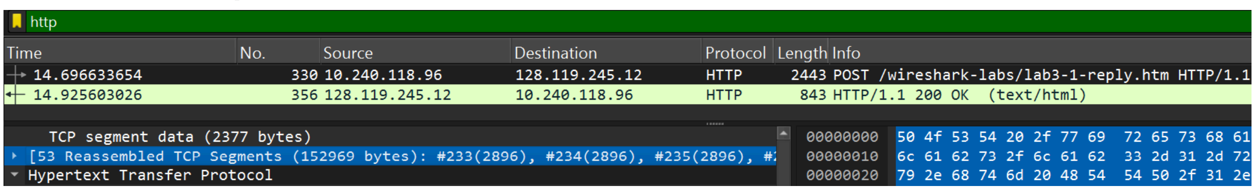






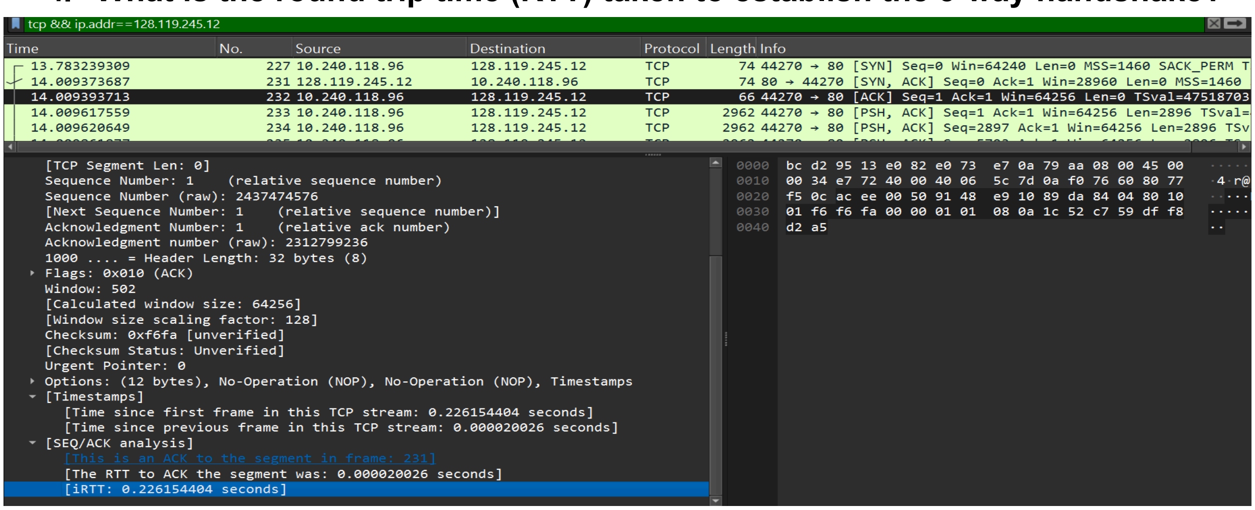
Packet Specification	SYN (Written under the FLAGS section)	Acknowledgment (Written under the FLAGS section)
[SYN]	1(Set)	0(Not set)
[SYN-ACK]	1(Set)	1(Set)
[ACK]	0(Not set)	1(Set)

f. How many reassembled TCP segments are present for the first HTTP request?



53 reassembled TCP segments are present for the first HTTP request.

4. What is the round trip time (RTT) taken to establish the 3-way handshake?



From the image,

First **SYN** Timestamp (T_syn)=13.783239309

Final **ACK** Timestamp (T_ack)=14.009393713

iRTT (Initial Round Trip Time) is displayed in the TCP analysis section as-

[iRTT: 0.226154404 seconds]

iRTT (Initial Round Trip Time) is the time taken for the first round trip in a TCP connection, measured during the 3-way handshake.

1. Time taken for **SYN** to reach the server and **SYN-ACK** to return =T_(syn-ack)-T_syn
2. Time taken for **ACK** to reach the server after **SYN-ACK** is receive =T_ack-T_(syn-ack)

iRTT=[T_(syn-ack)-T_syn]+[T_ack-T_(syn-ack)]=T_ack−Tsyn

=14.009393713-13.783239309

=0.226154404

Round trip time (RTT) taken to establish the 3-way handshake=0.226154404 seconds

Part 2: Analysing the POST packet contents

Answer the following questions based on the traces in the HTTP POST request and its corresponding response packets.

1. **What does the `Reassembled TCP Segments` field in the HTTP POST packet indicate about how the file was uploaded?**
   - The **"Reassembled TCP Segments"** field in an HTTP **POST** packet indicates that the uploaded file was divided into multiple **TCP segments** during transmission and later reassembled at the destination.
   - Since large files often exceed the **Maximum Transmission Unit (MTU)** size, TCP breaks them into smaller segments to ensure reliable delivery. Each segment is assigned a sequence number, allowing the receiving system to reconstruct the original file correctly.
   - Wireshark displays this field in the last TCP segment that completes the HTTP request, showing how the data was transmitted in multiple packets and then reassembled.
   - This ensures that the entire file is received accurately despite network constraints. The process guarantees reliable data transfer while maintaining integrity throughout the transmission.
2. **Why do the first and last segments matter in verifying the integrity of the file uploaded? Does it contain any information regarding the file?**

First Segment:

- The first segment matters because it indicates the **start of the transmission**. It contains crucial metadata like the HTTP headers and may contain the first part of the actual file data.
- If the first segment is missing or corrupted, it would mean the file transfer didn't start correctly, and the server would not receive any data or would fail to begin processing the file.

Last Segment:

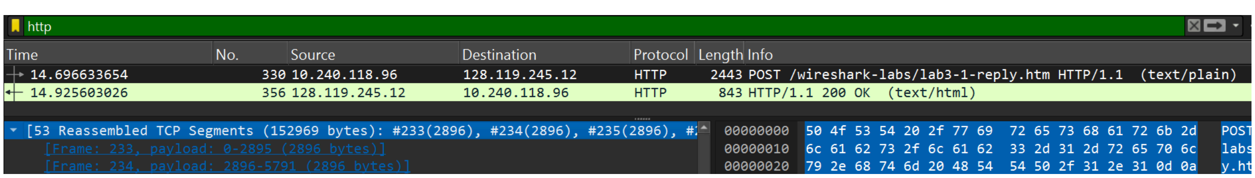
- The last segment indicates the **end of the transmission**. It typically contains the final bytes of the file and may also include important flags (such as EOF or end-of-file indicators).
- The last segment's sequence number confirms the **total amount of data received**, ensuring the file was uploaded completely. If it's missing or incomplete, the file may be considered truncated or corrupted.

Both the first and last segments are used to verify the **integrity and completeness** of the file upload. While they may not contain the entire file data, they play critical roles in signaling the beginning and end of the file transmission.

3. **In which TCP segment can you find the beginning of the actual file contents?**
   When a file is uploaded over TCP, the process begins with the **TCP three-way handshake**, which establishes a reliable connection between the client and the server. Once the connection is established, the actual file upload starts.

- Since the file size often **exceeds the Maximum Transmission Unit (MTU)**, TCP **splits the file into smaller segments** for transmission.
- The **beginning of the actual file contents** appears in the **first TCP segment that carries the file data**, which follows the **HTTP headers** in an **HTTP POST request**.
- In **Wireshark**, this segment can be identified by:
  - Finding the **first TCP segment** that contains the **HTTP message body** (after the headers).
  - Looking for the **sequence number** indicating the start of the file data.
  - Checking the **TCP stream reassembly** to identify the frame where the actual file contents begin.

The beginning of the actual file contents can be found in **the first TCP segment that follows the HTTP headers and contains the file payload**. In Wireshark, this corresponds to the first TCP segment with file data, often identified within the **reassembled TCP stream**. In this specific case, it is **Frame 233**, which is the **first segment of the reassembled file data.**



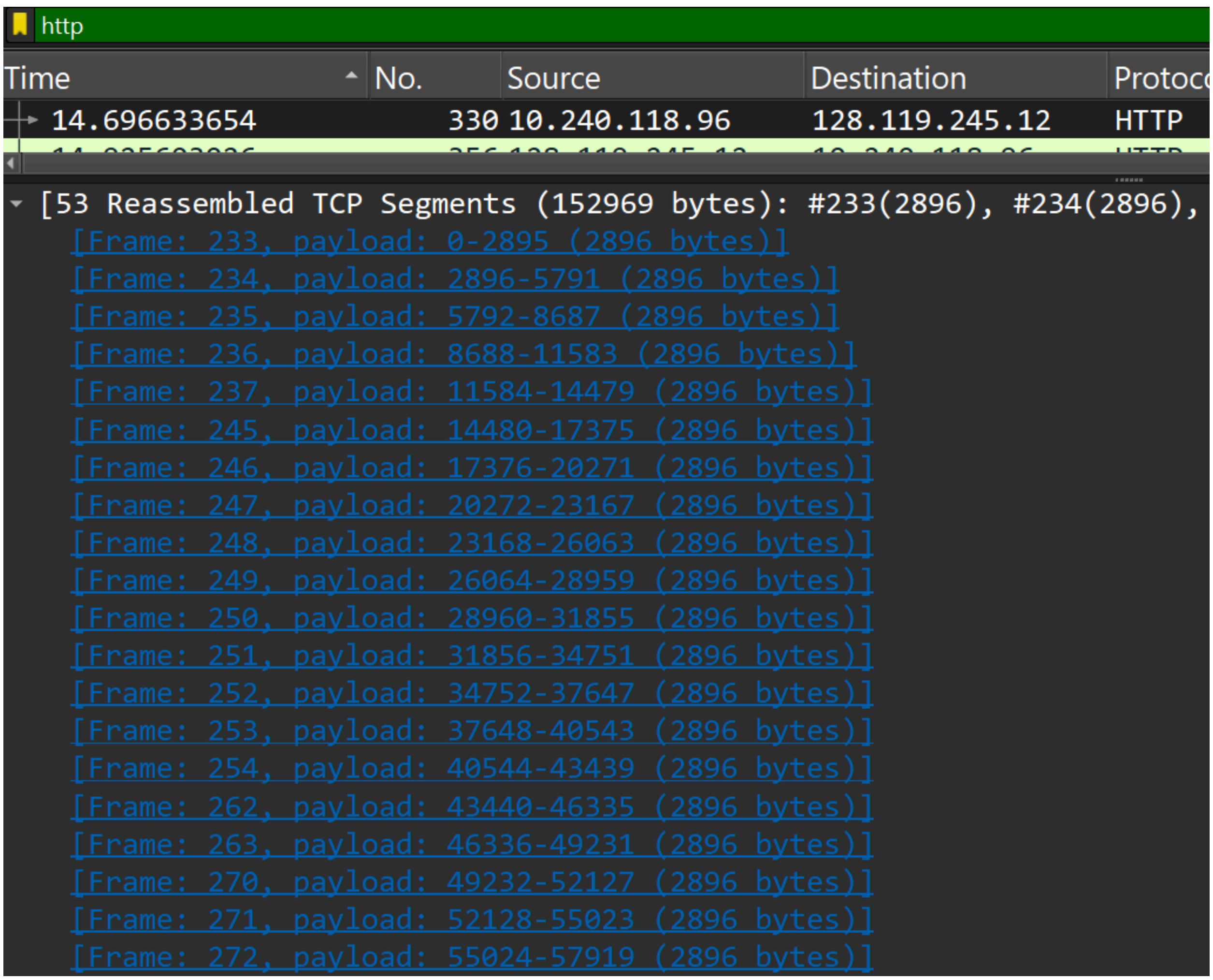
4. **What is the total size of the Reassembled TCP segments, and what does it represent?**
   Total size of the Reassembled TCP segments=**152,969 bytes.**

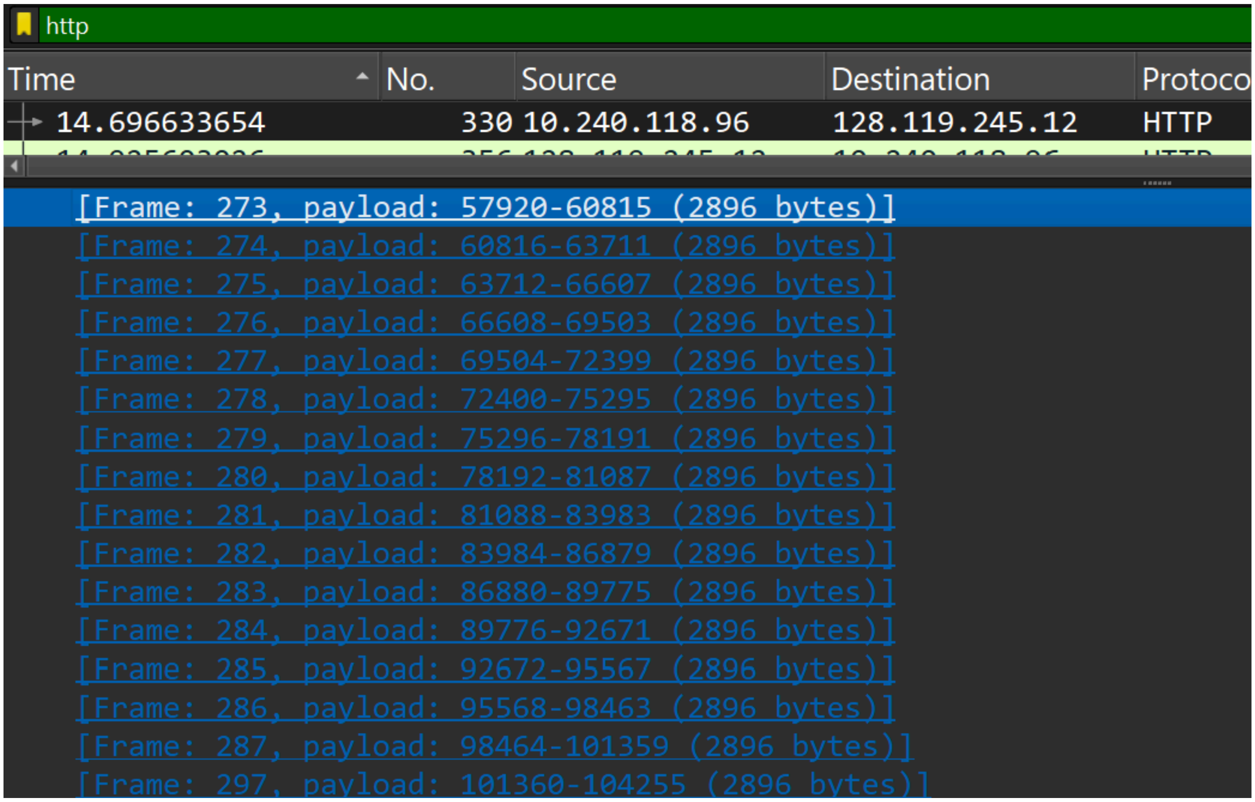
- This represents the **total size of the file** being uploaded, after all the individual TCP segments are reassembled by Wireshark.
- Since Wireshark is reassembling the individual TCP segments into a complete stream, this size represents the entire content of the file (e.g., the `alice.txt` file) that was transferred in the HTTP POST request.

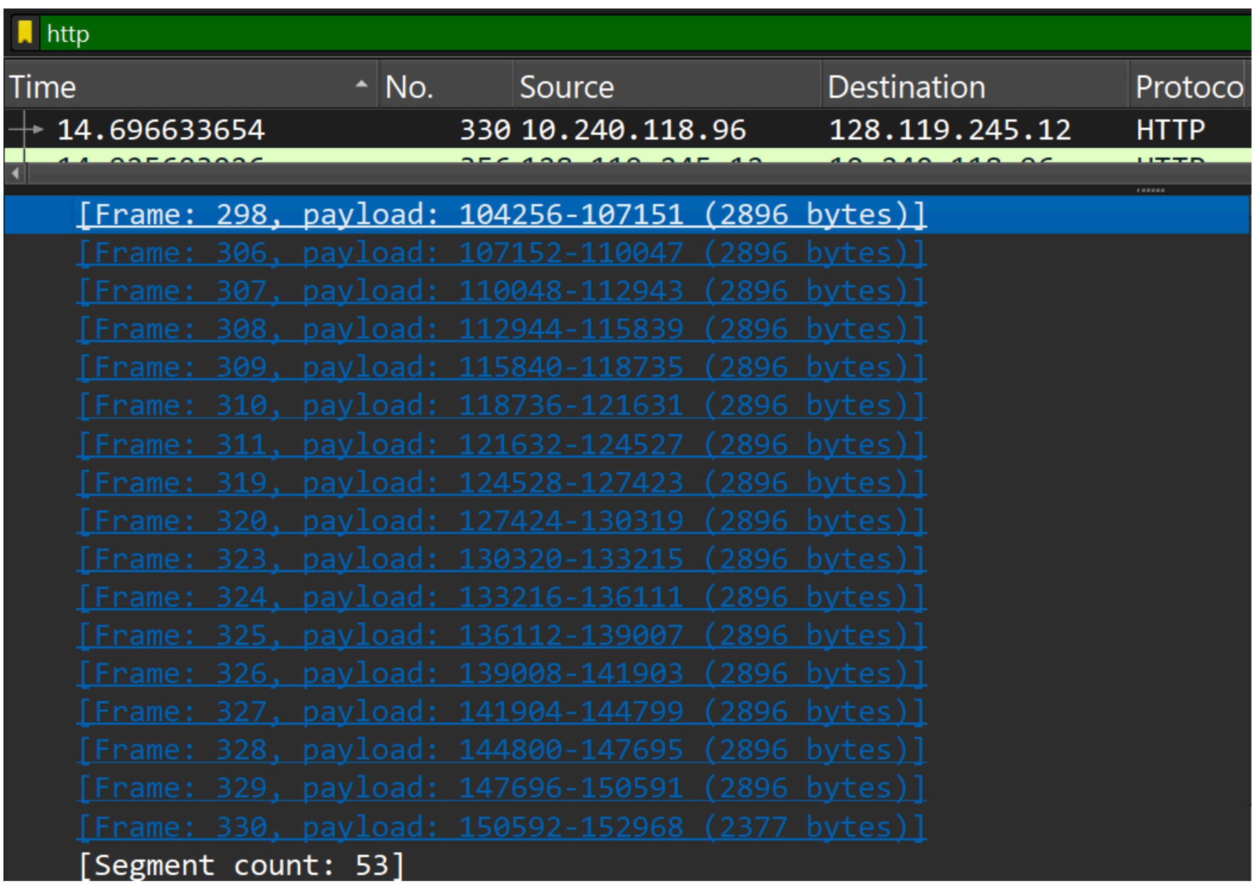
5. **What is the length of each TCP segment?**

   The length of the first 52 segments is **2896 bytes** and the last segment is **2377 bytes.**

- The reassembled TCP segments are indicated by the sequence numbers **#233(2896), #234(2896), #235(2896)**, and so on, where **2896 bytes** is the size of each segment.
- This means each TCP packet contains **2896 bytes** of the file's data, and they are sequentially numbered as part of the overall upload.



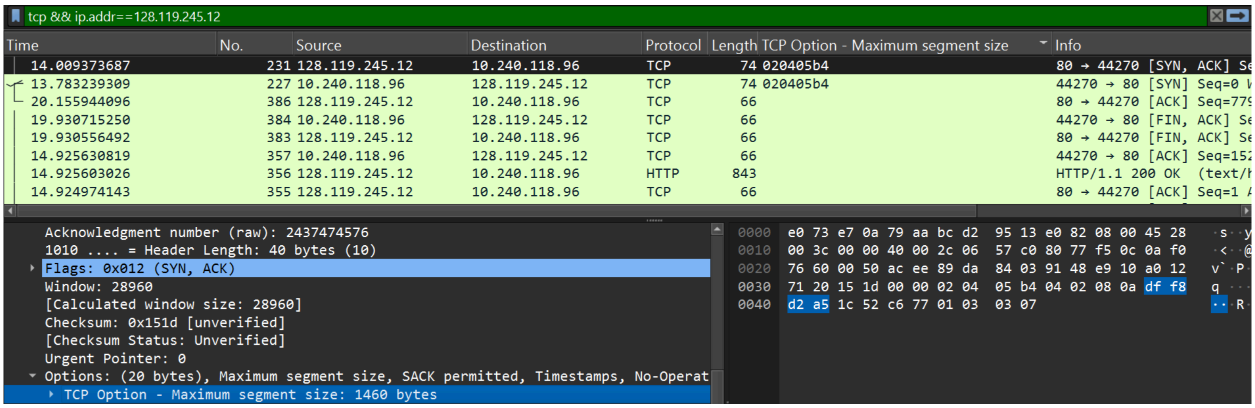

http
Time
No.
Source
Destination
Protoco
14.696633654
330 10.240.118.96
128.119.245.12
HTTP
[Frame: 273, payload: 57920-60815 (2896 bytes)]
[Frame: 274, payload: 60816-63711 (2896 bytes)]
[Frame: 275, payload: 63712-66607 (2896 bytes)]
[Frame: 276, payload: 66608-69503 (2896 bytes)]
[Frame: 277, payload: 69504-72399 (2896 bytes)]
[Frame: 278, payload: 72400-75295 (2896 bytes)]
[Frame: 279, payload: 75296-78191 (2896 bytes)]
[Frame: 280, payload: 78192-81087 (2896 bytes)]
[Frame: 281, payload: 81088-83983 (2896 bytes)]
[Frame: 282, payload: 83984-86879 (2896 bytes)]
[Frame: 283, payload: 86880-89775 (2896 bytes)]
[Frame: 284, payload: 89776-92671 (2896 bytes)]
[Frame: 285, payload: 92672-95567 (2896 bytes)]
[Frame: 286, payload: 95568-98463 (2896 bytes)]
[Frame: 287, payload: 98464-101359 (2896 bytes)]
[Frame: 297, payload: 101360-104255 (2896 bytes)]


http
Time
No.
Source
Destination
Protoco
14.696633654
330 10.240.118.96
128.119.245.12
HTTP
[Frame: 298, payload: 104256-107151 (2896 bytes)]
[Frame: 306, payload: 107152-110047 (2896 bytes)]
[Frame: 307, payload: 110048-112943 (2896 bytes)]
[Frame: 308, payload: 112944-115839 (2896 bytes)]
[Frame: 309, payload: 115840-118735 (2896 bytes)]
[Frame: 310, payload: 118736-121631 (2896 bytes)]
[Frame: 311, payload: 121632-124527 (2896 bytes)]
[Frame: 319, payload: 124528-127423 (2896 bytes)]
[Frame: 320, payload: 127424-130319 (2896 bytes)]
[Frame: 323, payload: 130320-133215 (2896 bytes)]
[Frame: 324, payload: 133216-136111 (2896 bytes)]
[Frame: 325, payload: 136112-139007 (2896 bytes)]
[Frame: 326, payload: 139008-141903 (2896 bytes)]
[Frame: 327, payload: 141904-144799 (2896 bytes)]
[Frame: 328, payload: 144800-147695 (2896 bytes)]
[Frame: 329, payload: 147696-150591 (2896 bytes)]
[Frame: 330, payload: 150592-152968 (2377 bytes)]
[Segment count: 53]

Part 3: Analysing the window scaling

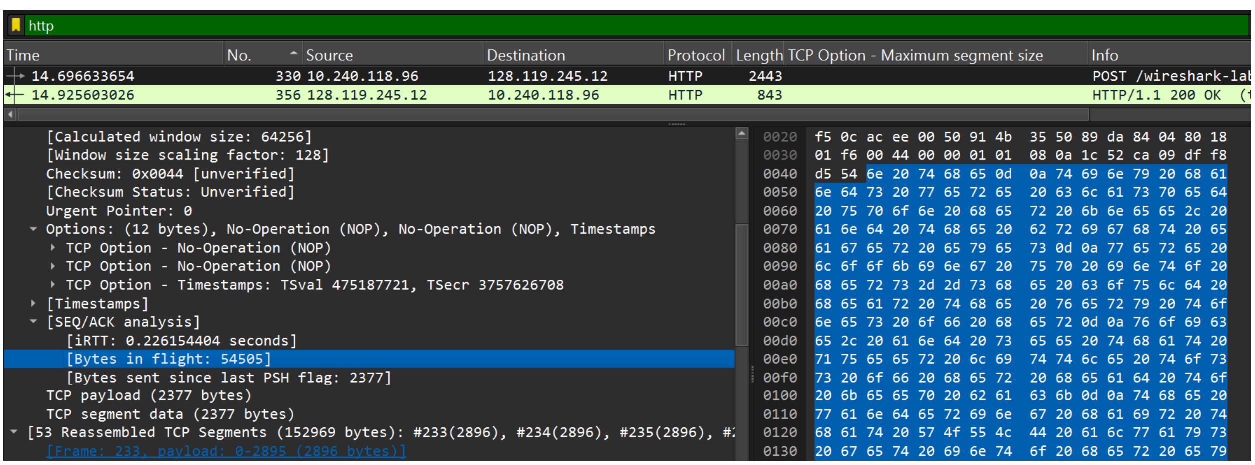
1. **What is the Maximum Segment Size value considered in this connection? (HINT: You will find this information in one of the packets of the 3-way handshake)**

Instruction: Select the following two fields in any of the packets of the TCP stream and right-click and select “Add as column” option, these two fields will be visible as columns in the Wireshark.



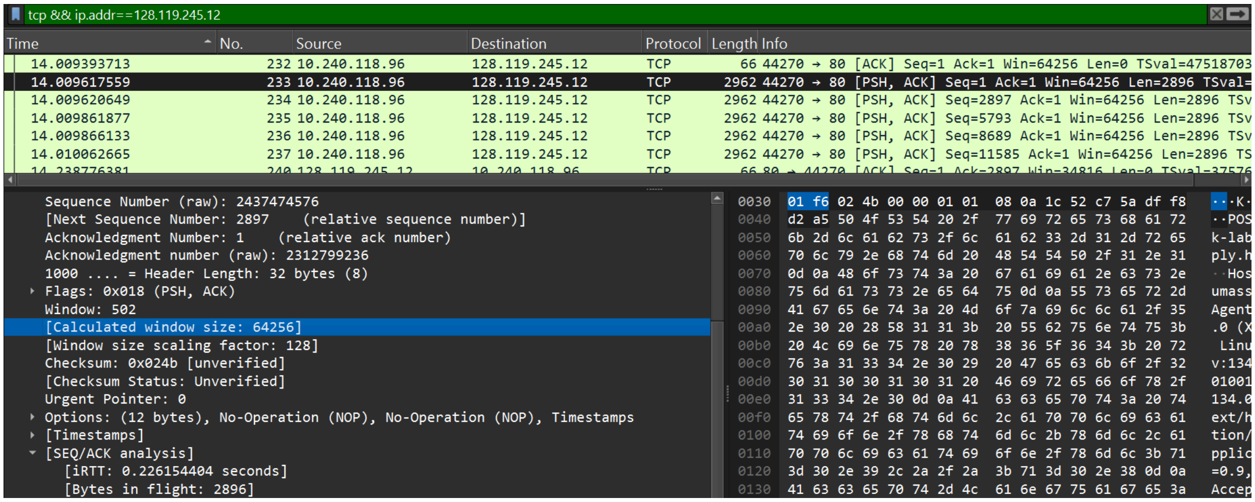
Maximum Segment Size value considered in this connection=**1460 bytes.**

2. **What is the value of the `Calculated window size` and the `Bytes in flight` for the `HTTP POST` packet (the last TCP segment)? Does any of these two field values change in comparison to the first TCP segment?**



The `HTTP POST` packet (the last TCP segment):- Frame 330

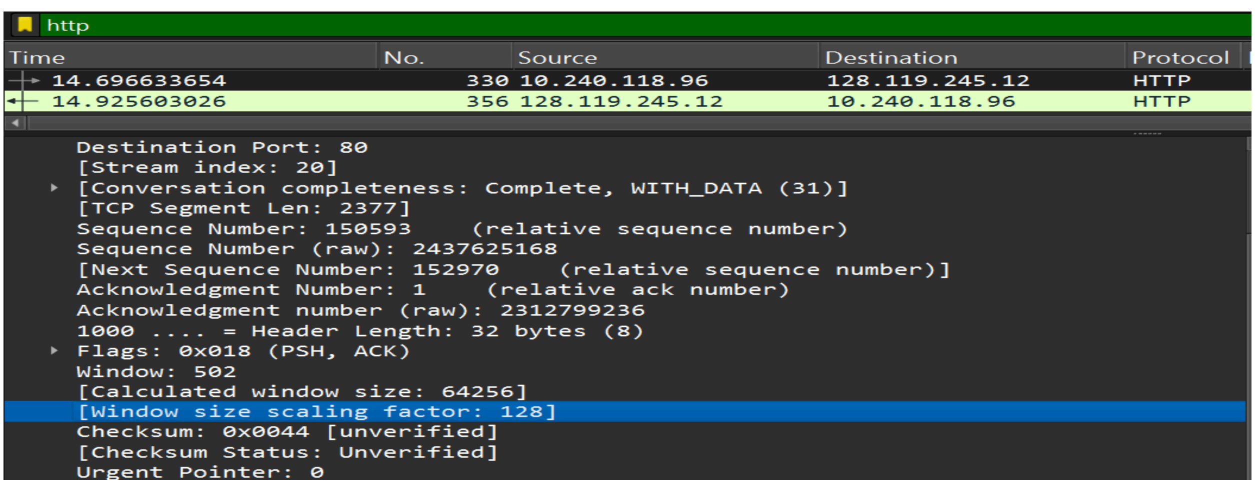
The first TCP segment :- Frame 233



TCP Segment	Calculated window size	Bytes in flight
HTTP POST packet (the last TCP segment)	64256	54505
first TCP segment	64256	2896

→The value of **Calculated window size** for **`HTTP POST`** packet (the last TCP segment) & first TCP segment are not changed. This indicates that the receiver's advertised buffer space did not change throughout the transmission.
→But, the value of Bytes in flight are changed in comparison to the first TCP segment & the last TCP segment.Because the **Bytes in Flight increases**, reflecting the accumulation of unacknowledged data as the HTTP POST request continues.

3. What is the value of the `Window size scaling factor`?



The value of the Window size scaling factor=128.