Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií
Ilkovičova 2, 842 16 Bratislava 4

Predmet / Subject

–Počítačové a komunikačné siete / Computer and communication networks –

- Dokumentácia / Documentation -

Zadanie č.1

Ak. Rok / Academic term: 2022/2023, zimný semester

Cvičiaci / Instructors:
Ing. Lukáš Mastiľák

Študent / Student:
Adam Grík

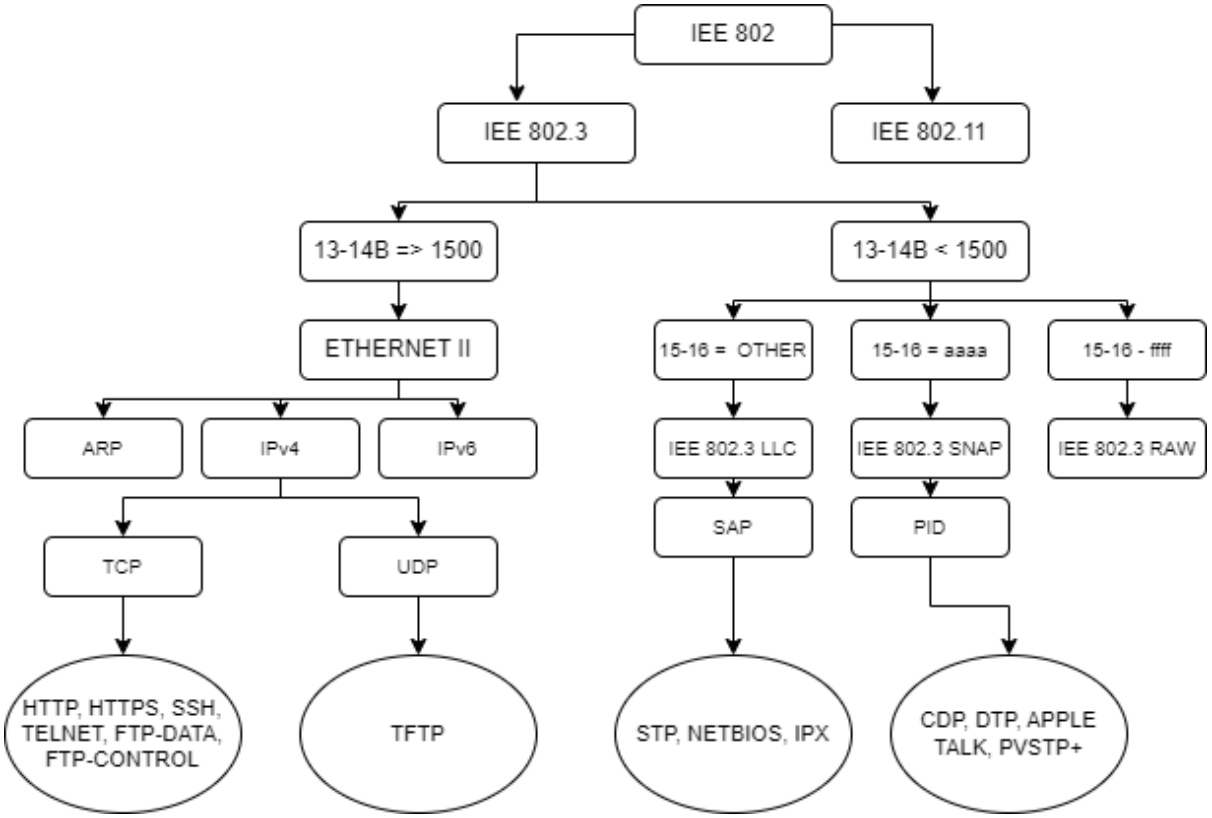
Bratislava, 2022.

Obsah / Content:

1 Úvod

Zadanie č. 1 som implementoval v programovacom jazyku Python. Na projekte som pracoval v implementačnom prostredí IntelliJ IDEA.

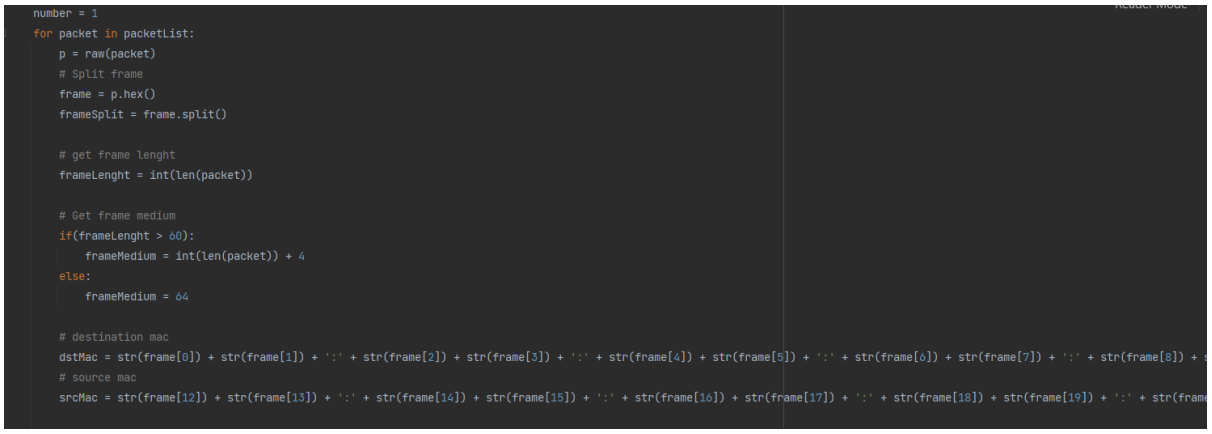
2 Diagram spracovávania a fungovania riešenia



3 Navrhnutý mechanizmus – popis fungovania programu

3.1 Funkcia na výpis všetkých frame-ov do súboru YAML

Prvú som implementoval funkciu, ktorá vypisuje všetky frame-y v zadanom .pcap súbore. Hlavný princíp fungovania tejto metódy spočíva v tom, že prechádzam každý jeden frame kde následne potom čítam údaje z jednotlivých bajtov.

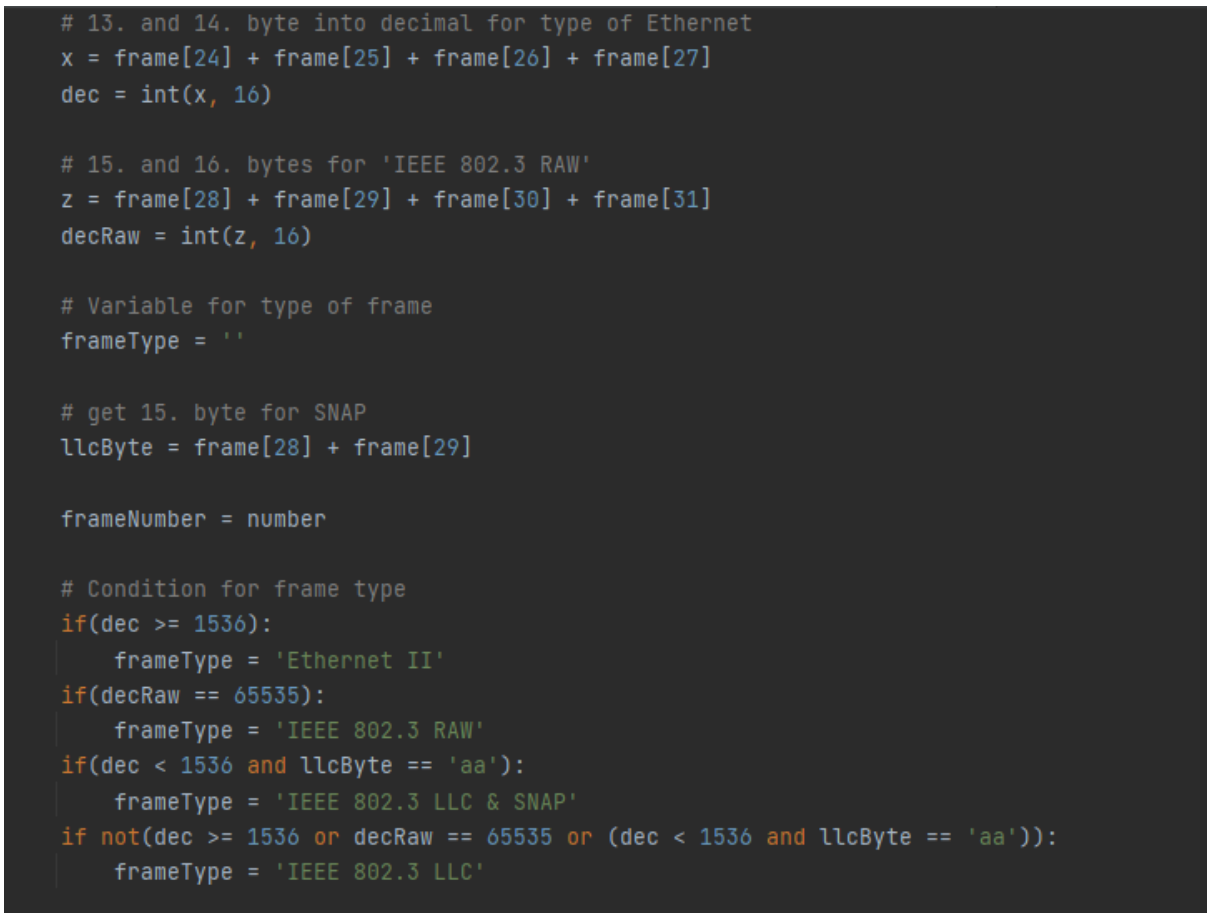


```
number = 1
for packet in packetList:
    p = raw(packet)
    # Split frame
    frame = p.hex()
    frameSplit = frame.split()

    # get frame lenght
    frameLenght = int(len(packet))

    # Get frame medium
    if(frameLenght > 60):
        frameMedium = int(len(packet)) + 4
    else:
        frameMedium = 64

    # destination mac
    dstMac = str(frame[0]) + str(frame[1]) + ':' + str(frame[2]) + str(frame[3]) + ':' + str(frame[4]) + str(frame[5]) + ':' + str(frame[6]) + str(frame[7]) + ':' + str(frame[8]) + s
    # source mac
    srcMac = str(frame[12]) + str(frame[13]) + ':' + str(frame[14]) + str(frame[15]) + ':' + str(frame[16]) + str(frame[17]) + ':' + str(frame[18]) + str(frame[19]) + ':' + str(frame
```



```
    # 13. and 14. byte into decimal for type of Ethernet
    x = frame[24] + frame[25] + frame[26] + frame[27]
    dec = int(x, 16)

    # 15. and 16. bytes for 'IEEE 802.3 RAW'
    z = frame[28] + frame[29] + frame[30] + frame[31]
    decRaw = int(z, 16)

    # Variable for type of frame
    frameType = ''

    # get 15. byte for SNAP
    llcByte = frame[28] + frame[29]

    frameNumber = number

    # Condition for frame type
    if(dec >= 1536):
        frameType = 'Ethernet II'
    if(decRaw == 65535):
        frameType = 'IEEE 802.3 RAW'
    if(dec < 1536 and llcByte == 'aa'):
        frameType = 'IEEE 802.3 LLC & SNAP'
    if not(dec >= 1536 or decRaw == 65535 or (dec < 1536 and llcByte == 'aa')):
        frameType = 'IEEE 802.3 LLC'
```

Ako prvé si bajty daného frame-u rozdelím do poľa. Následne čítam údaje z tohto poľa, podľa toho na akom bajte (v mojom prípade na akej pozícií v poli) sa nachádza daný údaj. Postupne zisťujem dĺžku frame-u, dĺžku frame-u po médiu, destination mac adresu, source mac adresu. Ako ďalšie si zisťujem hodnoty pre to aby som vedel určiť o aký typ Ethernetu sa jedná, kde to potom zisťujem pomocou podmienok.

```
# Get PID for IEEE 802.3 LLC & SNAP
dec2 = 0
pid = ''
if(frameType == 'IEEE 802.3 LLC & SNAP'):
    if not(dstMac == '01:00:0c:00:00:00'):
        y = frame[40] + frame[41] + frame[42] + frame[43]
        dec2 = int(y, 16)
        if(dec2 == 267):
            pid = 'PVSTP+'
        elif(dec2 == 32923):
            pid = 'AppleTalk'
        elif(dec2 == 8196):
            pid = 'DTP'
        elif(dec2 == 8192):
            pid = 'CDP'
```

```
# Get SAP for IEE 802.3 LLC
dec3 = 0
sap = ''
if(frameType == 'IEEE 802.3 LLC'):
    w = frame[30] + frame[31]
    dec3 = int(w, 16)
    if(dec3 == 66): # STP -> 42
        sap = 'STP'
    elif(dec3 == 224): # IPX -> E0
        sap = 'IPX'
    elif(dec3 == 240): # Netbios -> F0
        sap = 'NETBIOS'
```

Ako ďalšie čítam bajty na zistenie SAP-u a PID-u.

```
if(frameType == 'Ethernet II'):
    eth_type = ''
    val = str(x)
    for key, value in jsonData['eth_type'].items():
        if(key == val):
            eth_type = value
            if(eth_type == 'IPv4'):
                srcIP = str(int((frame[52] + frame[53]), 16)) + "." + str(int((frame[54] + frame[55]), 16)) + "." + str(int((frame[56] + frame[57]), 16)) + "." + str(int((frame[58] + frame[59]), 16))
                dstIP = str(int((frame[60] + frame[61]), 16)) + "." + str(int((frame[62] + frame[63]), 16)) + "." + str(int((frame[64] + frame[65]), 16)) + "." + str(int((frame[66] + frame[67]), 16))
                senders_ip.append(srcIP)
                for key, value in jsonData['protocols'].items():
                    if(key == str(frame[46] + frame[47])):
                        protocol = value
                if(protocol == 'UDP'):
                    src_port = int((frame[68] + frame[69] + frame[70] + frame[71]), 16)
                    dst_port = int((frame[72] + frame[73] + frame[74] + frame[75]), 16)
                    for key, value in jsonData['app_protocol'].items():
                        if(key == str(src_port) or key == str(dst_port)):
                            app_protocol = value
                if(protocol == 'TCP'):
                    src_port = int((frame[68] + frame[69] + frame[70] + frame[71]), 16)
                    dst_port = int((frame[72] + frame[73] + frame[74] + frame[75]), 16)
                    for key, value in jsonData['app_protocol'].items():
                        if(key == str(src_port) or key == str(dst_port)):
                            app_protocol = value
            if(eth_type == 'ARP'):
                srcIP = str(int((frame[56] + frame[57]), 16)) + "." + str(int((frame[58] + frame[59]), 16)) + "." + str(int((frame[60] + frame[61]), 16)) + "." + str(int((frame[62] + frame[63]), 16))
                dstIP = str(int((frame[76] + frame[77]), 16)) + "." + str(int((frame[78] + frame[79]), 16)) + "." + str(int((frame[80] + frame[81]), 16)) + "." + str(int((frame[82] + frame[83]), 16))
```

Reader Mode

Následne ak sa jedná o Ethernet II tak zisťujem vo viacerých podmienkach vnorený protokol v hlavičke rámca, kde mám potom ďalšie podmienky ktoré zisťujú cieľový port kde potom prechádzam môj .json file a zisťujem či sa nejedná o nejaký známy port.

```
# Hexa frame
hexaFrame = ''
i = 1;
for z in frame:
    if not(i%32==0):
        if(i != len(frame)):
            if(i%2 == 0):
                hexaFrame += z + " "
                # print(z, end=" ")
            else:
                hexaFrame += z
                # print(z, end="")
    if(i == len(frame)):
        hexaFrame += z + "\n"
        break
    if(i%32 == 0):
        hexaFrame += z + "\n"
        # print(z,  end="\n")
    i = i+1
```

Ako ďalšie som čítal údaje pre hex frame.

```
if(frameType == 'Ethernet II'):
    if(eth_type == 'ARP'):
        packet_data = {'frame_number': frameNumber, 'len_frame_pcap': frameLenght, 'len_frame_medium': frameMedium,
                       'frame_type': frameType, 'src_mac': srcMac, 'dst_mac': dstMac, "ether_type": eth_type, 'src_ip': srcIP, 'dst_ip': dstIP,
                       'hexa_frame': ruamel.yaml.scalarstring.LiteralScalarString(hexaFrame)}
    if(eth_type == 'IPv4'):
        if(protocol == 'TCP' or protocol == 'UDP'):
            if not(app_protocol == ''):
                packet_data = {'frame_number': frameNumber, 'len_frame_pcap': frameLenght, 'len_frame_medium': frameMedium,
                               'frame_type': frameType, 'src_mac': srcMac, 'dst_mac': dstMac, "ether_type": eth_type, 'src_ip': srcIP, 'dst_ip': dstIP, 'protocol': protocol,
                               'src_port': src_port, 'dst_port': dst_port, 'app_protocol': app_protocol, 'hexa_frame': ruamel.yaml.scalarstring.LiteralScalarString(hexaFrame)}
            else:
                packet_data = {'frame_number': frameNumber, 'len_frame_pcap': frameLenght, 'len_frame_medium': frameMedium,
                               'frame_type': frameType, 'src_mac': srcMac, 'dst_mac': dstMac, "ether_type": eth_type, 'src_ip': srcIP, 'dst_ip': dstIP, 'protocol': protocol,
                               'src_port': src_port, 'dst_port': dst_port,'hexa_frame': ruamel.yaml.scalarstring.LiteralScalarString(hexaFrame)}
        else:
            packet_data = {'frame_number': frameNumber, 'len_frame_pcap': frameLenght, 'len_frame_medium': frameMedium,
                           'frame_type': frameType, 'src_mac': srcMac, 'dst_mac': dstMac, "ether_type": eth_type, 'src_ip': srcIP, 'dst_ip': dstIP, 'protocol': protocol,
                           'hexa_frame': ruamel.yaml.scalarstring.LiteralScalarString(hexaFrame)}
    if not(eth_type == 'IPv4' or eth_type == 'ARP'):
        packet_data = {'frame_number': frameNumber, 'len_frame_pcap': frameLenght, 'len_frame_medium': frameMedium,
                       'frame_type': frameType, 'src_mac': srcMac, 'dst_mac': dstMac, "ether_type": eth_type,
                       'hexa_frame': ruamel.yaml.scalarstring.LiteralScalarString(hexaFrame)}
if(frameType == 'IEEE 802.3 LLC & SNAP'):
    packet_data = {'frame_number': frameNumber, 'len_frame_pcap': frameLenght, 'len_frame_medium': frameMedium,
                   'frame_type': frameType, 'src_mac': srcMac, 'dst_mac': dstMac, 'pid': pid,
                   'hexa_frame': ruamel.yaml.scalarstring.LiteralScalarString(hexaFrame)}
if(frameType == 'IEEE 802.3 LLC'):
    packet_data = {'frame_number': frameNumber, 'len_frame_pcap': frameLenght, 'len_frame_medium': frameMedium,
                   'frame_type': frameType, 'src_mac': srcMac, 'dst_mac': dstMac, 'sap': sap,
                   'hexa_frame': ruamel.yaml.scalarstring.LiteralScalarString(hexaFrame)}
if not(frameType == 'Ethernet II' or frameType == 'IEEE 802.3 LLC & SNAP' or frameType == 'IEEE 802.3 LLC'):
    packet_data = {'frame_number': frameNumber, 'len_frame_pcap': frameLenght, 'len_frame_medium': frameMedium,
```

Následne všetky tieto údaje zapisujem do jednej premennej, ktorú potom posielam do poľa packetov, kde potom to pole zapisujem do yaml súboru.

```
for key, value in Counter(senders_ip).items():
    senders_data = {"node": key, "number_of_sent_packet": value}
    data["IPv4_senders"].append(senders_data)

number_of_packets = Counter(senders_ip)
best_senders_list = list(filter(lambda t: t[1]>=number_of_packets.most_common()[0][1], number_of_packets.most_common()))
for key, value in best_senders_list:
    max_senders_data = {"node": key, "number_of_sent_packet": value}
    data["max_send_packets_by"].append(max_senders_data)
```

A ako posledné si zapisujem IP adresu ktorá posiela dáta, kde potom zapisujem do súboru ktorá IP adresa odoslala najviac packetov.

3.2 Funkcia na výpis kompletných a nekompletných TCP komunikácií

Na výpis kompletných a nekompletných TCP komunikácií som implementoval dve funkcie.

```
# Ethernet II type
app_protocol = ''
if(frameType == 'Ethernet II'):
    eth_type = ''
    val = str(x)
    for key, value in jsonData['eth_type'].items():
        if(key == val):
            eth_type = value
            if(eth_type == 'IPv4'):
                srcIP = str(int((frame[52] + frame[53]), 16)) + "." + str(int((frame[54] + frame[55]), 16)) + "." + str(in
                dstIP = str(int((frame[60] + frame[61]), 16)) + "." + str(int((frame[62] + frame[63]), 16)) + "." + str(in
                for key, value in jsonData['protocols'].items():
                    if(key == str(frame[46] + frame[47])):
                        protocol = value
                if(protocol == 'TCP'):
                    src_port = int((frame[68] + frame[69] + frame[70] + frame[71]), 16)
                    dst_port = int((frame[72] + frame[73] + frame[74] + frame[75]), 16)
                    for key, value in jsonData['app_protocol'].items():
                        if(key == protocol_port and (dst_port == int(key) or src_port == int(key))):
                            # print(str(number) + " " + " " + str(srcIP) + " " + " " + str(dstIP))
                            tcp_frames.append(packet)
                            number_tcp_frames.append(number)
```

V prvej funkcií robí program presne to isté čo vo funkcií predtým, čiže zisťuje si potrebné údaje z jednotlivých bajtov, následne ak zistí že je to Ethernet II nad TCP a s protokolom , ktorý užívateľ zadal na vstupe, tak tento frame pridám do poľa, kde sa nachádzajú všetky frame-y, ktoré spĺňajú tieto podmienky. Následne toto pole posielam do ďalšej rekurzívnej funkcie.

```
def tcp_frames(http_frames, number_http_frames, complete_frames, number_frames, fileName, protocol_name):
    communication = []
    frame_numbers = []
    if (len(http_frames) > 0):
        b = raw(http_frames[0])
        frame = b.hex()
        frameSplit = frame.split()

        main_src_port = int((frame[68] + frame[69] + frame[70] + frame[71]), 16)
        main_dst_port = int((frame[72] + frame[73] + frame[74] + frame[75]), 16)

        new_arr = []
        pom = 0
        for packet in http_frames:
            # print(len(http_frames))
            p = raw(packet)
            # Split frame
            frame = p.hex()
            frameSplit = frame.split()

            src_port = int((frame[68] + frame[69] + frame[70] + frame[71]), 16)
            dst_port = int((frame[72] + frame[73] + frame[74] + frame[75]), 16)


            if((src_port == main_dst_port and dst_port == main_src_port) or (src_port == main_src_port and dst_port == main_dst_port)):
                # (src_port == main_dst_port or src_port == main_src_port) and (dst_port == main_src_port or dst_port == main_dst_port)
                # src_port_int == dst_ports[0] and dst_port_int == src_ports[0]) or (src_port_int == src_ports[0] and dst_port_int == dst_ports[0]
                communication.append(packet)
                # frame_numbers.append(number_http_frames[pom])
                number_frames.append(number_http_frames[pom])
                # http_frames.remove(packet)
            pom = pom + 1
```

V ďalšej funkcii už pracujem len s polom frame-ov, ktoré majú daný protokol (napr. HTTP). Ako prvé mám podmienku, či sa v poli frame-ov nachádza ešte nejaký frame, ak je podmienka splnená tak si zoberiem prvý frame tohto poľa, a nastavím si jeho source a destination porty ako main (hlavné) porty. Následne prechádzam celé pole po frame-och

a ak sa porty toho frame-u rovnajú hlavným portom, tak pridávam tento frame do pola, pre danú komunikáciu.

```
new_arr = []
numbers_arr = []
pom = 0
for packet in http_frames:
    # print(len(http_frames))
    p = raw(packet)
    # Split frame
    frame = p.hex()
    frameSplit = frame.split()

    src_port = int((frame[68] + frame[69] + frame[70] + frame[71]), 16)
    dst_port = int((frame[72] + frame[73] + frame[74] + frame[75]), 16)


    if not((src_port == main_dst_port or src_port == main_src_port) and (dst_port == main_src_port or dst_port == main_dst_port)):
        # (src_port == main_dst_port or src_port == main_src_port) and (dst_port == main_src_port or dst_port == main_dst_port)
        # src_port_int == dst_ports[0] and dst_port_int == src_ports[0]) or (src_port_int == src_ports[0] and dst_port_int == dst_ports[0]
        # communication.append(packet)
        # http_frames.remove(packet)
        new_arr.append(packet)
        numbers_arr.append(number_http_frames[pom])
    pom = pom + 1
```

Následne tie frames, ktoré som vložil do poľa danej komunikácie som vymazal z poľa kde sa nachádzajú všetky frames s daným protokolom. A potom som vložil toto pole kde sú frames len danej komunikácie, do poľa kde sa nachádzajú všetky komunikácie. Potom som opäť zavolal túto funkciu.

```
else:
    data = {"name": "PKS2022/23", "pcap_name": fileName, "filter_name": protocol_name, "complete_comms": [], "partial_comms": [] }
    pom = 0
    comm_number = 1
    for packet in complete_frames:
        frames = []
        fin_ack = 0
        rst = 0
        rst_ack = 0
        flags = []
        pom2 = 0
        src_dst = 0
        for p in packet:
            b = raw(p)
            frame = b.hex()
            frameSplit = frame.split()

            flag = frame[94] + frame[95]
            flags.append(flag)

            if(flag == '11' or flag == '19'):
                fin_ack = fin_ack + 1
            if(flag == '04'):
                rst = rst + 1
            if(flag == '14'):
                rst_ack = rst_ack + 1

            main_src_port = int((frame[68] + frame[69] + frame[70] + frame[71]), 16)
            main_dst_port = int((frame[72] + frame[73] + frame[74] + frame[75]), 16)
            # print(str(number_frames[pom]) + ": " + flags[pom2])
```

Ak nebola splnená prvá podmienka, to znamená, že som prešiel celé pole tak som prešiel do else vetvy, kde prechádzam pole komunikácií, následne prechádzam všetky frame-y danej komunikácie, zistím si všetky potrebné informácie pre výpis o danom frame (mac adresy, ip adresy, porty atď) vrátane flagu, ktorý si ukladám do poľa flagov pre danú komunikáciu a potom idem zisťovať či je komunikácia kompletná alebo nie.

```
if(len(flags) > 3):
    if((flags[0] == '02' and flags[1] == '12' and flags[2] == '10') and (fin_ack == 2 or rst == 1 or rst_ack == 1)):
        comm = {"number_comm": comm_number, "src_comm": src_comm, "dst_comm":dst_comm, "packets": frames}
        comm_number = comm_number + 1
        data["complete_comms"].append(comm)
        print("Complete communication")
    else:
        comm = {"number_comm": comm_number, "packets": frames}
        comm_number = comm_number + 1
        data["partial_comms"].append(comm)
        print("Incomplete communication")
else:
    comm = {"number_comm": comm_number, "packets": frames}
    comm_number = comm_number + 1
    data["partial_comms"].append(comm)
    print("Incomplete communication")
```

Zisťovanie kompletnosti a nekompletnosti funguje tak, že zisťujem vďaka podmienke či sa v poli flagov nachádzajú flagy na otvorenie komunikácie a zatvorenie zisťujem pomocou toho či sa v poli flagov nachádza dva krát flag fin alebo raz rst. Ak je podmienka splnená pridávam komunikáciu do kompletných, ak nie pridávam komunikáciu do nekompletných.

3.2 Funkcia na výpis UDP komunikácií

Na výpis UDP komunikácií používam opäť dve funkcie z toho jedna je rekurzívna.

```
def udp_protocols(packet_list, file_name, protocol):
    # Open json file
    f = open('data.json')
    jsonData = json.load(f)
    for key, value in jsonData['app_protocol'].items():
        if(value == protocol):
            protocol_port = key
            protocol_name = value
            print(value)

    frame_number = 1

    arr_number = []
    number_tftp_frame = 0
    pom = 1
    for p in packet_list:
        b = raw(p)
        # Split frame
        frame = b.hex()
        frameSplit = frame.split()

        src_port = int((frame[68] + frame[69] + frame[70] + frame[71]), 16)
        dst_port = int((frame[72] + frame[73] + frame[74] + frame[75]), 16)

        if(str(src_port) == protocol_port or str(dst_port) == protocol_port):
            number_tftp_frame = number_tftp_frame + 1
        arr_number.append(pom)
        pom = pom + 1


    print(number_tftp_frame)
    tftp_frames = []
    tftp_numbers = []
    comms = []
    udp_frames(packet_list, file_name, protocol_port, protocol_name, frame_number, tftp_frames, number_tftp_frame, arr_number, tftp_numbers, comms)
```

Prvá funkcia slúži iba nato, aby som si vytvoril pole s číslami frame-ov ktoré obsahujú zadaný protokol od užívateľa. Na konci tejto funkcie volám druhú funkciu, ktorá je rekurzívna.

```
def udp_frames(packet_list, file_name, protocol_port, protocol_name, frame_number, tftp_frames, number_tftp_frame, arr_number, tftp_numbers, comms):
    actual_frames = []
    if(number_tftp_frame > 0):
        start_frame = 0
        main_port = 0
        new_arr = []
        new_arr_numbers = []
        frame_numbers = []
        pom = 0
        pom2 = 0
        for p in packet_list:
            b = raw(p)
            # Split frame
            frame = b.hex()
            frameSplit = frame.split()

            src_port = int((frame[68] + frame[69] + frame[70] + frame[71]), 16)
            dst_port = int((frame[72] + frame[73] + frame[74] + frame[75]), 16)

            udp_protocol = str(frame[46] + frame[47])

            if(start_frame == 1 and (src_port == main_port or dst_port == main_port) and udp_protocol == '11'):
                actual_frames.append(p)
                frame_numbers.append(frame_number)
                tftp_numbers.append(arr_number[pom2])
                pom = pom + 1

            if(str(dst_port) == protocol_port and start_frame == 0):
                actual_frames.append(p)
                main_port = src_port
                start_frame = 1
                frame_numbers.append(frame_number)
```

V rekurzívnej funkcii mám ako prvé podmienku, ktorá zisťuje počet ešte neprejdených frame-ov s daným protokol, na konci toto číslo vždy dekrementujem. Následne prechádzam všetky frame-y daného .pcap súboru, ak som našiel frame ktorý má port pre daný protokol (69 – TFTP), tak tento frame pridám do poľa danej komunikácie, následne prechádzam ďalšie frame-y a ak zistím že frame má ten istý port ako frame, ktorý posielal údaje na port známeho protokolu ktorého názov zadal používateľ (v tomto prípade 69 - TFTP), tak pridám aj tento frame do poľa pre danú komunikáciu.
Ak už som prešiel všetky frame-y, tak pridám komunikáciu do poľa všetkých komunikácií, ktoré chcem vypísať (v tomto prípade s protokolom TFTP).

```
pom2 = 0
for p in packet_list:
    b = raw(p)
    # Split frame
    frame = b.hex()
    frameSplit = frame.split()

    src_port = int((frame[68] + frame[69] + frame[70] + frame[71]), 16)
    dst_port = int((frame[72] + frame[73] + frame[74] + frame[75]), 16)

    if not(src_port == main_port or dst_port == main_port):
        new_arr.append(p)
        new_arr_numbers.append(arr_number[pom2])

    pom2 = pom2 + 1

comms.append(actual_frames)
number_tftp_frame = number_tftp_frame - 1
udp_frames(new_arr, file_name, protocol_port, protocol_name, frame_number, tftp_frames, number_tftp_frame, new_arr_numbers, tftp_numbers, comms)
```

Ako ďalšie vymažem všetky frame-y, ktoré som pridal do aktuálnej komunikácie, a to tak že mi vznikne nové pole packetov, ktoré posielam do volanie tejto danej funkcie, pretože je rekurzívna.

```
else:
    ...
    data = {"name": "PKS2022/23", "pcap_name": fileName, "filter_name": protocol_name, "complete_comms": [] }
    pom = 0
    comm_number = 1

    for packet in comms:
        frames = []
        for p in packet:
            b = raw(p)
            frame = b.hex()
            frameSplit = frame.split()

            main_src_port = int((frame[68] + frame[69] + frame[70] + frame[71]), 16)
            main_dst_port = int((frame[72] + frame[73] + frame[74] + frame[75]), 16)
            # print(str(number_frames[pom]) + ": " + flags[pom2])

            # destination mac
            dst_mac = str(frame[0]) + str(frame[1]) + ':' + str(frame[2]) + str(frame[3]) + ':' + str(frame[4]) + str(frame[5]) + ':' + str(frame[6]) + str(frame[7]) + ':' + str(frame[8])
            # source mac
            src_mac = str(frame[12]) + str(frame[13]) + ':' + str(frame[14]) + str(frame[15]) + ':' + str(frame[16]) + str(frame[17]) + ':' + str(frame[18]) + str(frame[19]) + ':' + str(fr

            # get frame lenght
            frameLenght = int(len(p))

            # Get frame medium
            if(frameLenght > 60):
                frameMedium = int(len(p)) + 4
            else:
                frameMedium = 64
```

Ak nebola splnená prvotná podmienka, to znamená že už som prešiel všetky frame-y, tak idem do vetvy else, ktorej prechádzam pole komunikácií, následne prechádzam každý frame v danej komunikácií, a opäť ako vždy zisťujem údaje o danom frame-e.

```
        packet_data = {"frame_number": tftp_numbers[pom], "len_frame_pcap": frameLenght, "len_frame_medium": frameMedium,
                       "frame_type": "Ethernet II", "src_mac": src_mac, "dst_mac": dst_mac, "ether_type": "IPv4", "src_ip": srcIP,
                       "dst_ip": dstIP, "protocol": "TCP", "src_port": main_src_port, "dst_port": main_dst_port,
                       "app_protocol": protocol_name, "hexa_frame": ruamel.yaml.scalarstring.LiteralScalarString(hexaFrame)}
        frames.append(packet_data)
        pom = pom + 1
    comm = {"number_comm": comm_number, "packets": frames}
    comm_number = comm_number + 1
    data["complete_comms"].append(comm)



yaml = ruamel.yaml.YAML()
with open('out.yaml', 'w') as f:
    data = yaml.dump(data, f)
```

Ako posledné už iba všetky tieto údaje zapisujem do súboru yaml.

4 Štruktúra externých súborov

Externý súbor z ktorého čítam údaje a dáta mám typu .json. Čítam z neho údaje o vnorenom protokole hlavičky rámca typu Ethernet II, vnorené protokoly a tak isto aj čísla a názvy známych portov.

```json
{
  "eth_type": {
    "0800": "IPv4",
    "86dd": "IPv6",
    "0806": "ARP",
    "88cc": "LLDP",
    "9000": "ECTP"
  },
  "protocols": {
    "11": "UDP",
    "06": "TCP",
    "02": "IGMP",
    "01": "ICMP",
    "67": "PIM"
  },
  "app_protocol": {
    "20": "FTP-DATA",
    "21":"FTP-CONTROL",
    "22": "SSH",
    "23": "TELNET",
    "24": "SMPT",
    "53": "DNS",
    "80": "HTTP",
```

5 Používateľské rozhranie

Ako používateľské rozhranie som zvolil klasické jednoduché menu, ktoré vypisujem do konzoly, kde následne aj používateľ vypisuje dané údaje.

```
Vyber si moznost:
1 - vypis vsetkych ramcov
2 - protokol s komunikaciou so spojenim (TCP)
3 - protokol s komunikaciou bez spojenia (UDP)
2
Zadaj protokol (HTTP, HTTPS, TELNET, SSH, FTP-CONTROL, FTP-DATA):HTTP
Zadaj nazov suboru: trace-10.pcap
```

Ako prvé si od používateľa vypýtam aby zvolil z možností, či chce vypísať všetky frames, alebo chce vypísať TCP alebo UDP komunikácie. Následne si vypýtam aby používateľ napísal protokol ktorý chce filtrovať a ako posledné musí užívateľ ešte napísať názov .pcap súboru ktorý chce analyzovať.

6 Knižnice a importy

```
import sys
sys.path.append("c:/users/adam4/appdata/roaming/python/python39/site-packages")
from scapy.all import *
from collections import Counter
from scapy. layers. inet import IP
sys.path.append("c:/python39/lib/site-packages")
import yaml
import ruamel.yaml
import json
```

7 Záver

Navrhnutý algoritmus je z pohľadu výkonnosti a rýchlosti pomerne efektívny. Myslím si že mnou navrhnuté riešenie rekurzívnymi funkciami je vcelku praktické.

Riešenie je implementované tak, že ak externý .json súbor rozšírime o ďalšie protokoly a porty, tak program bude stále fungovať, bez akejkoľvek úpravy, takže riešenie sa môže rozšíriť o akékoľvek ďalšie porty.

Algoritmus by sa dal rozšíriť ešte napríklad o analýzu ARP alebo ICMP komunikácií.