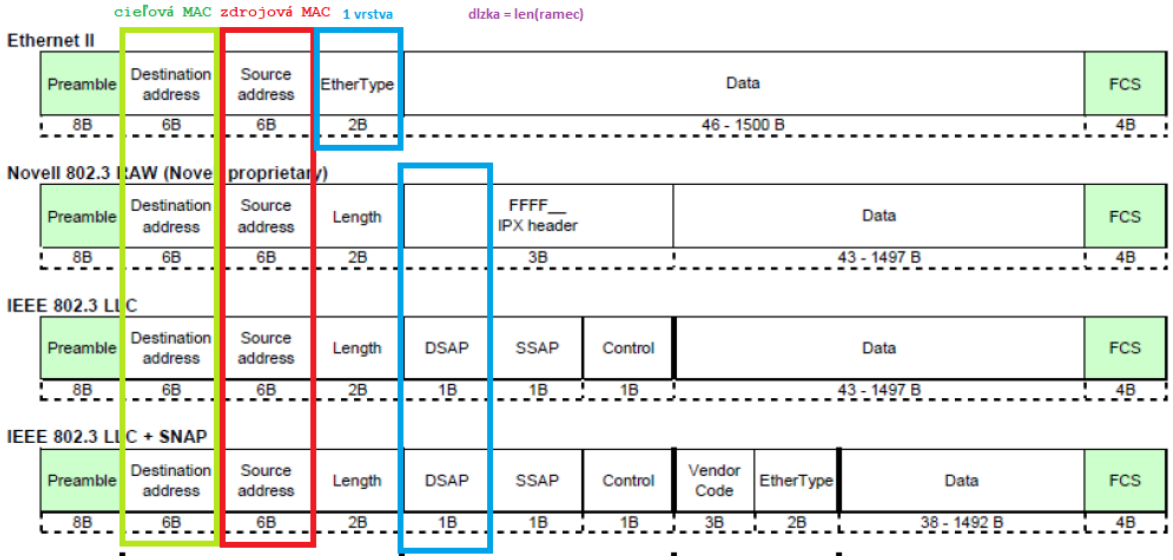
TASK №1

HRYTSYNA ANASTASIIA-SOLOMIIA (105323)

For solving my problem I decided to make a program in **Python** (using PyCharm), as I consider this language easier for implementing my ideas (without pointers and such stuff…). Also, I used **scapy** library to read my .pcap files. To show my outputs I used **console** (where you have to *give the way* to the .pcap files at first and *choose the analysis* you want to implement. In addition, for better identification of all protocols and ports I used extern **Type.txt** (*important: during testing my programm locate Type.txt near to it, so it can be read by the programm*). It contains the hex identification of the protocol and its name.

1. Task: write all frames and basic info about them.

To solve this task I used this scheme and then wrote the whole info I needed:

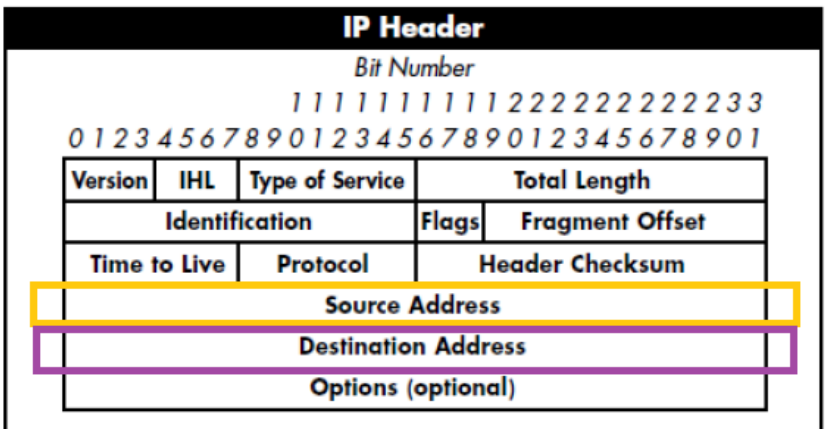


2. Task: write second nested frame.

Into IEEE 802.3 Raw in always nested IPX. Then I look for a nested protocol in IEEE 802.3 LLC+SNAP: it is on 21-22B, IEEE 802.3 LLC – 15B, Ethernet 2 – 13-14B. If hex number (located on this B) I will find in my Types.txt – I will print this nested protocol.

3. Task: write and count ports for IPv4.

For each IPv4 protocol I print this info:



Then destination_ip I add to the dictionary, where key is our ip and value – number, that indicate how many times our ip was is the array of frames. At the end of the output I print all unique ip (keys) and the one that has the biggest number (value) (is there are more ip with the same value – I print them too).

4. Task: analyze communications between frames.

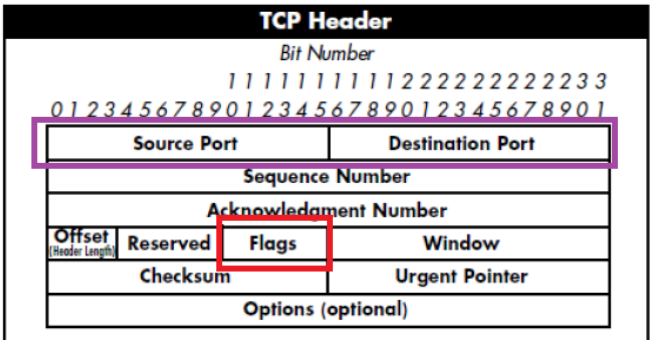
For completing this task I divided all protocols into 4 groups (groups a-c are nested to the IPv4):

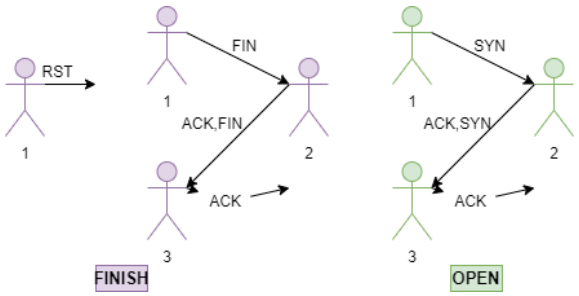
a) TCP

For saving my communications I have created dictionary

communication_TCP = {'SSH': [], 'HTTP': [], 'HTTPS': [], 'TELNET': [], 'FTP_control': [], 'FTP_data': []}

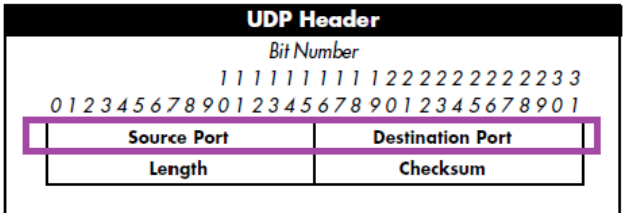
where in each item.value I will put all communications created on this protocol. First I am looking for the nested protocol in TCP (smaller port will tell me which one it is among item.keys). Then (if its ip&ports match and the communication is not finished) I fulfill my communication array with the frames. If there is no matches – I create new communication in the next 'window' of array of the same type(item.key). Communication can be finished if Flags in TCP will create next sequence of handshakes:

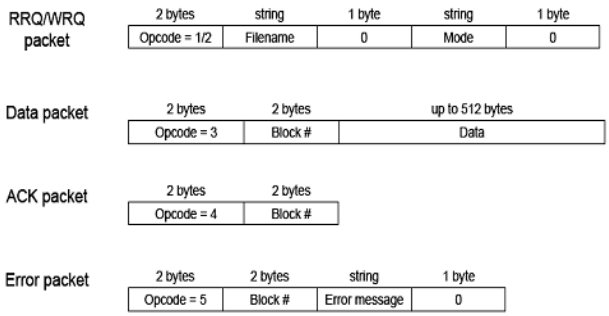




b) UDP

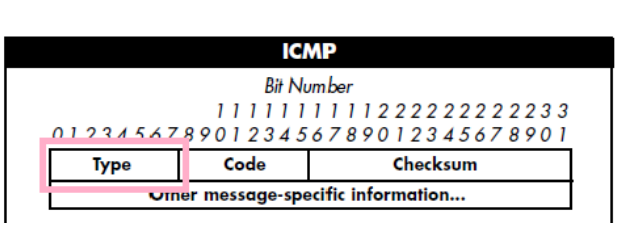
For analyzing this group I compare source_ip and dest_ip again, and the smaller one will show me nested protocol. If it is TFTP – I look at opcode number. If it is 1 or 2 – I create new communication and add it to the array. If it is another number – I try to find proper existing communication in the array communication_UDP['TFTP'] and match them.

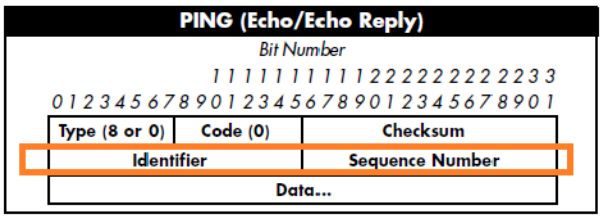




c) ICMP

In this group I take into account Type in ICMP header. If it shows that our frame is Echo/Echo-Reply I try to make communication between them considering just identifier&sequence_number and put it into the array communication_ICMP['PING']. If it is none of them I create communications in array communication_ICMP['OTHERS'] (do not combine them with Echo) considering ip_address.





d) ARP

This one is nested into Ethernet 2, not IPv4. ARP protocol is used for knowing MAC address of another user. So, it sends requests and waits for the reply with information. Operation can help us to know if it is a req/rep. Then, all requests of the same type I add to the one communication. If they get at least one reply – communication is ended.