

MINISTRY OF EDUCATION REPUBLIC OF MOLDOVA

TECHNICAL UNIVERSITY OF MOLDOVA

REPORT

Laboratory work no. 5

On Network Programming

Performed by:

st. gr. FAF-141

Cristea Victor

Verified by:

Ciorba Dumitru

Chisinau 2017

Wireshark. Quick-star

Task: The purpose of the lab work involves understanding the protocol and Its application. Thus, type tasks vary between a simple Http client at Application-tools for collecting information on the Web: Simple Http client able to send and receive responses to requests Get, Head and Post type;

Objectives: Understanding the HTTP protocol and its role in Web communication, study of Java components useful in application HTTP protocol; The specific objective of the work is to create a Java client applications that would interact with Web servers via HTTP methods studied.

Theory

Wireshark is a network packet analyzer, known previously as Ethereal. It lets you examine the network traffic flowing into and out of your Windows or Unix machine. Network professionals use Wireshark to troubleshoot networking problems, but it is also an excellent way to learn exactly how the network protocols work. For example, it allows us to see the data that your system sends and receives when you type a web address into a web browser (e.g., Internet Explorer or Mozilla's Firefox).

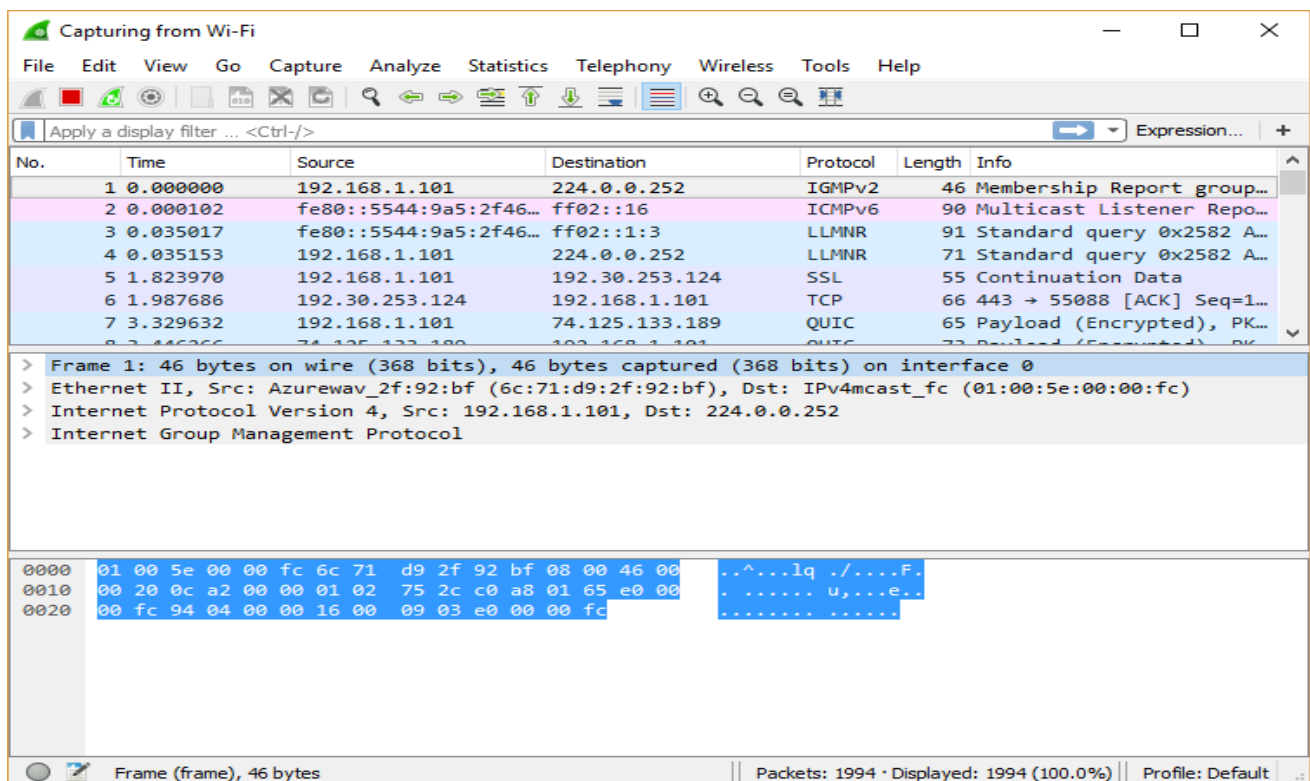


Fig. 1,2 Wireshark GUI

Wireshark uses colors to help you identify the types of traffic at a glance. By default, green is TCP traffic, dark blue is DNS traffic, light blue is UDP traffic, and black identifies TCP packets with problems — for example, they could have been delivered out-of-order.

Another interesting thing you can do is right-click a packet and select Follow TCP Stream

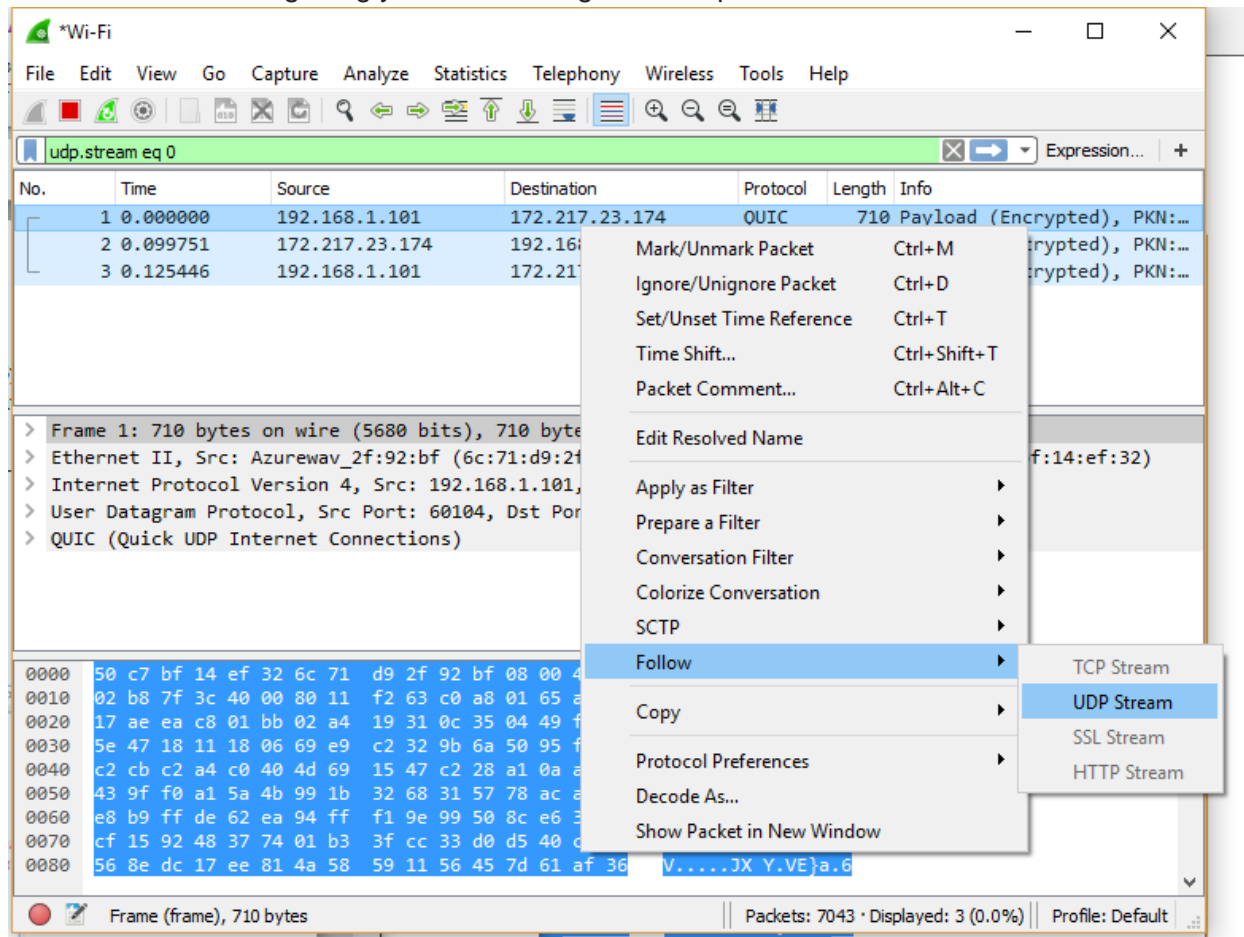


Fig.5

You'll see the full conversation between the client and the server.

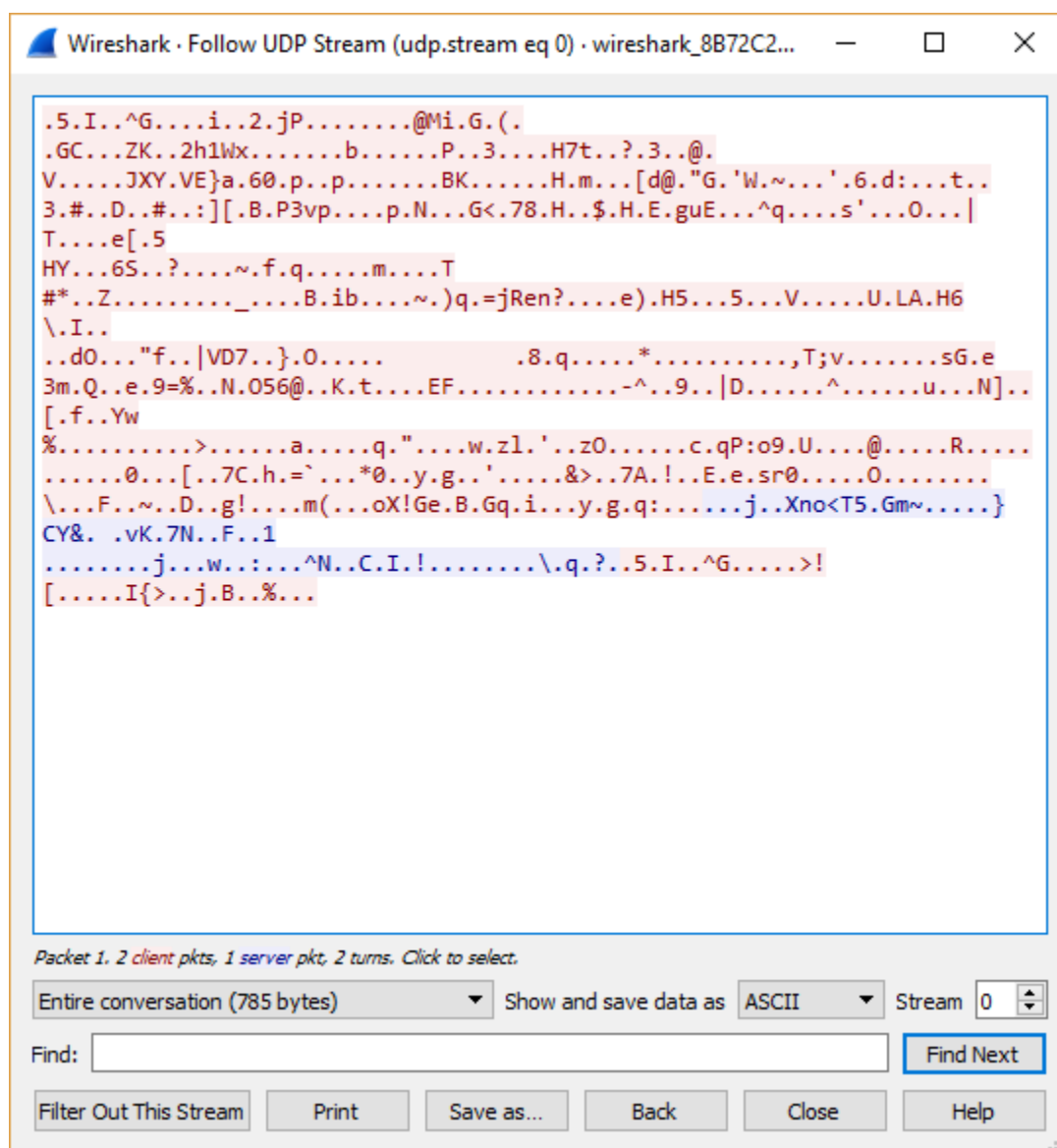


Fig.6

When we close the window and you'll find a filter has been applied automatically — Wireshark is showing you the packets that make up the conversation.

QuickChat

In order to understand how QCP works, we will need to intercept the messages being broadcasted. For that, we use a network sniffer, WireShark.

To only intercept relevant messages, we apply a filter for the port == 8167, which we have learned by checking the settings for QuickChat.

By looking at the messages that are broadcasted when different events happen, we can determine the formula for these messages.

Below is the list of events and messages observed by me.

Check users list: "0 %s-nickname%"

RespondToCheck: "1 %s-RequesterNickname% %s-nickname%"

Message: "2 %s-channelName% %s-nickname% %message%"

Change nickname: "3 %s-old_nickname% %s-new_nickname% 0"

Join channel: "4 %s-nickname% %s-channelName% 0"

Leave channel: "5 %s-nickname% %s-channelName% 0"

Private message: "6 %s-nickname% %s-targetNickname% %message%"

Private message read: "7 %s-senderNickname% %s-targetNickname% 0"

Change topic: "B %s-NewTopic% (%s-nickname)%"

Set Status: "D %s-Nickname% %i-StatusCode% 0"

Status codes: 0 - normal, 1 - DND, 2 - away, 3 - offline.

Join chat: "J0 %s-nickname1% %s-nickname2%"

Leave chat: "J1 %s-nickname1% %s-nickname2%"

Chat message: "J2 %s-nickname1% %s-nickname2% %s-message%"

Request List of users on channel: "L %s-nickname% %s-channel%"

Response to channel users request: "L %s-RequesterNickname% %s-channel% %s-nickname% %i-mode%" //Mode can be online (1) and offline (0 or anything else)

List channels: "N %s-nickname%"

Respond to list channels: "O %s-requesterNickname% %s-channelName1% %s-channelName2% ...

Whenever a channel is joined, a JoinChannel message is sent, along with a request to view already connected members. Members then send a response marked with the requester nickname so that the newly joined member can see everybody on the channel.

Whenever the channels dialog is open, QuickChat sends a ListChannels request, and every client online sends a message with a list of active channels.

When a private message is sent, the receiver sends an acknowledgement message to let the sender know he got the message.

The minimalistic console client I designed intercepts all messages sent to port 8167. If the message belongs to the messages analyzed above, it is explained. Otherwise, the message is displayed in its raw form.

From the console, a few commands are available.

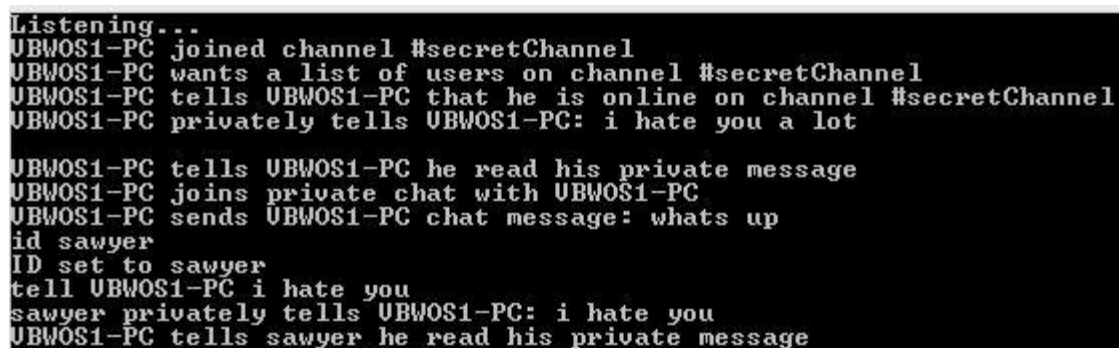
id %new id% changes the id the client is using (doesn't send a message).

channel %new channel% changes the channel that messages will be sent to

msg %message% sends a message to the current channel with the current id

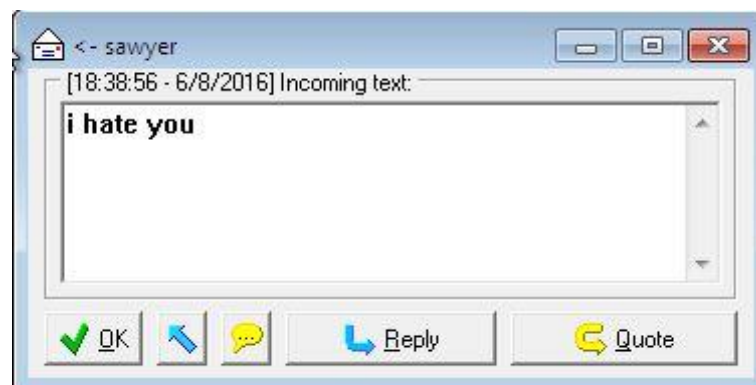
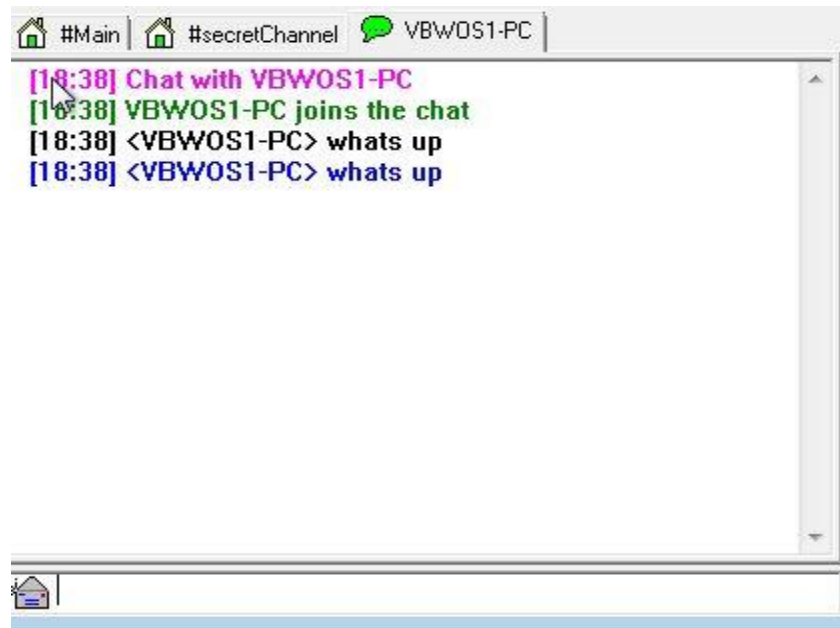
tell %targetId% %message% sends a private message to the target id with the current id

Results:



```
Listening...
UBWOS1-PC joined channel #secretChannel
UBWOS1-PC wants a list of users on channel #secretChannel
UBWOS1-PC tells UBWOS1-PC that he is online on channel #secretChannel
UBWOS1-PC privately tells UBWOS1-PC: i hate you a lot

UBWOS1-PC tells UBWOS1-PC he read his private message
UBWOS1-PC joins private chat with UBWOS1-PC
UBWOS1-PC sends UBWOS1-PC chat message: whats up
id sawyer
ID set to sawyer
tell UBWOS1-PC i hate you
sawyer privately tells UBWOS1-PC: i hate you
UBWOS1-PC tells sawyer he read his private message
```



In order to prevent such problems as spoofing and to preserve privacy, any programs that communicate over a network must encode their messages. A pretty cool way to do this is using the public/private keys algorithm.

If something is encrypted with public key A, only private key A can decrypt it, and vice-versa, if something is encrypted with private key B, only public key B can decrypt it.

For example, in a communication between A and B, A will encode its message using the public key B (visible to everyone) and then encrypt it again using private key A (secret).

When B will get the message, it will decrypt it using A's public key (so it will know for sure A sent the message) and then using its own private key B (is only known to B). This way, the message will only be readable for B, and it will know for sure A sent it.

TCP and UDP

TCP stands for Transmission Control Protocol. It's the most commonly used protocol on the Internet.

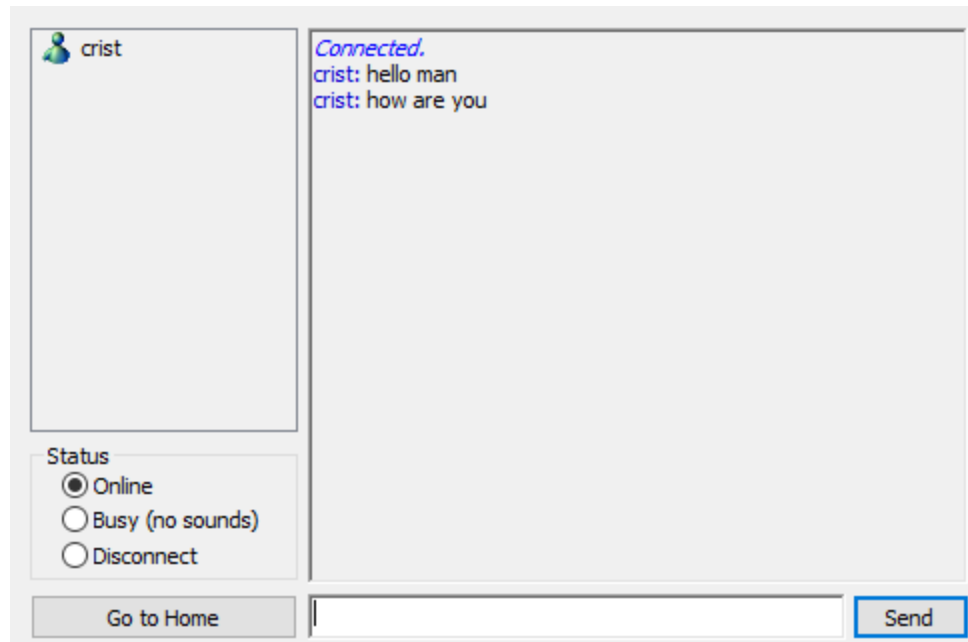
When you load a web page, your computer sends TCP packets to the web server's address, asking it to send the web page to you. The web server responds by sending a stream of TCP packets, which your web browser stitches together to form the web page and display it to you. When you click a link, sign in, post a comment, or do anything else, your web browser sends TCP packets to the server and the server sends TCP packets back. TCP isn't just one way communication — the remote system sends packets back to acknowledge it's received your packets.

UDP stands for User Datagram Protocol — a datagram is the same thing as a packet of information. The UDP protocol works similarly to TCP, but it throws all the error-checking stuff out. All the back-and-forth communication and deliverability guarantees slow things down.

When using UDP, packets are just sent to the recipient. The sender won't wait to make sure the recipient received the packet — it will just continue sending the next packets. If you're the recipient and you miss some UDP packets, too bad — you can't ask for those packets again. There's no guarantee you're getting all the packets and there's no way to ask for a packet again if you miss it, but losing all this overhead means the computers can communicate more quickly.

UDP is used when speed is desirable and error correction isn't necessary. For example, UDP is frequently used for live broadcasts and online games.

Work flow:



We start quickchat and start sniffing the datas from conversation

By the same time, we run Wireshark, and catch the interface for Wireless Network, after that we filter the IP address of our second device.

2599	217.960281	63.141.200.69	192.168.55.67	UDP	106	3478 → 55964 Len=64
2627	221.709253	213.200.111.115	192.168.55.67	UDP	107	3478 → 55963 Len=65
2665	222.862233	81.180.72.3	192.168.55.67	DNS	226	Standard query response 0x8a52 A www.google.com
2667	222.923708	81.180.72.3	192.168.55.67	DNS	375	Standard query response 0x5dbd A ping.char
2746	226.044258	63.141.200.84	192.168.55.67	UDP	106	3478 → 55964 Len=64
2833	234.086928	217.212.238.148	192.168.55.67	UDP	106	3478 → 55964 Len=64
2838	234.219773	81.180.72.3	192.168.55.67	DNS	500	Standard query response 0x26c3 A v10.vorte
2915	237.875217	213.200.111.115	192.168.55.67	UDP	107	3478 → 55963 Len=65
2931	239.411344	192.168.55.148	192.168.55.67	UDP	59	62310 → 8167 Len=17
3020	249.296335	81.180.72.61	192.168.55.67	ICMP	86	Destination unreachable (Port unreachable)
3027	249.816422	192.168.55.148	192.168.55.67	UDP	57	62312 → 8167 Len=15
3033	250.354492	69.192.2.146	192.168.55.67	UDP	106	3478 → 55964 Len=64
3196	266.217493	63.141.200.69	192.168.55.67	UDP	106	3478 → 55964 Len=64
173	19.273501	192.168.55.67	192.168.55.86	UDP	61	64457 → 8167 Len=19
3022	249.298727	192.168.55.67	192.168.55.86	UDP	58	53345 → 8167 Len=16
171	19.271343	192.168.55.67	192.168.56.1	UDP	61	64455 → 8167 Len=19
3019	249.295396	192.168.55.67	192.168.56.1	UDP	58	53343 → 8167 Len=16
172	19.272433	192.168.55.67	192.168.56.86	UDP	61	64456 → 8167 Len=19
3023	249.297352	192.168.55.67	192.168.56.86	UDP	58	53344 → 8167 Len=16

Fig 15 Wireshark filtering the address of the second device

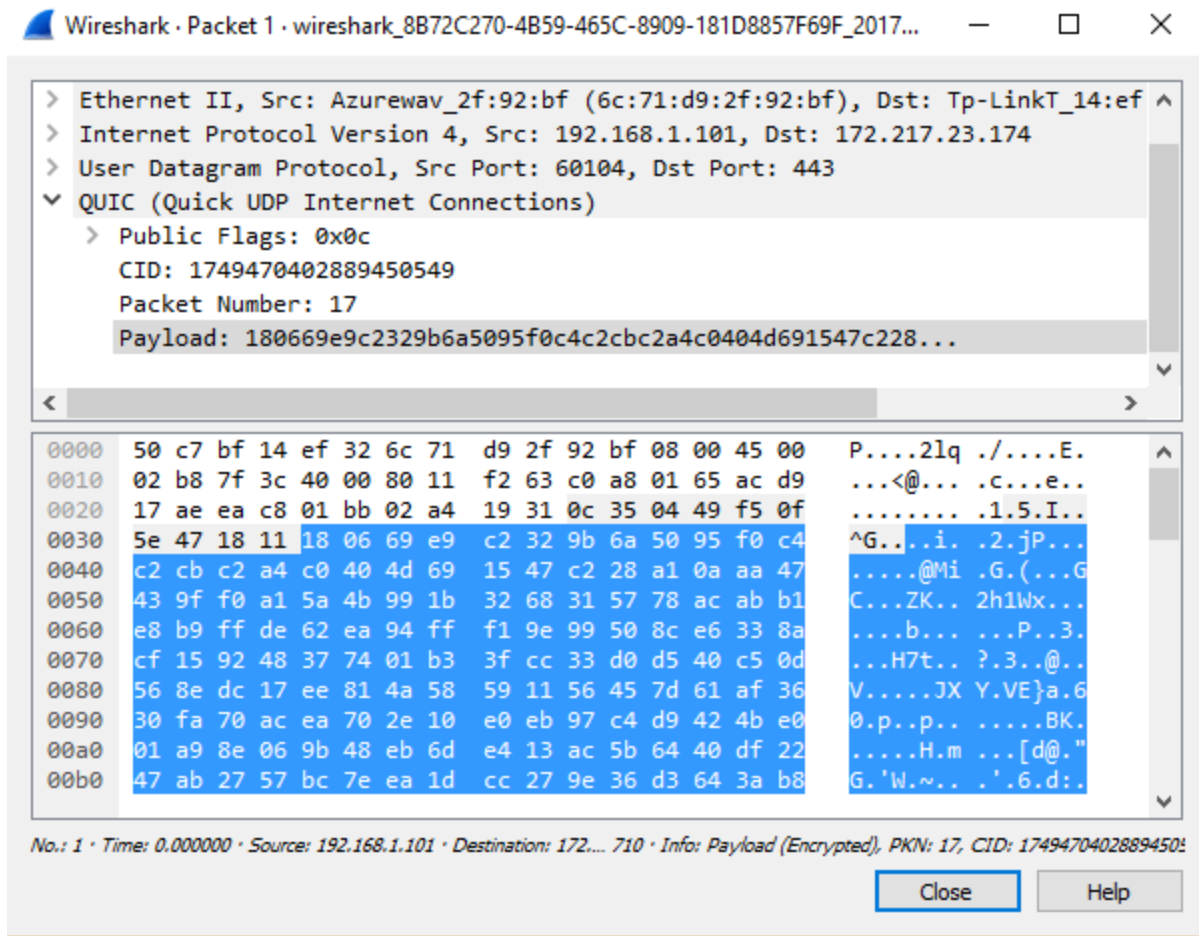


Fig.16 Message sent by quickchat via UDP

Conclusion

Documentation is very important when building clients that communicate over the network. Every type of message sent must be distinguished from others, and work with a formula that also works with all similar messages.

By adding a code at the beginning of the message, we can distinguish a message about setting a topic, and another one that is a private message to someone.

Also, WireShark is a really awesome tool that allows us to see how the computer is communicating over the network, and what binary information is held in the packages.

Repository: <https://github.com/cristeav49/PR/>