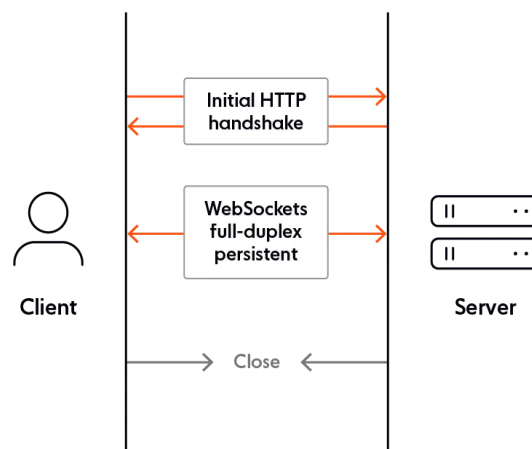


# WebSocket Protocol

Abhishek Raj, En-21114004

## Overview:

HTTP and WebSocket are both communication protocols used in client-server communication. HTTP is unidirectional, where the client sends the request, and the server sends the response. At the same time, WebSocket is bidirectional, a full-duplex protocol that is used in the same scenario of client-server communication. WebSocket communication involves a handshake, messaging (sending and receiving messages), and closing the connection.



## Why WebSocket are used?

The advantage of WebSocket is that it enables real-time communication between the client and server without frequent HTTP requests/responses. This brings benefits such as reduced latency and improved performance and responsiveness of web apps.

Due to its persistent and bidirectional nature, the WebSocket protocol is more flexible than HTTP when building real-time apps that require frequent data

exchanges. WebSocket is also more efficient, transmitting data without repetitive HTTP headers and handshakes. This can reduce bandwidth usage and server load.

## Use cases of WebSocket:

- ✚ Chatting
- ✚ Live feeds
- ✚ Multiplayer gaming
- ✚ Real-time Location Tracking
- ✚ Data synchronization (in Bitcoin)

## Analyze the WebSocket traffic using Wireshark.

We make WebSocket server and Client file, then pass the message I.e., Abhishek (21114004).

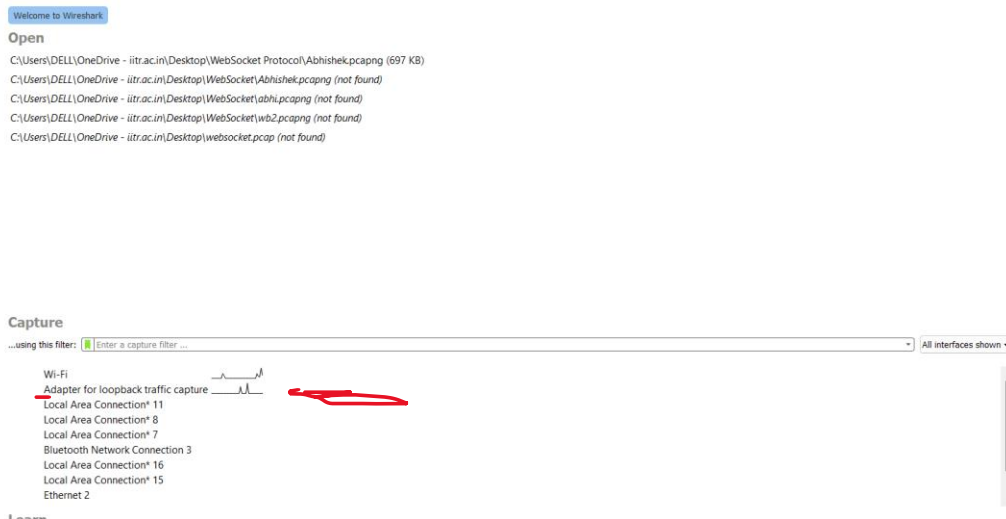
The screenshot shows a Visual Studio Code editor with a file explorer on the left containing 'Abhishek.pcapng', 'client.py', and 'server.py'. The 'server.py' file is open in the editor, showing the following code:

```
1 import asyncio
2 import websockets
3
4 async def echo(websocket, path):
5     async for message in websocket:
6         await websocket.send(message)
7
8 async def main():
9     server = await websockets.serve(echo, "localhost", 8080)
10
11     await server.wait_closed()
12
13 if __name__ == "__main__":
14     asyncio.run(main())
15
```

Below the editor, the 'TERMINAL' tab is active, showing two command prompts. The left prompt is running 'python server.py' and the right prompt is running 'python client.py'. The client output shows the message being sent and received:

```
PS C:\Users\DELL\OneDrive - iitr.ac.in\Desktop\WebSocket Protocol> python server.py
PS C:\Users\DELL\OneDrive - iitr.ac.in\Desktop\WebSocket Protocol> python client.py
Enter a message to send: Abhishek(21114004)
Received response: Abhishek(21114004)
PS C:\Users\DELL\OneDrive - iitr.ac.in\Desktop\WebSocket Protocol>
```

To capture the traffic for Wireshark and then analyze it. So, let's begin with capturing the traffic. When we open Wireshark, it displays all the available network interfaces with live network traffic.



Here we choose Adapter for loopback traffic capture and then capture the packets.

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl>/>

No.	Time	Source	Destination	Protocol	Length	Info
24	14.687764	:::1	:::1	TCP	64	8080 → 61194 [RST, ACK] Seq=1 Ack=9 Win=0 Len=0
25	14.690487	127.0.0.1	127.0.0.1	TCP	45	61193 → 61192 [PSH, ACK] Seq=3 Ack=1 Win=1279 Len=1
26	14.690542	127.0.0.1	127.0.0.1	TCP	44	61192 → 61193 [ACK] Seq=1 Ack=4 Win=8442 Len=0
27	14.690956	127.0.0.1	127.0.0.1	TCP	44	61192 → 61193 [FIN, ACK] Seq=1 Ack=4 Win=8442 Len=0
28	14.691004	127.0.0.1	127.0.0.1	TCP	44	61193 → 61192 [ACK] Seq=4 Ack=2 Win=1279 Len=0
29	14.691051	127.0.0.1	127.0.0.1	TCP	44	61193 → 61192 [FIN, ACK] Seq=8 Ack=2 Win=1279 Len=0
30	14.691086	127.0.0.1	127.0.0.1	TCP	44	61192 → 61193 [ACK] Seq=2 Ack=5 Win=8442 Len=0
31	16.502414	10.81.40.111	10.81.40.111	ICMP	84	Destination unreachable (Host unreachable)
32	19.400251	127.0.0.1	127.0.0.1	TCP	56	61275 → 61274 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
33	19.400325	127.0.0.1	127.0.0.1	TCP	56	61274 → 61275 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
34	19.400392	127.0.0.1	127.0.0.1	TCP	44	61275 → 61274 [ACK] Seq=1 Ack=1 Win=2161152 Len=0
35	19.445457	127.0.0.1	127.0.0.1	TCP	45	61275 → 61274 [PSH, ACK] Seq=1 Ack=1 Win=2161152 Len=1
36	19.445492	127.0.0.1	127.0.0.1	TCP	44	61274 → 61275 [ACK] Seq=1 Ack=2 Win=2161152 Len=0
37	19.445837	:::1	:::1	TCP	76	61276 → 8080 [SYN] Seq=0 Win=65535 Len=0 MSS=65475 WS=256 SACK_PERM
38	19.445888	:::1	:::1	TCP	76	8080 → 61276 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65475 WS=256 SACK_PERM
39	19.445996	:::1	:::1	TCP	64	61276 → 8080 [ACK] Seq=1 Ack=1 Win=2160640 Len=0
40	19.446545	:::1	:::1	HTTP	328	GET / HTTP/1.1
41	19.446572	:::1	:::1	TCP	64	8080 → 61276 [ACK] Seq=1 Ack=265 Win=2160384 Len=0
42	19.447130	:::1	:::1	HTTP	367	HTTP/1.1 101 Switching Protocols
43	19.447158	:::1	:::1	TCP	64	61276 → 8080 [ACK] Seq=265 Ack=304 Win=2160128 Len=0
44	31.282015	:::1	:::1	WebSoc...	90	WebSocket Text [FIN] [MASKED]
45	31.282089	:::1	:::1	TCP	64	8080 → 61276 [ACK] Seq=304 Ack=291 Win=2160384 Len=0
46	31.282959	:::1	:::1	WebSoc...	86	WebSocket Text [FIN]
47	31.282996	:::1	:::1	TCP	64	61276 → 8080 [ACK] Seq=291 Ack=326 Win=2160128 Len=0
48	31.283818	:::1	:::1	WebSoc...	72	WebSocket Connection Close [FIN] [MASKED]
49	31.283870	:::1	:::1	TCP	64	8080 → 61276 [ACK] Seq=326 Ack=299 Win=2160128 Len=0
50	31.284198	:::1	:::1	WebSoc...	68	WebSocket Connection Close [FIN]
51	31.284240	:::1	:::1	TCP	64	61276 → 8080 [ACK] Seq=299 Ack=330 Win=2160128 Len=0
52	31.284382	:::1	:::1	TCP	64	8080 → 61276 [FIN, ACK] Seq=330 Ack=299 Win=2160128 Len=0
53	31.284420	:::1	:::1	TCP	64	61276 → 8080 [ACK] Seq=299 Ack=331 Win=2160128 Len=0
54	31.284617	:::1	:::1	TCP	64	61276 → 8080 [FIN, ACK] Seq=299 Ack=331 Win=2160128 Len=0
55	31.284672	:::1	:::1	TCP	64	8080 → 61276 [ACK] Seq=331 Ack=300 Win=2160128 Len=0
56	31.286141	127.0.0.1	127.0.0.1	TCP	45	61275 → 61274 [PSH, ACK] Seq=2 Ack=1 Win=2161152 Len=1
57	31.286205	127.0.0.1	127.0.0.1	TCP	44	61274 → 61275 [ACK] Seq=1 Ack=3 Win=2161152 Len=0
58	31.286496	127.0.0.1	127.0.0.1	TCP	44	61274 → 61275 [FIN, ACK] Seq=1 Ack=3 Win=2161152 Len=0
59	31.286529	127.0.0.1	127.0.0.1	TCP	44	61275 → 61274 [ACK] Seq=3 Ack=2 Win=2161152 Len=0
60	31.286557	127.0.0.1	127.0.0.1	TCP	44	61275 → 61274 [FIN, ACK] Seq=3 Ack=2 Win=2161152 Len=0
61	31.286580	127.0.0.1	127.0.0.1	TCP	44	61274 → 61275 [ACK] Seq=2 Ack=4 Win=2161152 Len=0

Let's try to analyze the network packets captured so far. First, since the initial handshake is on HTTP protocol, let's filter the packets for **WebSocket** protocol.

The top panel shows a list of captured packets filtered for 'websocket'. The bottom panel shows the details of packet 44, which is a WebSocket text message. A red circle highlights the 'Line-based text data (1 lines)' section, which contains the text 'Abhishek(21114004)'.

No.	Time	Source	Destination	Protocol	Length	Info
44	31.282015	:::1	:::1	WebSoc...	90	WebSocket Text [FIN] [MASKED]
46	31.282959	:::1	:::1	WebSoc...	86	WebSocket Text [FIN]
48	31.283818	:::1	:::1	WebSoc...	72	WebSocket Connection Close [FIN] [MASKED]
50	31.284198	:::1	:::1	WebSoc...	68	WebSocket Connection Close [FIN]
1374	176.695595	:::1	:::1	WebSoc...	90	WebSocket Text [FIN] [MASKED]
1376	176.696161	:::1	:::1	WebSoc...	86	WebSocket Text [FIN]
1378	176.697093	:::1	:::1	WebSoc...	72	WebSocket Connection Close [FIN] [MASKED]
1380	176.697336	:::1	:::1	WebSoc...	68	WebSocket Connection Close [FIN]

Frame 44: 90 bytes on wire (720 bits), 90 bytes captured (720 bits) on interface \Device\NPF\_{...}, id 0  
Null/Loopback  
Internet Protocol Version 6, Src: :::1, Dst: :::1  
Transmission Control Protocol, Src Port: 61276, Dst Port: 8080, Seq: 265, Ack: 304, Len: 26  
WebSocket  
... .. = Fin: True  
... .. = Reserved: 0x4  
... .. = Per-Message Compressed: True  
... .. 0001 = Opcode: Text (1)  
... .. = Mask: True  
... .. 001 0100 = Payload length: 20  
Masking-Key: 817be8ab  
Masked payload  
Payload  
Line-based text data (1 lines)  
Abhishek(21114004)

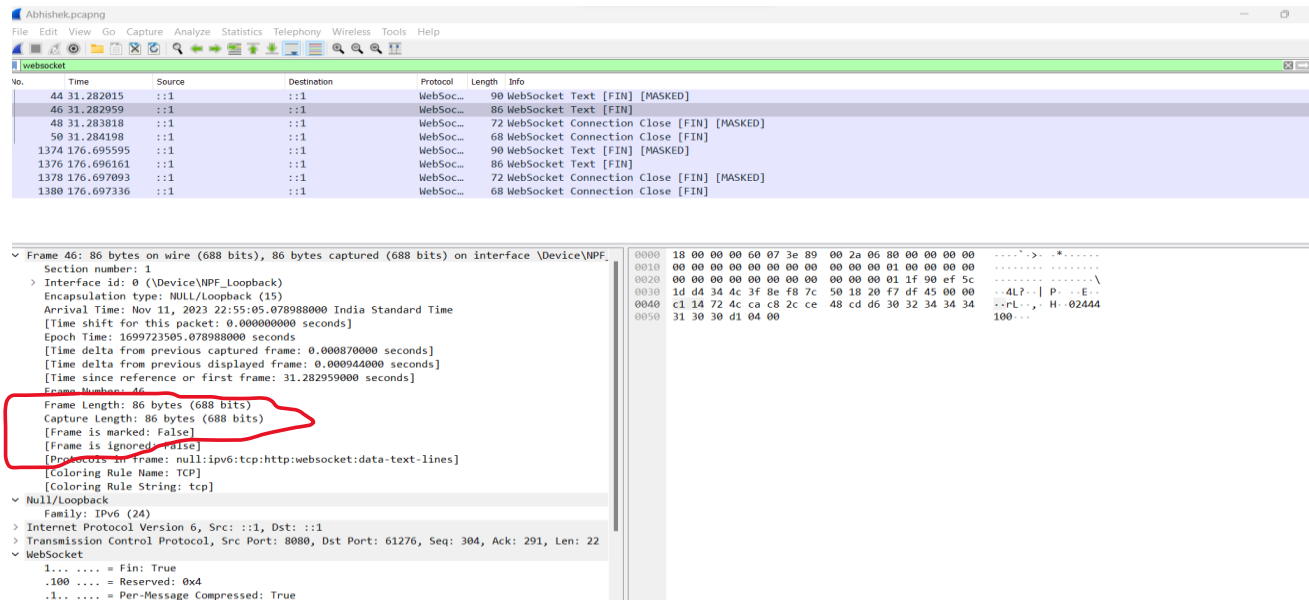
Next, to get a detailed view of the handshake, right-click on the packet → **Follow** → **TCP Stream**

The bottom panel shows the details of the WebSocket handshake. A blue callout points to the 'Request line' (GET / HTTP/1.1) and another blue callout points to the 'Response line' (HTTP/1.1 101 Switching Protocols).

Request line: GET / HTTP/1.1  
Host: localhost:8080  
Upgrade: websocket  
Connection: Upgrade  
Sec-WebSocket-Key: kFyK93ogUNefw4BQFmLJmw==  
Sec-WebSocket-Version: 13  
Sec-WebSocket-Extensions: permessage-deflate; client\_max\_window\_bits  
User-Agent: Python/3.11 websockets/12.0

Response line: HTTP/1.1 101 Switching Protocols  
Upgrade: websocket  
Connection: Upgrade  
Sec-WebSocket-Accept: GRhmGGB8eMmuBYzzItwUBGLGZs=  
Sec-WebSocket-Extensions: permessage-deflate; server\_max\_window\_bits=12; client\_max\_window\_bits=12  
Date: Sat, 11 Nov 2023 17:24:53 GMT  
Server: Python/3.11 websockets/12.0

Analyzing the performance of WebSocket involves monitoring key metrics to understand how well your WebSocket-based application is performing. Here are some steps and considerations for analyzing WebSocket performance:



## 1. Packet Capture and Analysis:

- Use tools like Wireshark to capture WebSocket traffic.
- Analyze the captured packets to ensure that the WebSocket connections are being established correctly and that messages are being exchanged efficiently.

## 2. Latency Measurement:

- Analyze the packet timestamps to calculate the latency.

## 3. Throughput Analysis:

- Measure the throughput or data transfer rate of your WebSocket connections.
- Evaluate how efficiently your application can send and receive data over WebSocket.
- Calculate throughput manually by dividing the amount of data transferred by the time taken.