

## 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
```

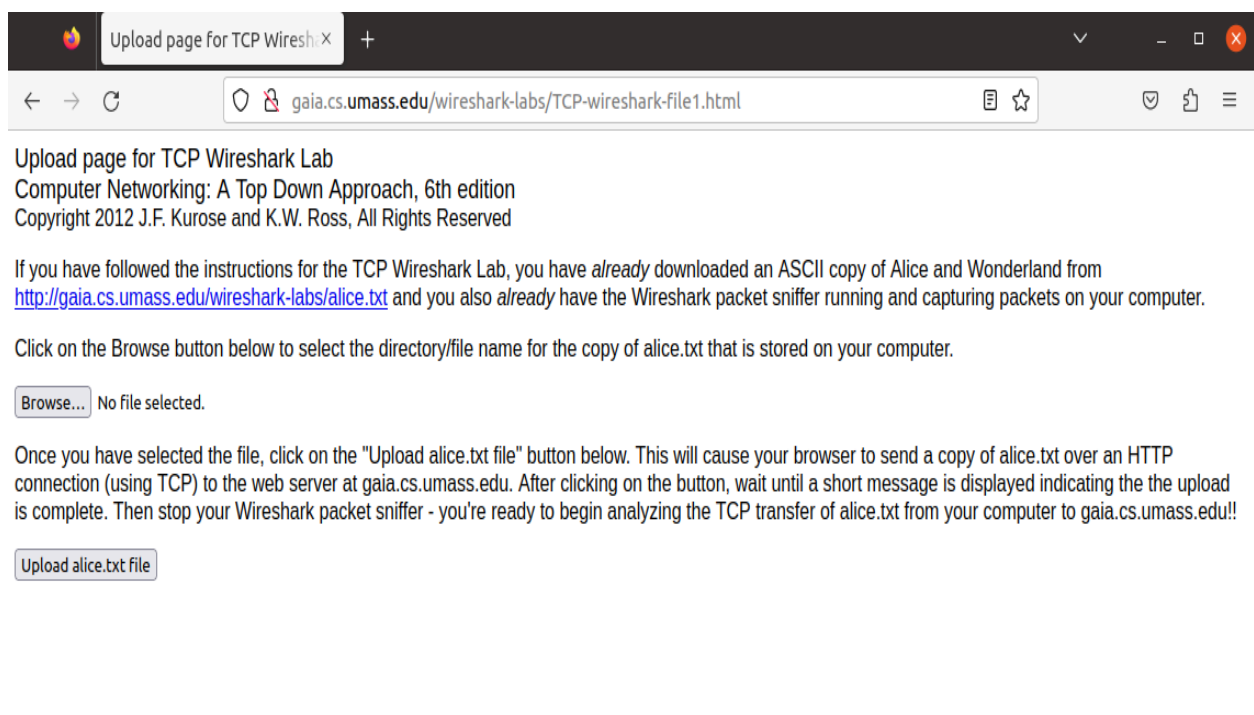
### Part 1: Capturing a bulk TCP transfer from your computer to a remote server

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

Time	No.	Source	Destination	Protocol	Length	Info
14.696633654	330	10.240.118.96	128.119.245.12	HTTP	2443	POST /wireshark-labs/lab3-1-reply.htm HTTP/1.1 (text/plain)
14.925603026	356	128.119.245.12	10.240.118.96	HTTP	843	HTTP/1.1 200 OK (text/html)

Transmission Control Protocol, Src Port: 44270, Dst Port: 80, Seq: 150593, Ack: 1, Source Port: 44270				0020	f5 0c bc ee 00 50 91 4b 35 50 89 da 84 04 80 18	...P
Destination Port: 80				0030	01 f6 00 44 00 00 01 01 08 0a 1c 52 ca 09 df f8	...D...
				0040	d5 54 6e 20 74 68 65 0d 0a 74 69 6e 79 20 68 61	...Tn th

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

Time	No.	Source	Destination	Protocol	Length	Info
13.783239309	227	10.240.118.96	128.119.245.12	TCP	74	44270 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM T
14.009373687	231	128.119.245.12	10.240.118.96	TCP	74	80 → 44270 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460
14.009393713	232	10.240.118.96	128.119.245.12	TCP	66	44270 → 80 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=47518703

- 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.

tcp && ip.addr==128.119.245.12

Time	No.	Source	Destination	Protocol	Length	Info
13.783239309	227	10.240.118.96	128.119.245.12	TCP	74	44270 → 80 [SYN] Seq=0 Win=64240 Len=0
14.009373687	231	128.119.245.12	10.240.118.96	TCP	74	80 → 44270 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0
14.009393713	232	10.240.118.96	128.119.245.12	TCP	66	44270 → 80 [ACK] Seq=1 Ack=1 Win=64256 Len=0
14.009617559	233	10.240.118.96	128.119.245.12	TCP	2962	44270 → 80 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=0
14.009620649	234	10.240.118.96	128.119.245.12	TCP	2962	44270 → 80 [PSH, ACK] Seq=2897 Ack=1 Win=64256 Len=0
14.009861877	235	10.240.118.96	128.119.245.12	TCP	2962	44270 → 80 [PSH, ACK] Seq=5793 Ack=1 Win=64256 Len=0
14.009866133	236	10.240.118.96	128.119.245.12	TCP	2962	44270 → 80 [PSH, ACK] Seq=8689 Ack=1 Win=64256 Len=0
14.010062665	237	10.240.118.96	128.119.245.12	TCP	2962	44270 → 80 [PSH, ACK] Seq=11585 Ack=1 Win=64256 Len=0
14.238776381	240	128.119.245.12	10.240.118.96	TCP	66	80 → 44270 [ACK] Seq=1 Ack=2897 Win=34816 Len=0

Flags: 0x002 (SYN)

- 000. .... = Reserved: Not set
- ...0 .... = Accurate ECN: Not set
- ....0... = Congestion Window Reduced: Not set
- ....0... = ECN-Echo: Not set
- ....0... = Urgent: Not set
- ....0... = Acknowledgment: Not set
- ....0... = Push: Not set
- ....0... = Reset: Not set
- ....1... = Syn: Set
- ....0... = Fin: Not set

[TCP Flags: .....S.]

tcp && ip.addr==128.119.245.12

Time	No.	Source	Destination	Protocol	Length	Info
13.783239309	227	10.240.118.96	128.119.245.12	TCP	74	44270 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=
14.009373687	231	128.119.245.12	10.240.118.96	TCP	74	80 → 44270 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0
14.009393713	232	10.240.118.96	128.119.245.12	TCP	66	44270 → 80 [ACK] Seq=1 Ack=1 Win=64256 Len=0
14.009617559	233	10.240.118.96	128.119.245.12	TCP	2962	44270 → 80 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=0
14.009620649	234	10.240.118.96	128.119.245.12	TCP	2962	44270 → 80 [PSH, ACK] Seq=2897 Ack=1 Win=64256 Len=0
14.009861877	235	10.240.118.96	128.119.245.12	TCP	2962	44270 → 80 [PSH, ACK] Seq=5793 Ack=1 Win=64256 Len=0
14.009866133	236	10.240.118.96	128.119.245.12	TCP	2962	44270 → 80 [PSH, ACK] Seq=8689 Ack=1 Win=64256 Len=0
14.010062665	237	10.240.118.96	128.119.245.12	TCP	2962	44270 → 80 [PSH, ACK] Seq=11585 Ack=1 Win=64256 Len=0
14.238776381	240	128.119.245.12	10.240.118.96	TCP	66	80 → 44270 [ACK] Seq=1 Ack=2897 Win=34816 Len=0

Flags: 0x012 (SYN, ACK)

- 000. .... = Reserved: Not set
- ...0 .... = Accurate ECN: Not set
- ....0... = Congestion Window Reduced: Not set
- ....0... = ECN-Echo: Not set
- ....0... = Urgent: Not set
- ....1... = Acknowledgment: Set
- ....0... = Push: Not set
- ....0... = Reset: Not set
- ....1... = Syn: Set
- ....0... = Fin: Not set

[TCP Flags: .....A..S.]

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?**

Time	No.	Source	Destination	Protocol	Length	Info
14.6966633654	330	10.240.118.96	128.119.245.12	HTTP	2443	POST /wireshark-labs/lab3-1-reply.htm HTTP/1.1
14.925603026	356	128.119.245.12	10.240.118.96	HTTP	843	HTTP/1.1 200 OK (text/html)

TCP segment data (2377 bytes)		00000000	
[53 Reassembled TCP Segments (152969 bytes): #233(2896), #234(2896), #235(2896), #236(2896), #237(2896), #238(2896), #239(2896), #240(2896), #241(2896), #242(2896), #243(2896), #244(2896), #245(2896), #246(2896), #247(2896), #248(2896), #249(2896), #250(2896), #251(2896), #252(2896), #253(2896), #254(2896), #255(2896), #256(2896), #257(2896), #258(2896), #259(2896), #260(2896), #261(2896), #262(2896), #263(2896), #264(2896), #265(2896), #266(2896), #267(2896), #268(2896), #269(2896), #270(2896), #271(2896), #272(2896), #273(2896), #274(2896), #275(2896), #276(2896), #277(2896), #278(2896), #279(2896), #280(2896), #281(2896), #282(2896), #283(2896), #284(2896), #285(2896), #286(2896), #287(2896), #288(2896), #289(2896), #290(2896), #291(2896), #292(2896), #293(2896), #294(2896), #295(2896), #296(2896), #297(2896), #298(2896), #299(2896), #300(2896), #301(2896), #302(2896), #303(2896), #304(2896), #305(2896), #306(2896), #307(2896), #308(2896), #309(2896), #310(2896), #311(2896), #312(2896), #313(2896), #314(2896), #315(2896), #316(2896), #317(2896), #318(2896), #319(2896), #320(2896), #321(2896), #322(2896), #323(2896), #324(2896), #325(2896), #326(2896), #327(2896), #328(2896), #329(2896), #330(2896), #331(2896), #332(2896), #333(2896), #334(2896), #335(2896), #336(2896), #337(2896), #338(2896), #339(2896), #340(2896), #341(2896), #342(2896), #343(2896), #344(2896), #345(2896), #346(2896), #347(2896), #348(2896), #349(2896), #350(2896), #351(2896), #352(2896), #353(2896), #354(2896), #355(2896), #356(2896), #357(2896), #358(2896), #359(2896), #360(2896), #361(2896), #362(2896), #363(2896), #364(2896), #365(2896), #366(2896), #367(2896), #368(2896), #369(2896), #370(2896), #371(2896), #372(2896), #373(2896), #374(2896), #375(2896), #376(2896), #377(2896), #378(2896), #379(2896), #380(2896), #381(2896), #382(2896), #383(2896), #384(2896), #385(2896), #386(2896), #387(2896), #388(2896), #389(2896), #390(2896), #391(2896), #392(2896), #393(2896), #394(2896), #395(2896), #396(2896), #397(2896), #398(2896), #399(2896), #400(2896), #401(2896), #402(2896), #403(2896), #404(2896), #405(2896), #406(2896), #407(2896), #408(2896), #409(2896), #410(2896), #411(2896), #412(2896), #413(2896), #414(2896), #415(2896), #416(2896), #417(2896), #418(2896), #419(2896), #420(2896), #421(2896), #422(2896), #423(2896), #424(2896), #425(2896), #426(2896), #427(2896), #428(2896), #429(2896), #430(2896), #431(2896), #432(2896), #433(2896), #434(2896), #435(2896), #436(2896), #437(2896), #438(2896), #439(2896), #440(2896), #441(2896), #442(2896), #443(2896), #444(2896), #445(2896), #446(2896), #447(2896), #448(2896), #449(2896), #450(2896), #451(2896), #452(2896), #453(2896), #454(2896), #455(2896), #456(2896), #457(2896), #458(2896), #459(2896), #460(2896), #461(2896), #462(2896), #463(2896), #464(2896), #465(2896), #466(2896), #467(2896), #468(2896), #469(2896), #470(2896), #471(2896), #472(2896), #473(2896), #474(2896), #475(2896), #476(2896), #477(2896), #478(2896), #479(2896), #480(2896), #481(2896), #482(2896), #483(2896), #484(2896), #485(2896), #486(2896), #487(2896), #488(2896), #489(2896), #490(2896), #491(2896), #492(2896), #493(2896), #494(2896), #495(2896), #496(2896), #497(2896), #498(2896), #499(2896), #500(2896), #501(2896), #502(2896), #503(2896), #504(2896), #505(2896), #506(2896), #507(2896), #508(2896), #509(2896), #510(2896), #511(2896), #512(2896), #513(2896), #514(2896), #515(2896), #516(2896), #517(2896), #518(2896), #519(2896), #520(2896), #521(2896), #522(2896), #523(2896), #524(2896), #525(2896), #526(2896), #527(2896), #528(2896), #529(2896), #530(2896), #531(2896), #532(2896), #533(2896), #534(2896), #535(2896), #536(2896), #537(2896), #538(2896), #539(2896), #540(2896), #541(2896), #542(2896), #543(2896), #544(2896), #545(2896), #546(2896), #547(2896), #548(2896), #549(2896), #550(2896), #551(2896), #552(2896), #553(2896), #554(2896), #555(2896), #556(2896), #557(2896), #558(2896), #559(2896), #560(2896), #561(2896), #562(2896), #563(2896), #564(2896), #565(2896), #566(2896), #567(2896), #568(2896), #569(2896), #570(2896), #571(2896), #572(2896), #573(2896), #574(2896), #575(2896), #576(2896), #577(2896), #578(2896), #579(2896), #580(2896), #581(2896), #582(2896), #583(2896), #584(2896), #585(2896), #586(2896), #587(2896), #588(2896), #589(2896), #590(2896), #591(2896), #592(2896), #593(2896), #594(2896), #595(2896), #596(2896), #597(2896), #598(2896), #599(2896), #600(2896), #601(2896), #602(2896), #603(28			

**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?**

tcp && ip.addr==128.119.245.12

Time	No.	Source	Destination	Protocol	Length	Info
13.783239309	227	10.240.118.96	128.119.245.12	TCP	74	44270 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM T
14.009373687	231	128.119.245.12	10.240.118.96	TCP	74	80 → 44270 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460
14.009393713	232	10.240.118.96	128.119.245.12	TCP	66	44270 → 80 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=47518703
14.009617559	233	10.240.118.96	128.119.245.12	TCP	2962	44270 → 80 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=2896 TSval=
14.009620649	234	10.240.118.96	128.119.245.12	TCP	2962	44270 → 80 [PSH, ACK] Seq=2897 Ack=1 Win=64256 Len=2896 TSV

[TCP Segment Len: 0]  
Sequence Number: 1 (relative sequence number)  
Sequence Number (raw): 2437474576  
[Next Sequence Number: 1 (relative sequence number)]  
Acknowledgment Number: 1 (relative ack number)  
Acknowledgment number (raw): 2312799236  
1000 .... = Header Length: 32 bytes (8)  
Flags: 0x010 (ACK)  
Window: 502  
[calculated window size: 64256]  
[window size scaling factor: 128]  
Checksum: 0xf6fa [unverified]  
[checksum Status: Unverified]  
Urgent Pointer: 0  
Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps  
[Timestamps]  
[Time since first frame in this TCP stream: 0.226154404 seconds]  
[Time since previous frame in this TCP stream: 0.000020026 seconds]  
[SEQ/ACK analysis]  
[This is an ACK to the segment in frame: 231]  
[The RTT to ACK the segment was: 0.000020026 seconds]  
[RTT: 0.226154404 seconds]

From the image,

First **SYN** Timestamp ( $T_{\text{syn}}$ )=13.783239309

Final **ACK** Timestamp ( $T_{\text{ack}}$ )=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  $=T_{\text{(syn-ack)}}-T_{\text{syn}}$
2. Time taken for **ACK** to reach the server after **SYN-ACK** is received  $=T_{\text{ack}}-T_{\text{(syn-ack)}}$

$$\begin{aligned}\mathbf{iRTT} &= [T_{\text{(syn-ack)}} - T_{\text{syn}}] + [T_{\text{ack}} - T_{\text{(syn-ack)}}] = T_{\text{ack}} - T_{\text{syn}} \\ &= 14.009393713 - 13.783239309 \\ &= 0.226154404\end{aligned}$$

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.

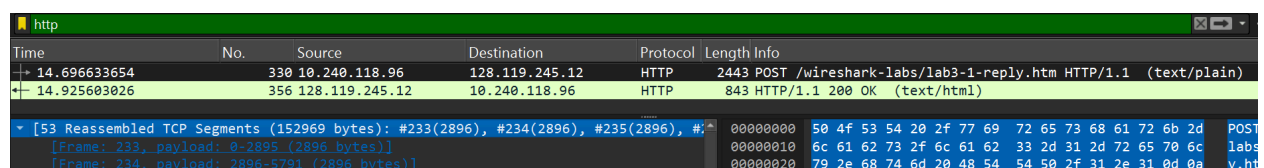
Both the first and last segments are used to verify the **integrity and completeness** of the file upload. While they may not contain the entire file data, they play critical roles in signaling the beginning and end of the file transmission.

### 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**.



Time	No.	Source	Destination	Protocol	Length	Info
14.696633654	330	10.240.118.96	128.119.245.12	HTTP	2443	POST /wireshark-labs/lab3-1-reply.htm HTTP/1.1 (text/plain)
14.925603026	356	128.119.245.12	10.240.118.96	HTTP	843	HTTP/1.1 200 OK (text/html)

[53 Reassembled TCP Segments (152969 bytes): #233(2896), #234(2896), #235(2896), #...	
[Frame: 233, payload: 0-2895 (2896 bytes)]	00000000 50 4f 53 54 20 2f 77 69 72 65 73 68 61 72 6b 2d POST
[Frame: 234, payload: 2896-5791 (2896 bytes)]	00000010 6c 61 62 73 2f 6c 61 62 33 2d 31 2d 72 65 70 6c lab
	00000020 79 2e 68 74 6d 20 48 54 54 50 2f 31 2e 31 0d 0a y.ht

### 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	Protocol
14.696633654	330	10.240.118.96	128.119.245.12	HTTP
14.825692886	356	128.119.245.12	10.240.118.96	HTTP
[53 Reassembled TCP Segments (152969 bytes): #233(2896), #234(2896), <a href="#">[Frame: 233, payload: 0-2895 (2896 bytes)]</a> <a href="#">[Frame: 234, payload: 2896-5791 (2896 bytes)]</a> <a href="#">[Frame: 235, payload: 5792-8687 (2896 bytes)]</a> <a href="#">[Frame: 236, payload: 8688-11583 (2896 bytes)]</a> <a href="#">[Frame: 237, payload: 11584-14479 (2896 bytes)]</a> <a href="#">[Frame: 245, payload: 14480-17375 (2896 bytes)]</a> <a href="#">[Frame: 246, payload: 17376-20271 (2896 bytes)]</a> <a href="#">[Frame: 247, payload: 20272-23167 (2896 bytes)]</a> <a href="#">[Frame: 248, payload: 23168-26063 (2896 bytes)]</a> <a href="#">[Frame: 249, payload: 26064-28959 (2896 bytes)]</a> <a href="#">[Frame: 250, payload: 28960-31855 (2896 bytes)]</a> <a href="#">[Frame: 251, payload: 31856-34751 (2896 bytes)]</a> <a href="#">[Frame: 252, payload: 34752-37647 (2896 bytes)]</a> <a href="#">[Frame: 253, payload: 37648-40543 (2896 bytes)]</a> <a href="#">[Frame: 254, payload: 40544-43439 (2896 bytes)]</a> <a href="#">[Frame: 262, payload: 43440-46335 (2896 bytes)]</a> <a href="#">[Frame: 263, payload: 46336-49231 (2896 bytes)]</a> <a href="#">[Frame: 270, payload: 49232-52127 (2896 bytes)]</a> <a href="#">[Frame: 271, payload: 52128-55023 (2896 bytes)]</a> <a href="#">[Frame: 272, payload: 55024-57919 (2896 bytes)]</a>				



http				
Time	No.	Source	Destination	Protocol
14.696633654	330	10.240.118.96	128.119.245.12	HTTP
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	Protocol
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.

The image shows a Wireshark packet capture of a TCP connection. The packet list pane shows several packets, with packet 14 selected. The packet details pane shows the TCP header and options. The 'TCP Options' field is expanded, showing 'Maximum segment size: 1460 bytes'. The packet bytes pane shows the raw data of the packet.

Time	No.	Source	Destination	Protocol	Length	TCP Option - Maximum segment size	Info
14.009373687	231	128.119.245.12	10.240.118.96	TCP	74	020405b4	80 → 44270 [SYN, ACK] Seq=0
13.783239309	227	10.240.118.96	128.119.245.12	TCP	66		44270 → 80 [SYN] Seq=0
20.155944096	386	128.119.245.12	10.240.118.96	TCP	66		80 → 44270 [ACK] Seq=775
19.930715250	384	10.240.118.96	128.119.245.12	TCP	66		44270 → 80 [FIN, ACK] Seq=15
19.930556492	383	128.119.245.12	10.240.118.96	TCP	66		80 → 44270 [FIN, ACK] Seq=15
14.925630819	357	10.240.118.96	128.119.245.12	TCP	66		44270 → 80 [ACK] Seq=15
14.925603026	356	128.119.245.12	10.240.118.96	HTTP	843		HTTP/1.1 200 OK (text/html)
14.924974143	355	128.119.245.12	10.240.118.96	TCP	66		80 → 44270 [ACK] Seq=1

Packet 14 details:

- Acknowledgment number (raw): 2437474576
- 1010 ... = Header Length: 40 bytes (10)
- Flags: 0x012 (SYN, ACK)
- Window: 28960
- [Calculated window size: 28960]
- Checksum: 0x151d [unverified]
- [Checksum Status: Unverified]
- Urgent Pointer: 0
- Options: (20 bytes), Maximum segment size, SACK permitted, Timestamps, No-Operation
- TCP Option - Maximum segment size: 1460 bytes

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 image shows a Wireshark packet capture of an HTTP POST request. The packet list pane shows two packets, with packet 14 selected. The packet details pane shows the HTTP header and body. The 'HTTP body' field is expanded, showing the raw data of the packet.

Time	No.	Source	Destination	Protocol	Length	TCP Option - Maximum segment size	Info
14.696633654	330	10.240.118.96	128.119.245.12	HTTP	2443		POST /wireshark-lab
14.925603026	356	128.119.245.12	10.240.118.96	HTTP	843		HTTP/1.1 200 OK (text/html)

Packet 14 details:

- [Calculated window size: 64256]
- [Window size scaling factor: 128]
- Checksum: 0x0044 [unverified]
- [Checksum Status: Unverified]
- Urgent Pointer: 0
- Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
- TCP Option - No-Operation (NOP)
- TCP Option - No-Operation (NOP)
- TCP Option - Timestamps: TSval 475187721, TSecr 3757626708
- [Timestamps]
- [SEQ/ACK analysis]
- [RTT: 0.226154404 seconds]
- [Bytes in flight: 54505]
- [Bytes sent since last PSF flag: 2377]
- TCP payload (2377 bytes)
- TCP segment data (2377 bytes)
- [53 Reassembled TCP Segments (152969 bytes): #233(2896), #234(2896), #235(2896), #...
- [Frame 233, payload 0-2895 (2896 bytes)]

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

The first TCP segment :- Frame 233

The image shows a Wireshark packet capture of a TCP connection. The top pane displays a list of packets. The selected packet (No. 232) is a TCP segment from 10.240.118.96 to 128.119.245.12. The details pane shows the following information:

- Sequence Number (raw): 2437474576
- [Next Sequence Number: 2897 (relative sequence number)]
- Acknowledgment Number: 1 (relative ack number)
- Acknowledgment number (raw): 2312799236
- 1000 .... = Header Length: 32 bytes (8)
- Flags: 0x018 (PSH, ACK)
- Window: 502
- [Calculated window size: 64256]
- [Window size scaling factor: 128]
- Checksum: 0x024b [unverified]
- [Checksum Status: Unverified]
- Urgent Pointer: 0
- Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
- [Timestamps]
- [SEQ/ACK analysis]
- [iRTT: 0.226154404 seconds]
- [Bytes in flight: 2896]

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 image shows a Wireshark packet capture of an HTTP connection. The top pane displays a list of packets. The selected packet (No. 356) is an HTTP packet from 128.119.245.12 to 10.240.118.96. The details pane shows the following information:

- Destination Port: 80
- [Stream index: 20]
- [Conversation completeness: Complete, WITH\_DATA (31)]
- [TCP Segment Len: 2377]
- Sequence Number: 150593 (relative sequence number)
- Sequence Number (raw): 2437625168
- [Next Sequence Number: 152970 (relative sequence number)]
- Acknowledgment Number: 1 (relative ack number)
- Acknowledgment number (raw): 2312799236
- 1000 .... = Header Length: 32 bytes (8)
- Flags: 0x018 (PSH, ACK)
- Window: 502
- [Calculated window size: 64256]
- [Window size scaling factor: 128]
- Checksum: 0x0044 [unverified]
- [Checksum Status: Unverified]
- Urgent Pointer: 0

The value of the Window size scaling factor=128.