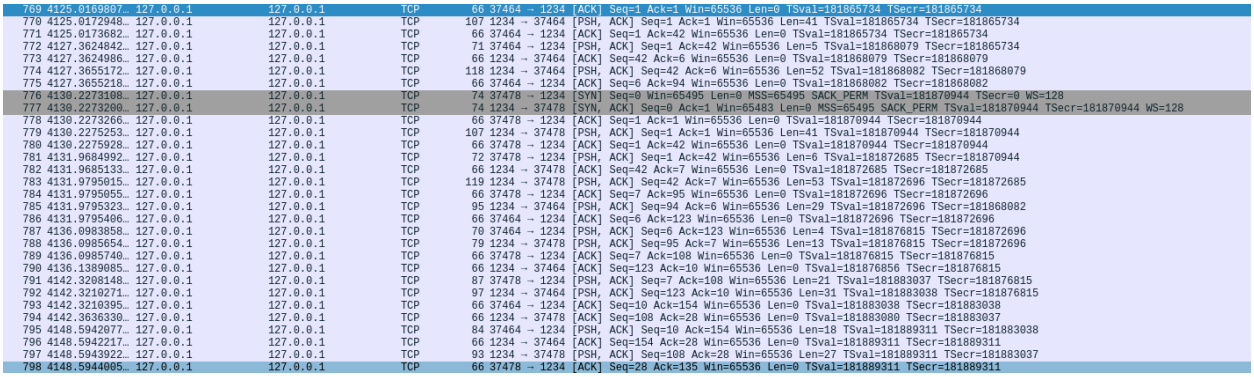
Tirth V Patel (200435378)

Ramanpreet Singh (200384219)

ENSE 472

Lab: 2

Phase 1:



769	4125.0169807…	127.0.0.1	127.0.0.1	TCP	66 37464 → 1234 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=181865734 TSecr=181865734
770	4125.0172948…	127.0.0.1	127.0.0.1	TCP	107 1234 → 37464 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=41 TSval=181865734 TSecr=181865734
771	4125.0173682…	127.0.0.1	127.0.0.1	TCP	66 37464 → 1234 [ACK] Seq=1 Ack=42 Win=65536 Len=0 TSval=181865734 TSecr=181865734
772	4127.3624842…	127.0.0.1	127.0.0.1	TCP	71 37464 → 1234 [PSH, ACK] Seq=1 Ack=42 Win=65536 Len=5 TSval=181868079 TSecr=181865734
773	4127.3624986…	127.0.0.1	127.0.0.1	TCP	66 1234 → 37464 [ACK] Seq=42 Ack=6 Win=65536 Len=0 TSval=181868079 TSecr=181868079
774	4127.3655172…	127.0.0.1	127.0.0.1	TCP	118 1234 → 37464 [PSH, ACK] Seq=42 Ack=6 Win=65536 Len=52 TSval=181868082 TSecr=181868079
775	4127.3655218…	127.0.0.1	127.0.0.1	TCP	66 37464 → 1234 [ACK] Seq=6 Ack=94 Win=65536 Len=0 TSval=181868082 TSecr=181868082
776	4130.2273108…	127.0.0.1	127.0.0.1	TCP	74 37478 → 1234 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=181870944 TSecr=0 WS=128
777	4130.2273200…	127.0.0.1	127.0.0.1	TCP	74 1234 → 37478 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM TSval=181870944 TSecr=181870944 WS=128
778	4130.2273266…	127.0.0.1	127.0.0.1	TCP	66 37478 → 1234 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=181870944 TSecr=181870944
779	4130.2275253…	127.0.0.1	127.0.0.1	TCP	107 1234 → 37478 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=41 TSval=181870944 TSecr=181870944
780	4130.2275928…	127.0.0.1	127.0.0.1	TCP	66 37478 → 1234 [ACK] Seq=1 Ack=42 Win=65536 Len=0 TSval=181870944 TSecr=181870944
781	4131.9684992…	127.0.0.1	127.0.0.1	TCP	72 37478 → 1234 [PSH, ACK] Seq=1 Ack=42 Win=65536 Len=6 TSval=181872685 TSecr=181870944
782	4131.9685133…	127.0.0.1	127.0.0.1	TCP	66 1234 → 37478 [ACK] Seq=42 Ack=7 Win=65536 Len=0 TSval=181872685 TSecr=181872685
783	4131.9795015…	127.0.0.1	127.0.0.1	TCP	119 1234 → 37478 [PSH, ACK] Seq=42 Ack=7 Win=65536 Len=53 TSval=181872696 TSecr=181872685
784	4131.9795055…	127.0.0.1	127.0.0.1	TCP	66 37478 → 1234 [ACK] Seq=7 Ack=95 Win=65536 Len=0 TSval=181872696 TSecr=181872696
785	4131.9795323…	127.0.0.1	127.0.0.1	TCP	95 1234 → 37464 [PSH, ACK] Seq=94 Ack=6 Win=65536 Len=29 TSval=181872696 TSecr=181868082
786	4131.9795406…	127.0.0.1	127.0.0.1	TCP	66 37464 → 1234 [ACK] Seq=6 Ack=123 Win=65536 Len=0 TSval=181872696 TSecr=181872696
787	4136.0983858…	127.0.0.1	127.0.0.1	TCP	70 37464 → 1234 [PSH, ACK] Seq=6 Ack=123 Win=65536 Len=4 TSval=181876815 TSecr=181872696
788	4136.0985654…	127.0.0.1	127.0.0.1	TCP	79 1234 → 37478 [PSH, ACK] Seq=95 Ack=7 Win=65536 Len=13 TSval=181876815 TSecr=181872696
789	4136.0985740…	127.0.0.1	127.0.0.1	TCP	66 37478 → 1234 [ACK] Seq=7 Ack=108 Win=65536 Len=0 TSval=181876815 TSecr=181876815
790	4136.1389085…	127.0.0.1	127.0.0.1	TCP	66 1234 → 37464 [ACK] Seq=123 Ack=10 Win=65536 Len=0 TSval=181876856 TSecr=181876815
791	4142.3208148…	127.0.0.1	127.0.0.1	TCP	87 37478 → 1234 [PSH, ACK] Seq=7 Ack=108 Win=65536 Len=21 TSval=181883037 TSecr=181876815
792	4142.3210271…	127.0.0.1	127.0.0.1	TCP	97 1234 → 37464 [PSH, ACK] Seq=123 Ack=10 Win=65536 Len=31 TSval=181883038 TSecr=181876815
793	4142.3210395…	127.0.0.1	127.0.0.1	TCP	66 37464 → 1234 [ACK] Seq=10 Ack=154 Win=65536 Len=0 TSval=181883038 TSecr=181883038
794	4142.3636330…	127.0.0.1	127.0.0.1	TCP	66 1234 → 37478 [ACK] Seq=108 Ack=28 Win=65536 Len=0 TSval=181883080 TSecr=181883037
795	4148.5942077…	127.0.0.1	127.0.0.1	TCP	84 37464 → 1234 [PSH, ACK] Seq=10 Ack=154 Win=65536 Len=18 TSval=181889311 TSecr=181883038
796	4148.5942217…	127.0.0.1	127.0.0.1	TCP	66 1234 → 37464 [ACK] Seq=154 Ack=28 Win=65536 Len=0 TSval=181889311 TSecr=181889311
797	4148.5943922…	127.0.0.1	127.0.0.1	TCP	93 1234 → 37478 [PSH, ACK] Seq=108 Ack=28 Win=65536 Len=27 TSval=181889311 TSecr=181883037
798	4148.5944005…	127.0.0.1	127.0.0.1	TCP	66 37478 → 1234 [ACK] Seq=28 Ack=135 Win=65536 Len=0 TSval=181889311 TSecr=181889311

This capture shows the TCP 3-way handshake between the server and the client.

Packets **776-778** represent the handshake sequence:

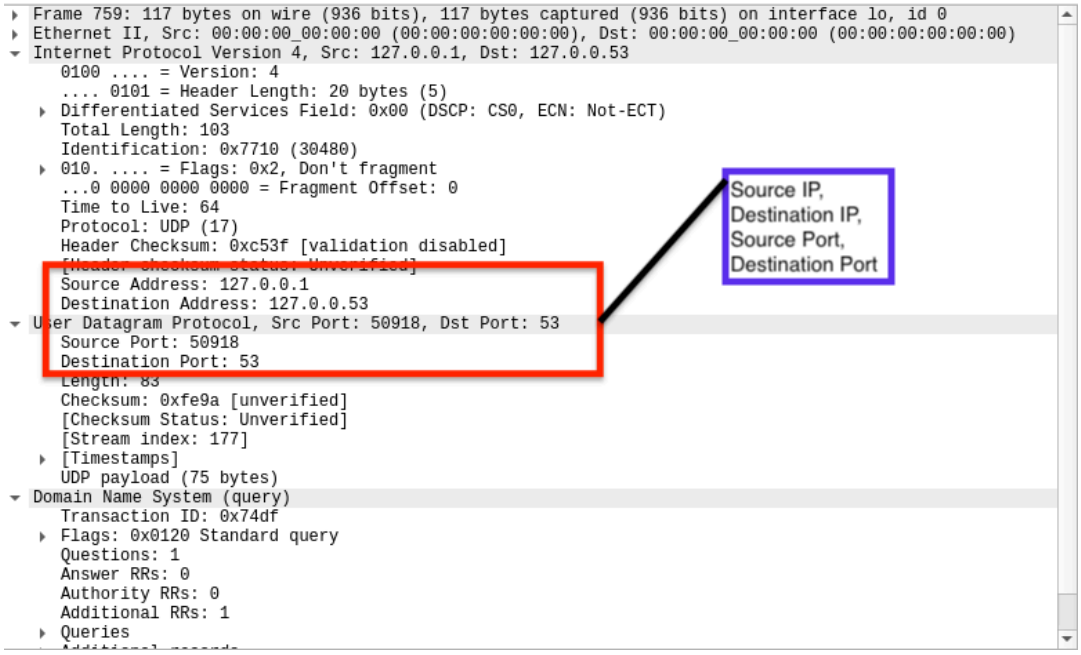
- SYN packet from the client to initiate the connection.
- SYN-ACK from the server acknowledging the request.
- ACK from the client to confirm the connection.

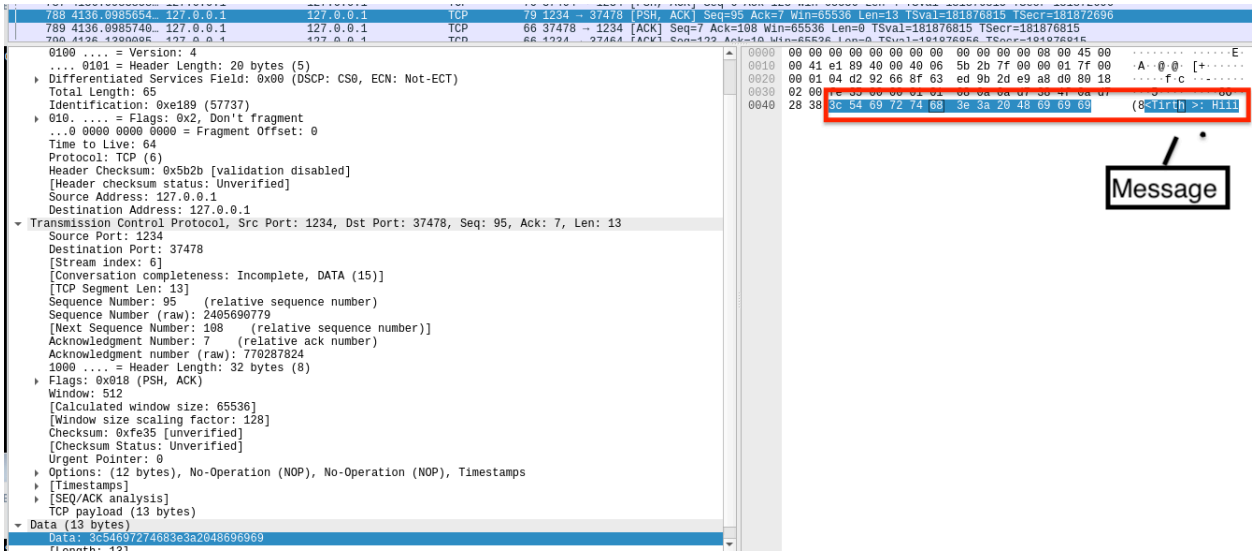
The packets traveled in a sequential manner:

1. Initial SYN request from client to server.
2. Server responds with SYN-ACK.
3. Client sends ACK to complete the connection.
4. Message is sent from the client to the server.
5. Server sends an ACK to confirm receipt.

This image below shows detailed information for a specific packet:

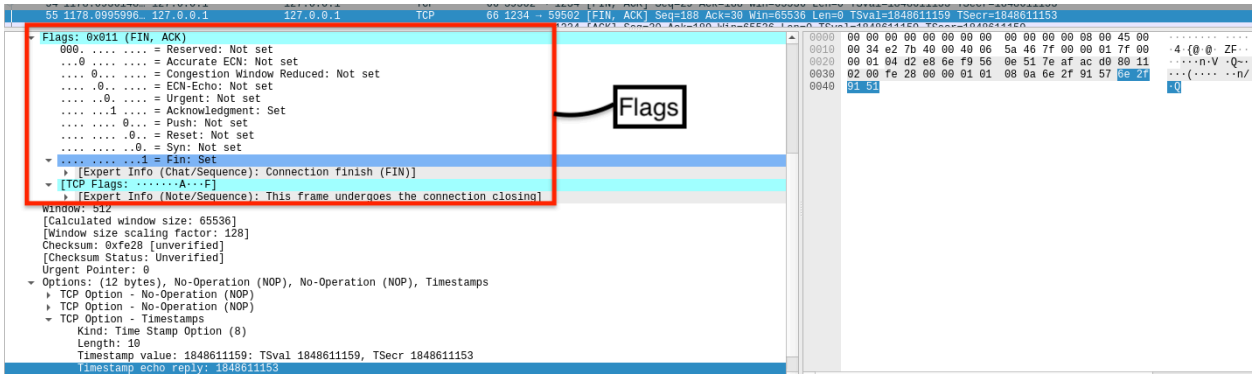
- Source IP: 127.0.0.1 (localhost).
- Destination IP: 127.0.0.53 (localhost).
- Source Port: 50918
- Destination Port: 53





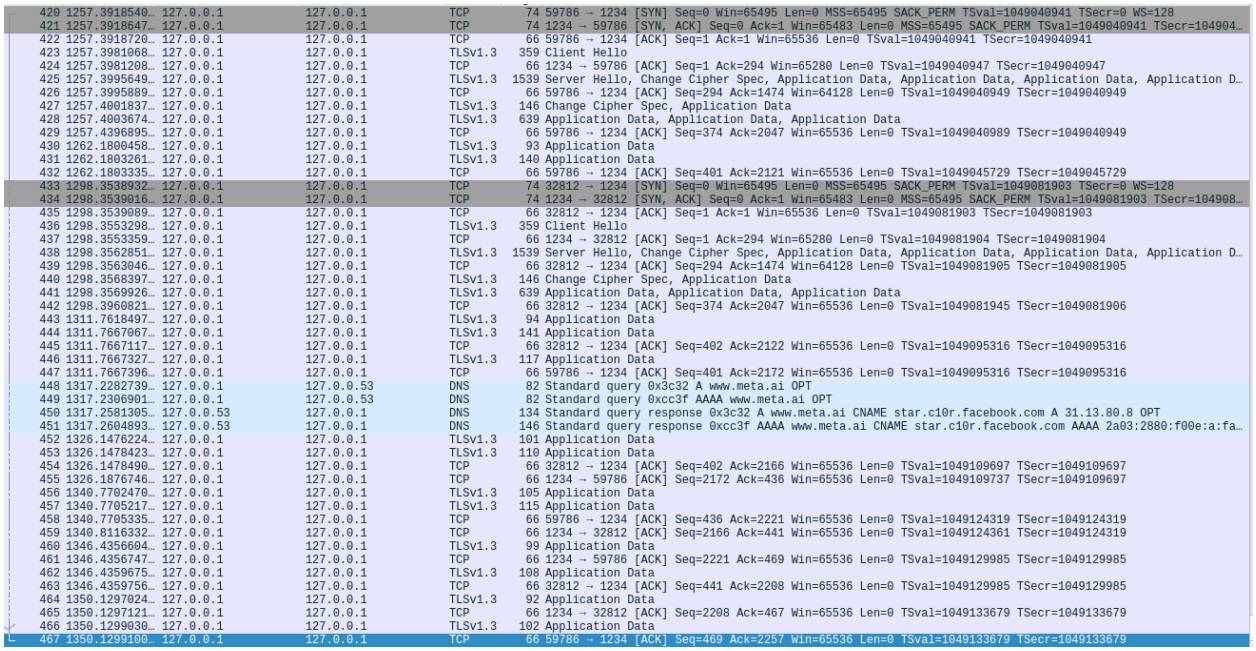
This capture illustrates the message payload being sent from one client to another.

- The message content **Tirth > Hiii** is visible in the TCP payload section.
- This transmission uses the **PSH, ACK** flags, meaning data is being pushed from the sender to the receiver, and an acknowledgment is sent back to confirm receipt.
- The messages are not secure. The content of the message is visible in the TCP payload, as shown in the image above. Since the message is transmitted in plain text and not encrypted, anyone with access to the network can intercept and read the messages.



The image above shows a TCP segment with the FIN and ACK flags set, indicating the sender is initiating a graceful connection termination. The FIN flag signals that no more data will be sent, while the ACK flag confirms receipt of previous data, marking the beginning of the TCP connection closure process.

Phase 2:



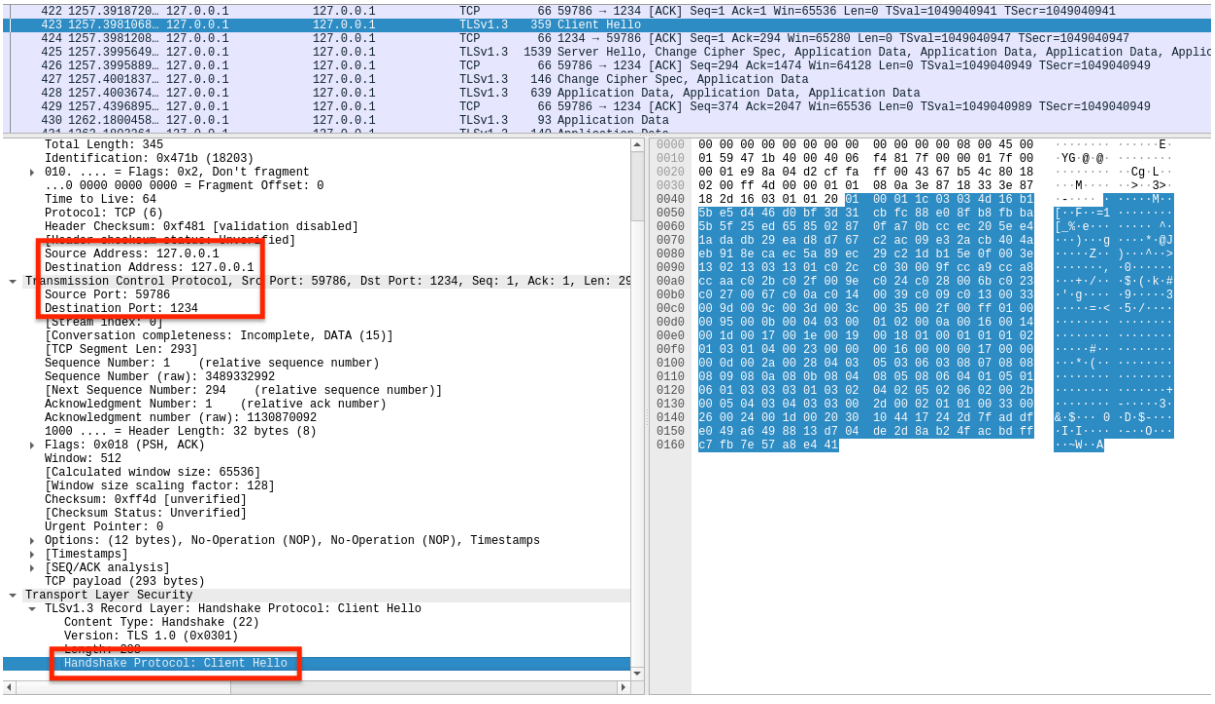
This image shows packet transfers captured between a client and a server, specifically related to the TCP and TLS handshake process.

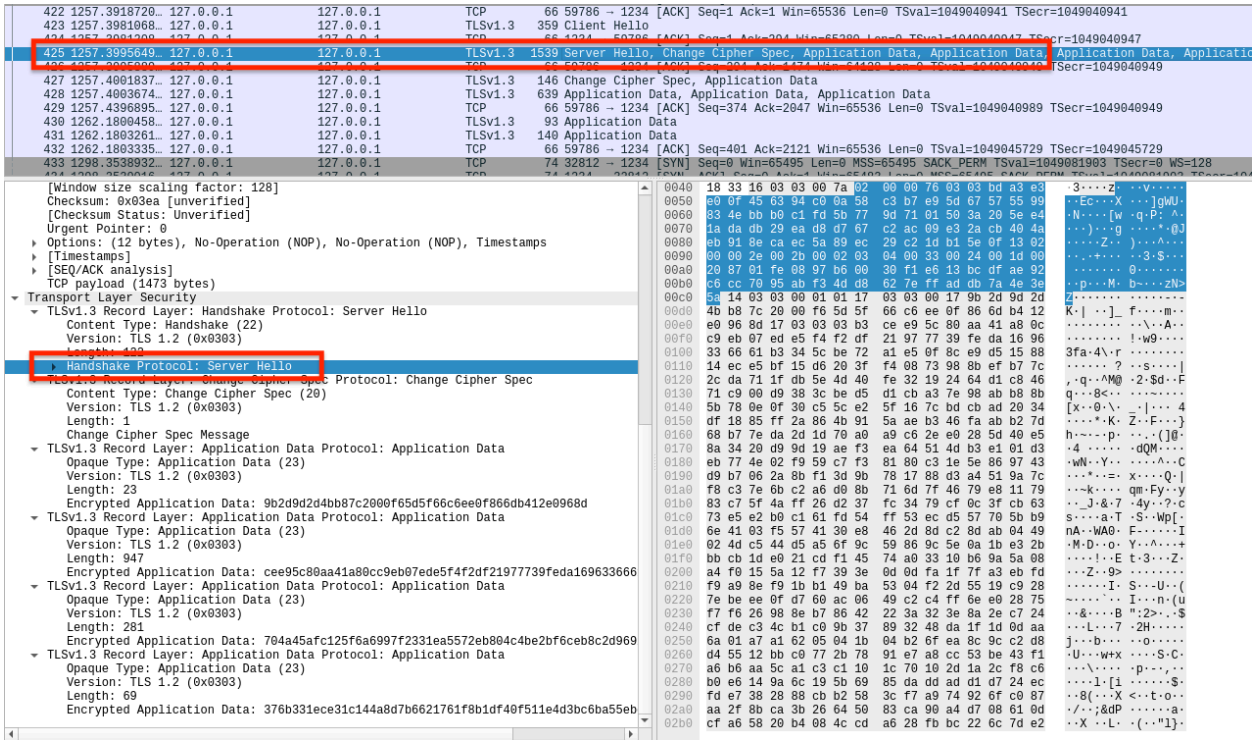
Order of Packet Transfer:

1. The client sends a SYN request to establish a connection with the server. **TCP SYN** (Packet 420)
2. The server responds with a **SYN-ACK** to acknowledge the client's **SYN** request. **TCP SYN-ACK** (Packet 421)
3. The client sends back an **ACK** packet to confirm receipt of the server's SYN-ACK. **TCP ACK** (Packet 422)
4. The client initiates the TLS handshake with a **Client Hello message**, which indicates supported versions of TLS (TLS 1.3). **TLS Client Hello** (Packet 423)
5. The server responds with a **Server Hello message. TLS Server Hello** (Packet 425)
6. The server sends a Change Cipher Spec message to indicate **encrypted communication.**

This image below shows detailed information for a specific packet:

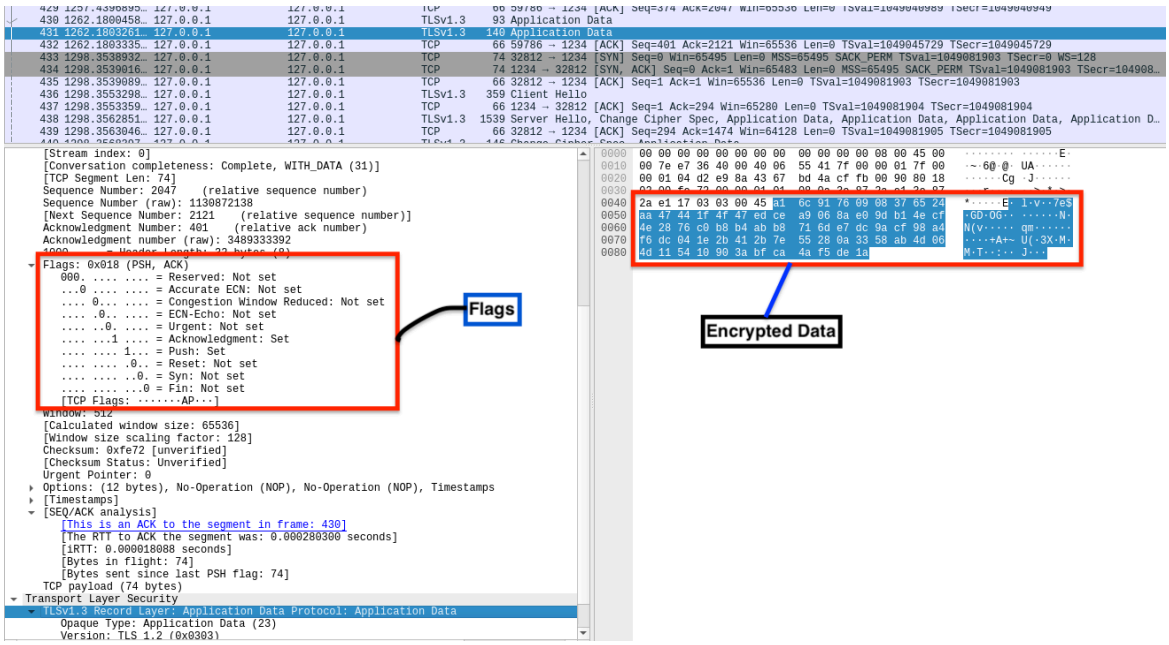
- Source IP: 127.0.0.1 (localhost).
- Destination IP: 127.0.0.1 (localhost).
- Source Port: 59786
- Destination Port: 1234





Which Protocol is Being Used Now?

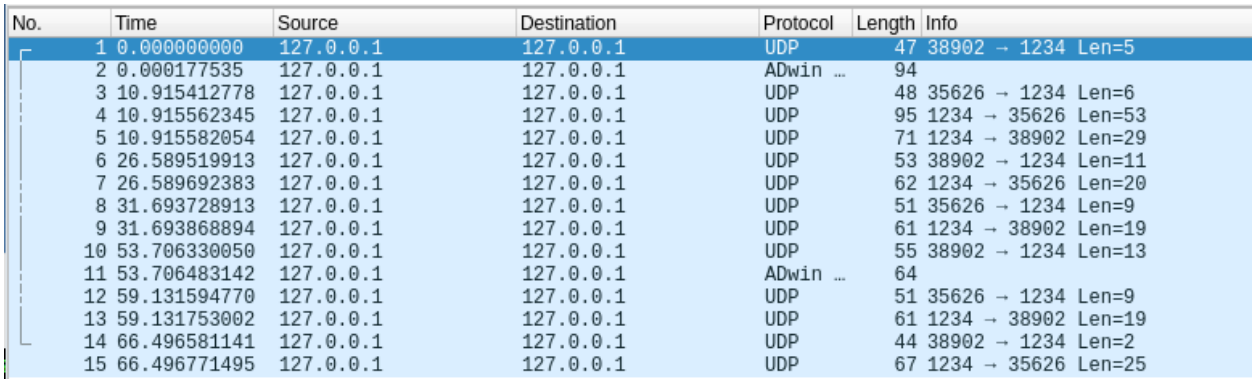
The protocol being used is TLS 1.2 (Transport Layer Security). This can be seen in the TLSv1.2 Record Layer section of the packet capture.



The images above show that we can't see the actual messages being sent. The data is encrypted, as you can tell from the label "Encrypted Application Data" in the capture. The hex values shown in the packet don't show any readable content, which means the messages are securely encrypted and protected. The **PSH** and **ACK** flags are set, indicating that the data is being pushed to the application and acknowledging the receipt of previous data.

From an end-user perspective, there would be no noticeable difference in terms of interaction with the application. The user interface and general behavior would remain the same.

Phase 3:

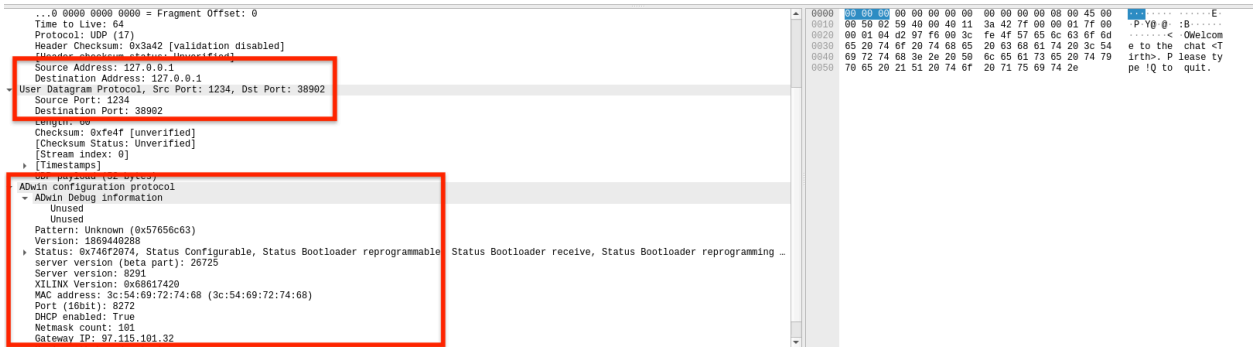


No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	UDP	47	38902 → 1234 Len=5
2	0.000177535	127.0.0.1	127.0.0.1	ADwin …	94	
3	10.915412778	127.0.0.1	127.0.0.1	UDP	48	35626 → 1234 Len=6
4	10.915562345	127.0.0.1	127.0.0.1	UDP	95	1234 → 35626 Len=53
5	10.915582054	127.0.0.1	127.0.0.1	UDP	71	1234 → 38902 Len=29
6	26.589519913	127.0.0.1	127.0.0.1	UDP	53	38902 → 1234 Len=11
7	26.589692383	127.0.0.1	127.0.0.1	UDP	62	1234 → 35626 Len=20
8	31.693728913	127.0.0.1	127.0.0.1	UDP	51	35626 → 1234 Len=9
9	31.693868894	127.0.0.1	127.0.0.1	UDP	61	1234 → 38902 Len=19
10	53.706330050	127.0.0.1	127.0.0.1	UDP	55	38902 → 1234 Len=13
11	53.706483142	127.0.0.1	127.0.0.1	ADwin …	64	
12	59.131594770	127.0.0.1	127.0.0.1	UDP	51	35626 → 1234 Len=9
13	59.131753002	127.0.0.1	127.0.0.1	UDP	61	1234 → 38902 Len=19
14	66.496581141	127.0.0.1	127.0.0.1	UDP	44	38902 → 1234 Len=2
15	66.496771495	127.0.0.1	127.0.0.1	UDP	67	1234 → 35626 Len=25

The above image shows the sequence of UDP packets exchanged between the client and the server.

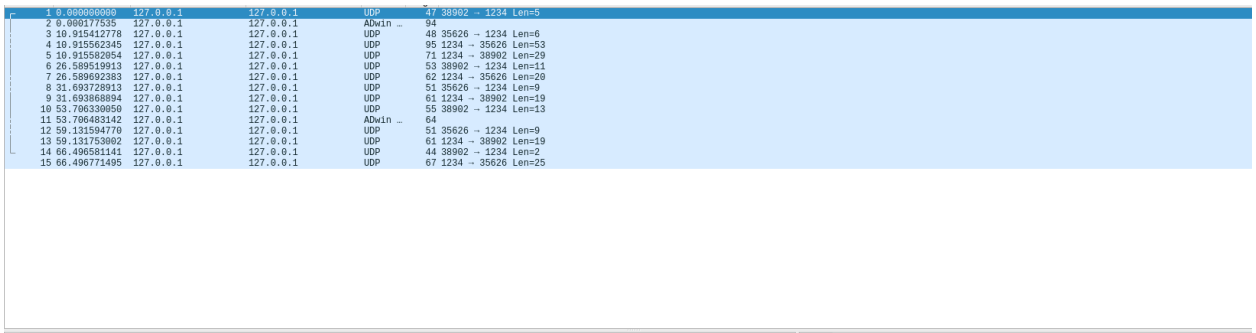
Order of Packet Transfer for UDP:

1. The client sends the first **UDP datagram** to the server. Since UDP is connectionless, this is sent without any prior handshake.(Packet 1)
2. After the initial message, subsequent UDP packets contain the chat messages sent by the client to the server and vice versa. (Packet3, Packet 7 and packets 11-15).
3. The client sends a message indicating that the user has left the chat, marking the end of communication.(Packet 15)



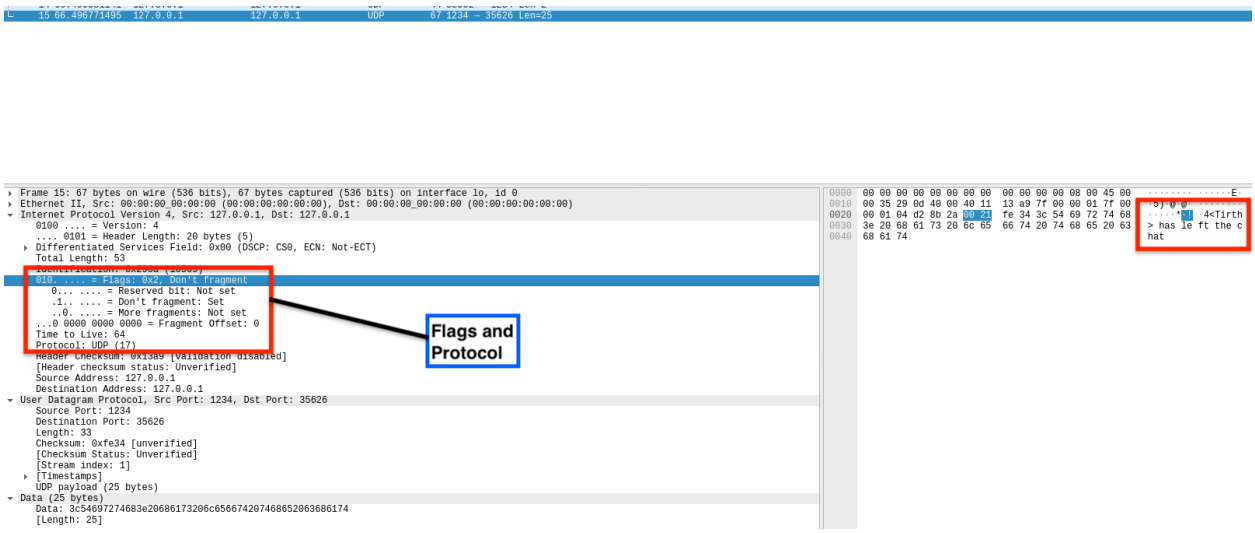
The above image shows detailed information for a specific packet:

- Source IP: 127.0.0.1 (localhost).
- Destination IP: 127.0.0.1 (localhost).
- Source Port: 1234
- Destination Port: 38902



The protocol used is **UDP (User Datagram Protocol)**, which is connectionless and does not guarantee message delivery.

These messages are not secure. The reason is that the messages are transmitted in plain text without any encryption. Anyone capturing these packets can easily view the content, as shown above the name "**Tirth**" is clearly readable.



This image shows the "**Don't Fragment**" flag in the UDP packet. It means that the message should be sent in one piece without being split into smaller parts. UDP is simpler and doesn't have extra flags for things like starting or ending a connection.