PKS Zadanie 1

Matúš Makay

STU FIIT

Úvod

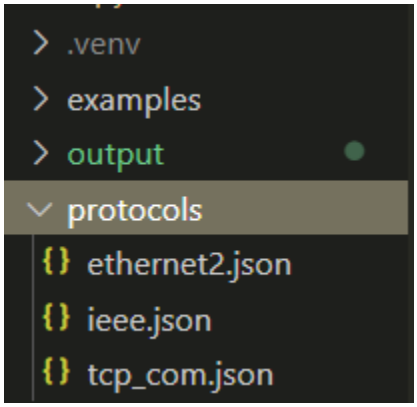
Cieľom tohto zadania bolo vypracovať analyzátor komunikácie v sieti. Vypracovaval som ho v sulade so zadanymi materialmi. Informacie som cerpal z prednasok a internetu.

Obsah

Vytvaranie class a nastavovanie atributov, dictionary

Nacitavanie dictionary-s z externych suborov

Pri rieseni problemu nastavovania potrebnych atributov som dospel k zaveru ze najlepsie riesenie bude ak budem pouzivat konretne hodnoty z hexa stringu ako key hodnoty v dictionary-s ktorym bude priradena hodnota konkretneho nazvu protokolu. Tieto dictionary-s som ulozil v externych suborov z ktorych si ich nacitavam podla potreby. Pre kazdy ether type mam vlastny dictionary. Subory su ulozene v priecinku protocols.



Metoda pre nacitanie dictionary-s

```
def create_eth_dics(eth_path, ieee_path):
    """
    return a dictionaris, first are eth_dictionaris, second are ieee dictionaris.
    eth_dics = [ipv_versions], ieee_dics = [FRAME_TYPE, SAP, PID]
    """
    eth_file = open(eth_path)
    ieee_file = open(ieee_path)
    # [0] = ipv
    dic_list_eth = json.load(eth_file)

    # [0] = RAW,LLC & SNAP [1] = SAP, [2] = PID
    dic_list_ieee = json.load(ieee_file)

    eth_file.close()
    ieee_file.close()

    return dic_list_eth, dic_list_ieee
```

Postup vytvaranie class-y je nasledovny:

1. Poslem do konstruktora kazdej classy prislusne dictionary-s ktore si nasledne ulozim pre dalsie pouzitie
2. Atributy ktore maju vsetky classy spolocne(MAC, dlzka) zistujem mimo classy a posielam ich ako parametre do konstruktora.
3. Z konstruktora zavolam prislusne metody na priradenie atributov

Class EthernetII

Pre ethernetove class-y su dolezite directory-s

1. Directory ktory obsahuje nazvy ether typu, protokolov a aplikacnych protokolov

```
{
    "ether_type" : {
        "0800" : "Ipv4",
        "86dd" : "Ipv6",
        "0806":"ARP",
        "9000": "ECTP",
        "88cc":"LLDP"
    },
    "protocols": {
        "01":"ICMP",
        "06":"TCP",
        "02":"IGMP",
        "11":"UDP"
    },
    "app_protocols": {
        "20":"FTP-DATA",
        "21":"FTP-CONTROL",
        "22":"SSH",
        "23":"TELNET",
        "25":"SMTP",
        "53":"DNS",
        "80":"HTTP",
        "110":"POP3",
        "119":"NNTP",
        "139":"NETBIOS-SSN",
        "143":"IMAP",
        "179":"BGP",
        "389":"LDAP",
        "443":"HTTPS",
        "123":"TIME",
        "67":"DHCP",
        "69":"TFTP",
        "137":"NETBIOS-NS",
        "138":"NETBIOS-DGM",
        "161":"SNMP",
        "162": "SNMP-TRAP",
        "514": "SYSLOG",
        "520":"RIP",
        "33434":"TRACEROUTE"}}
```

2. Directory ktory riadi vytvaranie jednotlivych framov sa nachadza ako atribut kazdej ethernet classy, key hodnota je ether_type daneho paketu. Value je odkaz na metodu ktora nastavi prislusne atributy.

Konstruktor

```
def __init__():
        #vymazal som jednotlive atributy aby zbytocne nezaberali miesto

        #directory ktore classa pouziva
        self.eth_type_dic = eth_type_dic
        self.protocols_dic = protocols_dic
        self.app_protocols_dic = app_protocols_dic

        self.set_ether_type(hex_frame)

        self.cr_ether_type_dic = {
            #nezistujes ip_adresy
            "LLDP": self.cr_lldp_etcp_ipv6,
            "ECTP": self.cr_lldp_etcp_ipv6,
            "Ipv6": self.cr_lldp_etcp_ipv6,
            #ip na inom mieste
            "ARP": self.cr_arp,
            #ipv4
            "Ipv4": self.cr_ipv4,
        }

        self.create_frame(self.cr_ether_type_dic, hex_frame)
```

Metoda create frame

```
def create_frame(self, dic, hex_frame):
        if(self.ether_type != False):
            dic[self.ether_type](hex_frame)
```

Arp paket

```
def cr_arp(self, hex_frame):
        self.set_ip_arp(hex_frame)
```

Ipv4 paket

```
def cr_ipv4(self, hex_frame):
        self.set_protocol(hex_frame)
        self.set_ip_ipv4(hex_frame)
        self.set_port(hex_frame)
        self.set_flags(hex_fram
```

Class IeeeFrame

Pre IEEE 802.3 class-y je dolezity directory

```
{
    "ether_type":{
        "aa":"LLC & SNAP",
        "ff":"RAW"

    },
    "sap": {
        "4242": "STP",
        "f0f0": "NETBIOS",
        "e0e0": "IPX",
        "0000": "STP"
    },
    "pid": {
        "2000":"CDP",
        "010b":"PVSTP+",
        "809b":"APPLETALK",
        "2004": "DTP"
    }
}
```

Konstruktor

```
def __init__():
        #dictionarys ktore classa potrebuje, nacitane z ext. subora
        self.frame_type_dic = frame_type_dic
        self.sap_dic = sap_dic
        self.pid_dic = pid_dic

        self.set_frame_type(hex_code, hex_frame)
```

Metoda set_frame_type

V tejto metode sa pokusim setnut atr. Frame type pomocou dictionary, pricom viem ze ak sa hex_code ako key v dictionary nenachadza. Packet bude typu LLC.

```
def set_frame_type(self, hex_code, hex_frame):
        self.frame_type = "IEEE 802.3 "

        try:
            self.frame_type += self.frame_type_dic[hex_code]
       #ak llc potom setujem sap
        except KeyError:
            self.frame_type += "LLC"
            self.set_sap(hex_frame)
        #ak llc a snap tak setujem pid
        if self.frame_type == "IEEE 802.3 LLC & SNAP":
            self.set_pid(hex_frame)
```

Class FrameCreator

Tato class-a zabezpecuje jednoduche vytvaranie ethernetII alebo ieee. Deje sa to na zaklade ukazovania metod pomocou atr. dic kde key je ether type a value eje prislusna metoda ktora bude zavolana a return-e vytvorenu classu s prislusnymi atributmi.

```
class FrameCreator:
    def __init__(self, eth_dics, ieee_dics):
        self.dic = {
            "IEEE 802.3": self.cr_ieee_frame,
            "ETHERNET II": self.cr_eth_frame,
        }
    def search_eth_type_dic(self, type, count, src_adr, dst_adr, len_pycap,
len_medium, hex_code):
        return self.dic[type](count, src_adr, dst_adr, len_pycap, len_medium,
hex_code)
def cr_eth_frame(self, count, src_adr, dst_adr, len_pycap, len_medium, hex_code):
        return Ethernet2Frame()

    def cr_ieee_frame(self, count, src_adr, dst_adr, len_pycap, len_medium,
hex_code):
        return IeeeFrame(
```

Class FrameWriter

Tato class-a zabezpecuje vytvorenie spravneho formatu pre zapis do yaml filu. Princip ako vyuzivam tuto classu: ze

1. zavolam metodu init_ul_x nastavim prislusnu hlavicku pre dany yaml file.
2. zavolam metodu write_in_file_ulx ktorej poslem listy ktore chcem zapisat.

Kedze rozne pakety maju rozny format, znova ako vsade v mojom projekte pre zjednodusenie pouzivam rozne dictionary-s ktore ukazuju na metody a s ktorymi si toto vytvaranie ulahcujem. Kedze pre ethernetove pakety je viacej moznosti vytvoril som dalsi eth_dic ktory riesi prave ethernetove pakety.

Konstruktor

```
    count = 1
    def __init__(self):

        self.dic = {
            "IEEE 802.3 RAW": self.create_dic_raw,
            "IEEE 802.3 LLC": self.create_dic_llc,
            "IEEE 802.3 LLC & SNAP": self.create_dic_llc_and_snap,
            "ETHERNET II": self.create_dic_eth,
        }

        self.eth_dic = {
            "LLDP": self.cr_lldp_etcp_ipv6,
            "ECTP": self.cr_lldp_etcp_ipv6,
            "Ipv6": self.cr_lldp_etcp_ipv6,
            #ip na inom mieste
            "ARP": self.cr_arp,
            #ipv4
            "Ipv4": self.cr_ipv4,
        }
        #finalny dictionary ktory po pridani vsetkych paketov zapisem do yaml-u
        self.dic_wr_in_file = {}
        #yaml konfiguracia
        self.yaml = YAML()
        self.yaml.default_flow_style = False
        self.yaml.representer.add_representer(str, repr_str)
```

Funkcia create_dic_eth

```
    def create_dic_eth(self, frame):
        try:
            return self.eth_dic[frame.ether_type](frame)
        except KeyError:
            return
```

Rozhodovanie ktory dictionary sa ma vytvorit

```
for pckt in packets:
     self.count += 1
     list_dic.append(self.dic[pckt.frame_type](pckt)
```

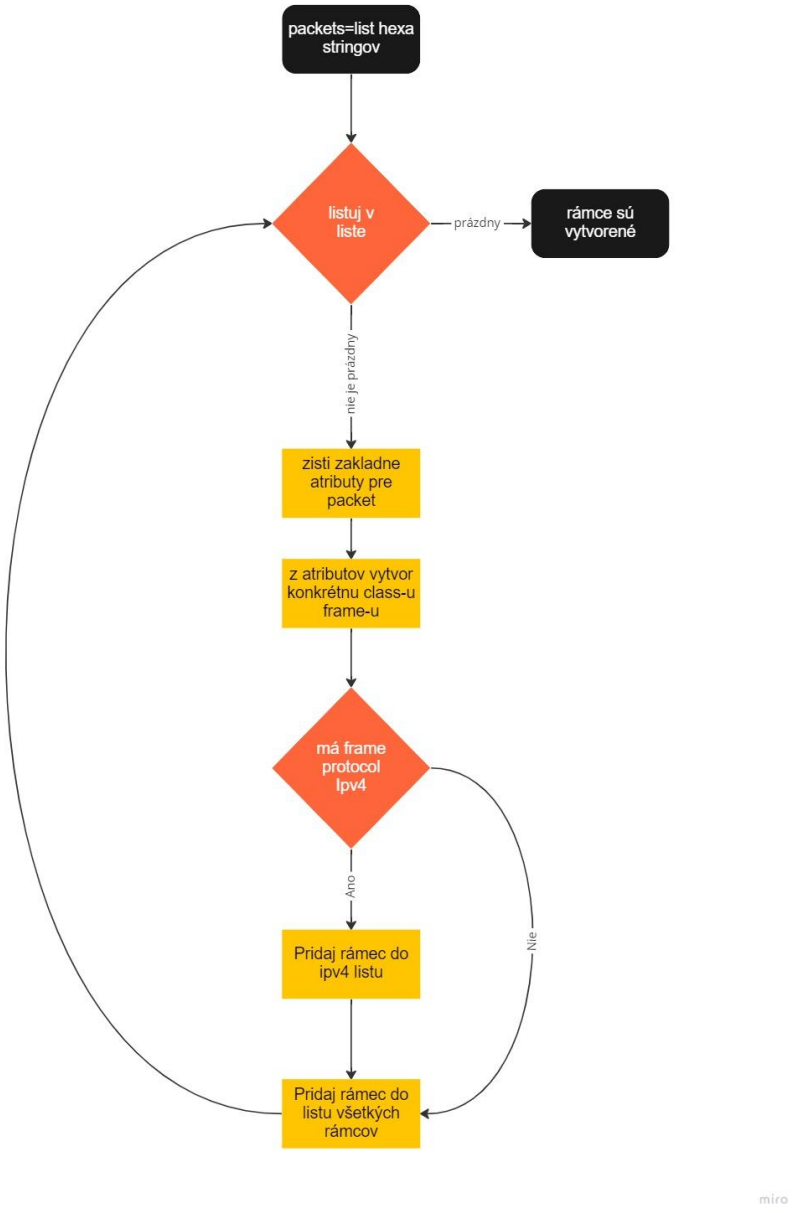
Priklad vytvorenia dictionary pre Ipv4

```
def cr_ipv4(self, frame):
        dic = {}
        dic["frame_number"] = frame.frame_number
        dic["len_frame_pcap"] = frame.len_frame_pcap
        dic["len_frame_medium"] = frame.len_frame_medium
        dic["frame_type"] = frame.frame_type
        dic["src_mac"] = frame.src_mac
        dic["dst_mac"] = frame.dst_mac.strip()
        dic["ether_type"] = frame.ether_type
        dic["src_ip"] = frame.src_ip
        dic["dst_ip"] = frame.dst_ip
        dic["protocol"] = frame.protocol
        dic["src_port"] = frame.src_port
        dic["dst_port"] = frame.dst_port
        dic["app_protocol"] = frame.app_protocol
        dic["hexa_frame"] = frame.hexa_frame

        return dic
```

Postup pre vytvorenie paketov

Vytvaranie listov z paketov



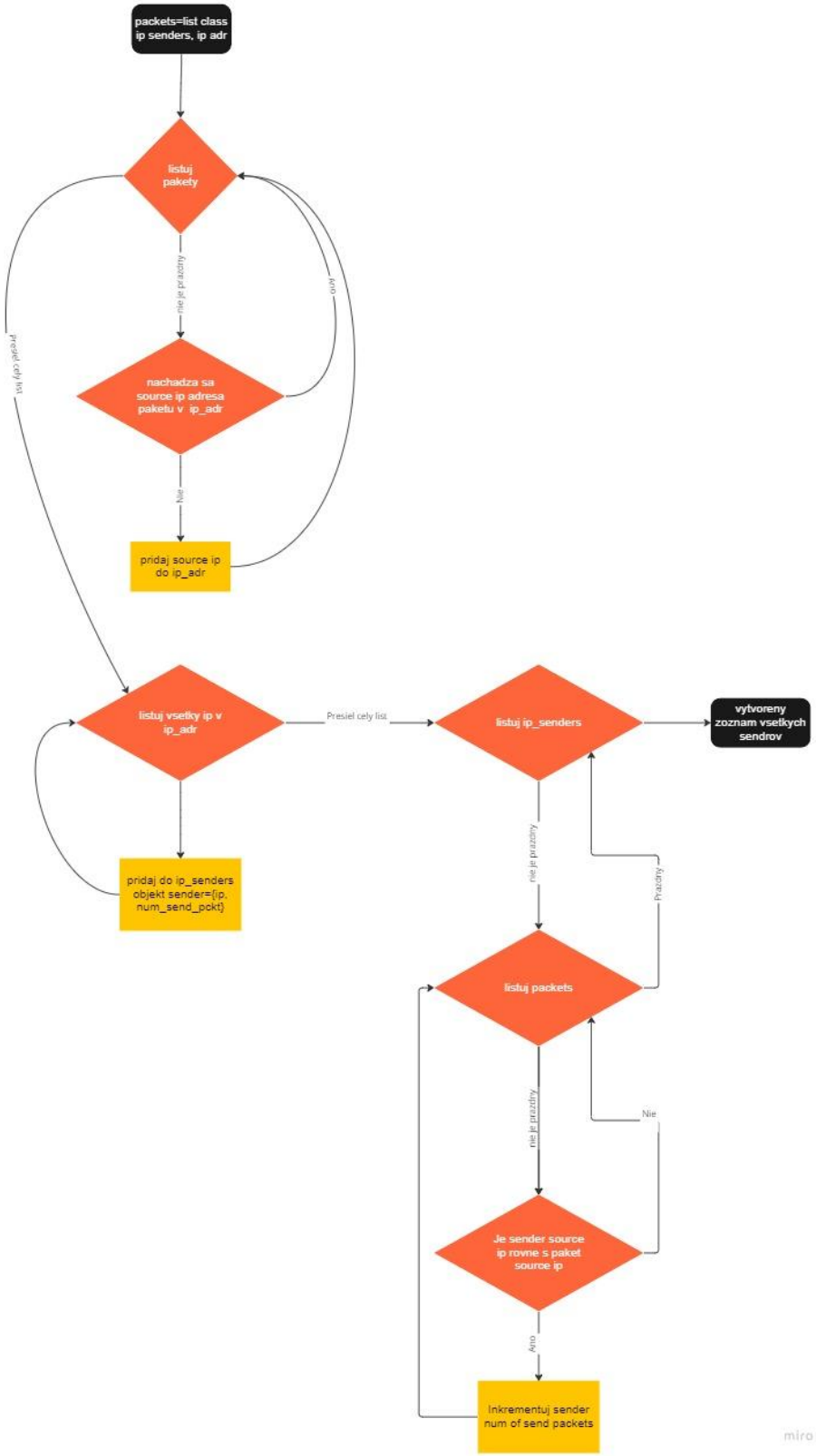
Slovny opis:

Nacitam si zo suboru pcap stringy ktore nacitam do listu packets. Nasledne vo for cykle prechadzam kazdy paket pre ktory zistim zakladne atributy a podla typu framu vytvorim

prislusnu classu. Bud ieee alebo ethernet. Taktiez sa hned pytam ci je ma paket ether type Ipv4, ak ano pridam ho do listu ipv4 paketov ktory budem neskor kvoli statistickej ulohe.

- Základné atribúty= zdrojová, cieľová mac adresa, dĺžka rámca, dĺžka rámca ktorý pretiekol cez médium, a typ rámca=IEE 802.3 alebo ETHERNET II
    - zdrojová, cieľová mac adresa: zistujem na prvych 24b hexa stringu
    - dĺžka rámca, dĺžka rámca ktorý pretiekol cez médium: dlzku ramca zistujem pomocou dlzky hexa stringu ktoru vydelim dvomi aby som dostal vysledok v B. Ak je dlzka ramca menej ako 60 potom dlzka pretecenych B po mediu bude 64 pretoze to je najmenej co moze tiect po sieti. Ak dlzka ramca je viac ako 60 potom pripocitam k dlzke ramca 4 a dostanem dlzku ramca ktory pretecie po mediu.
    - typ ramca zistujem pomocou 24 az 28b v hexa stringu ktory prevediem zo 16 na decimalnu hodnotu a podla nej urcim ci je paket IEE alebo ETHERNETII. 1536 || >1500 potom je ETHERNET inak je paket typu IEEE.

Vytvaranie statistiky odoslanych paketov pre Ipv4



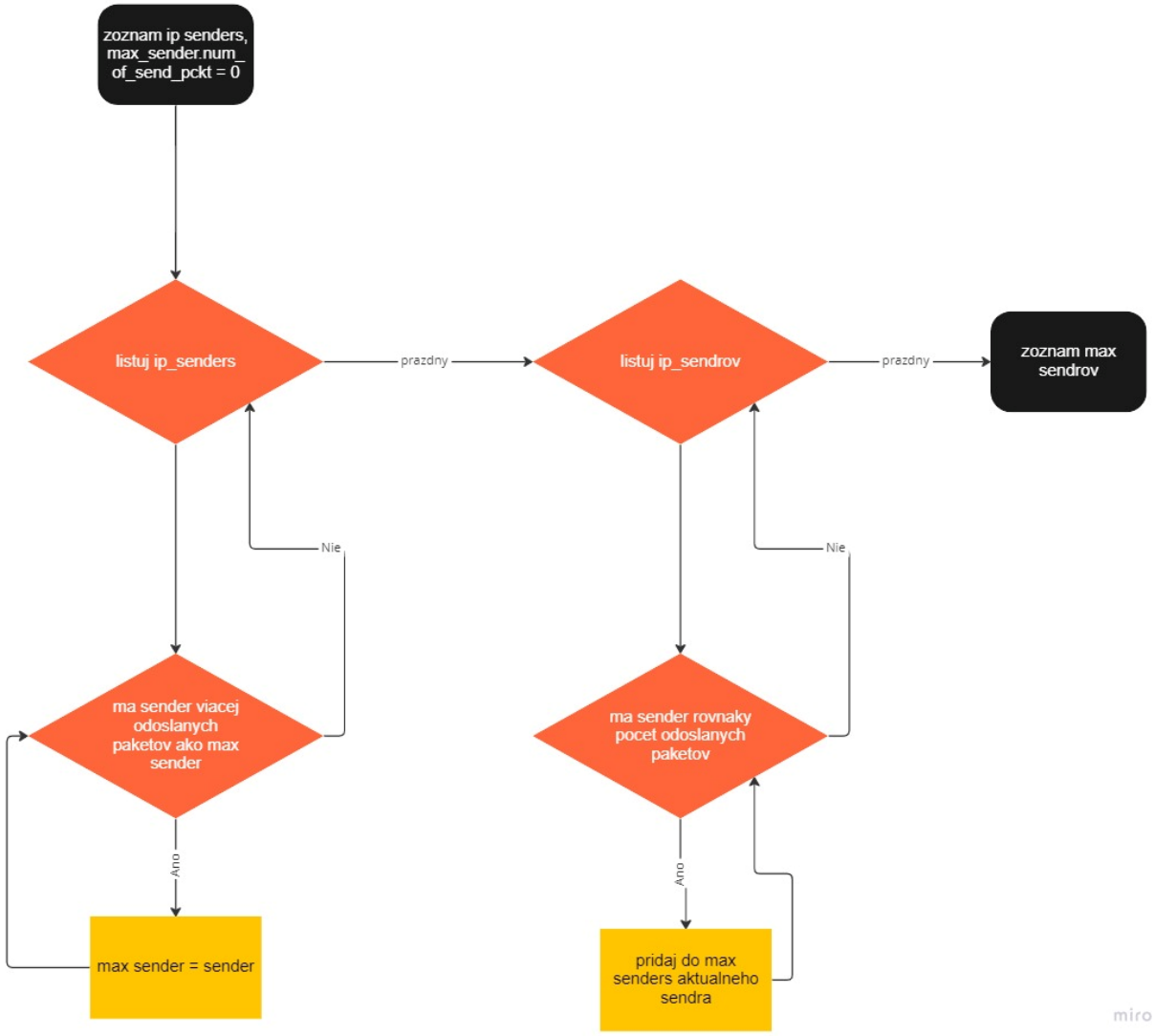
Slovny opis:

Najskor si vytvorim list ip adries ktore odoslali nejaky paket. Pricom kontrolujem aby som si ip adresu nepridal do zoznamu dvakrat nasledne vytvorim pole ip_senders

- objekt sender = {"node": ip, "number_of_sent_packets" : 0}

Potom prejdem pre kazdeho ip_sendra vsetky pakety a ak sa zhoduje ip adresa sendra so zdrojovou ip adresou inkrementujem atr. Number of sent packets o 1. Ked uz mam vytvorene pole sendrov. Prehladam ho a najdem maximalne hodnoty.

Hladanie maxima v zozname sendrov



Hladanie protokolov s komunikaciou so spojenim

Vytvaranie class paketov a pridavanie do listu je stale rovnake. Jediny rozdiel je pridanie nasledovneho ifu do funkcie

```
if "protocol" in frame.__dict__ and frame.protocol == "TCP" and "app_protocol" in
frame.__dict__ and frame.app_protocol == app_protocol:
            frame.set_raw_frame(hex_packet)
            ipv4_tcp_list.append(frame)
```

Dany if zabezpeci ze sa mi do ipv4_tcp_listu pridaju len pakety ktore splnaju filter zadany z konzoly

Vzory otvoreni a zatvoreni

Dany problem som sa rozhodol riesit sposobom ze som si vytvoril listy pre vsetky mozne sposoby otvorenia a zatvorenia komunikacie.

- Jeden sample: 3-way-handshake {point : 0, samples: [00010=syn, 10010=syn+ack, 10000=ack]}
    - point mi ukazuje do pola samples. V poli samples sa nachadzaju jednotlive flagy v takom poradi aby splnali poziadavky na otvorenie komunikacie. Ak pri prehladavani paketov narazim na paket ktory ma setnute rovnake flagy ako openings[0][samples[point]] inkrementujem point o jedna.


```
{
    "openings": [
        {
            "point": 0,
            "samples": ["00010", "10010", "10000"]
        },
        {
            "point": 0,
            "samples": ["00010", "00010", "10000", "10000"]
        }
    ],
    "closings": [
        {
            "point": 0,
            "samples": ["00001", "10000", "00001", "10000"]
        },
        {
            "point": 0,
            "samples": ["00001", "00001", "10000", "10000"]
        },
        {
            "point": 0,
            "samples": ["10001", "10000", "10001", "10000"]
        },
        {
            "point": 0,
            "samples": ["00001", "00100", "10000"]
```

```
        },
        {
            "point": 0,
            "samples": ["00100", "10100"]
        },
        {
            "point": 0,
            "samples": ["00100", "10000"]
        }
    ],}
```

Postup riesenia

Pri hladani komunikacii zacnem tym ze si najskor najdem vsetky regularne otvorenia a zatvorenia. To robim v metode find_openings_endings. Ked najdem vsetky otvorenia a zatvorenia komunikacii nasledne sa pokusim naparovat otvorenia s ukoncenim pricom sa musia rovnat porty a ip adresy. Ak sa mi takyto par podari najst nasledne najdem vsetky pakety ktore si medzi sebou zariadenia vymenili. Takyto par pridam do kompletnych komunikacii. Z tych co mi ostanu vyberiem jednu a oznacim ju za nekompletnu komunikaciu. Nasledne aj pre nu najdem vsetky pakety.

Find_openings_endings

Metoda ma za ulohu najst vsetky otvorenia a zatvorenia komunikacii a vratit ich.

```
def find_openings_endings(list, op_samples, cl_samples):
    openings = []
    endings = []
    tmp_open = {}
    tmp_closed = {}
    #nastavi atributy tmp
    set_tmp(tmp_open, op_samples)
    set_tmp(tmp_closed, cl_samples)

    for idx, frame in enumerate(list):
        #nachadza sa frame.flags v niektorom zo samplov
        if is_in_samples(frame, tmp_open, op_samples, "open"):
            find_com(tmp_open, list, idx + 1 ,op_samples, openings)
            set_tmp(tmp_open, op_samples)
             #resetne pointre pre kazdu vzorku na 0
            reset_samples(op_samples)
        #to iste len kontrolujem pre zatvorenia
        elif is_in_samples(frame, tmp_closed, cl_samples, "close"):
            find_com(tmp_closed, list, idx + 1 ,cl_samples, endings)
            set_tmp(tmp_closed, cl_samples)
            #resetne pointre pre kazdu vzorku na 0
            reset_samples(cl_samples)

    return openings, endings
```

Is_in_samples

Metoda prehladava pole vzoriek otvoreni a ak flagy nejakeho paketu zhoduju s danym samplom update point o jedno a prida paket do zoznamu. Kazde vzorove otvorenie alebo zatvorenie ma svoj vlastny list paketov.

```
def is_in_samples(frame, tmp, samples, type):
    """return True if sample.point pointing on pattern which is equal to
frame.flags"""
    find = False
      #prehlada vsetky vzorove otvorenia alebo zatvorenia a porovna ich s
frame.flags
    for idx, sample in enumerate(samples):
        #vypytam si od kazdej vzorky paterny na ukoncenie/zacatie komunikacie
        patterns = sample["samples"]
        #porovnam pattern s flagmi framu
        if patterns[sample["point"]] == frame.flags:
            #posuniem pointer na dalsiu flagu
            sample["point"] += 1
            tmp["sender_ip"] = frame.src_ip
            tmp["receiver_ip"] = frame.dst_ip
            tmp["sender_port"] = frame.src_port
            tmp["receiver_port"] = frame.dst_port
            tmp["type"] = type
            #pointer ukazuje na rovnake miesto v pamati, malo by stacit dat len
== netreba porovnavat atributy
            tmp[f"packets{idx}"].append(frame)
            find = True

    return find
```

find_com

Metoda sluzi na hladanie paketov otvorenia a zatvorenia.

```
# function which are looking for openings and endings of comunication
def find_com(tmp, tcp_list, l_idx, samples, list):

    for i in range(l_idx, len(tcp_list)):
        frame = tcp_list[i]
        #ma frame rovnake flags ako niektory zo samplov,
        #is complete len skontruluje ci uz sa point == dlzke pola so vzorkami =>
splnilo podmienky
        if is_second_sample(frame, tmp, samples) and is_complete(samples):
            node = cr_node(tmp)
            list.append(node)
            return

def is_second_sample(frame, tmp, sample):
    find = False

    for idx,s in enumerate(sample):
        pattern = s["samples"]
        #ak sa paket,flags zhoduju s vzorkou a sedia nam aj ipcky a porty
posuniem point na dalsiu vzorku a pridam paket ku komunikacii
        if pattern[s["point"]] == frame.flags and are_ports_ips_same(frame, tmp):
            find = True
            s["point"] += 1
            tmp[f"packets{idx}"].append(frame)

    return find
```

Metoda find_complete_uncomplete

Potom ako mam otvorenia a zatvorenia komunikacii pokusim sa naparovat otvorenia a zatvorenia. Ak najdem nejaku dvojicu nasledne najdem pakety ktore si medzi sebou odoslali, vymazem ich z pola openings a endings. Takuto dvojicu pridam do pola complete. Uncomplete komunikaciu ziskam z prvkov listu ktore sa nepodarilo naparovat

```
def find_complete_uncomplete(list, op_samples, cl_samples):
    complete = []
    uncomplete = []
    count = 1
    #endings and openings work just perfect
    openings, endings = find_openings_endings(list, op_samples, cl_samples)

    for open_pckt in openings:
      #pre kazde otvorenie prehladam vsetky zatvorenia
        for end_pckt in endings:
            #ak sa zhoduju porty a ip viem ze sa jedna spravne uzavretie
            if open_pckt["sender_ip"] == end_pckt["sender_ip"] and
open_pckt["receiver_ip"] == end_pckt["receiver_ip"] and
open_pckt["receiver_port"] == end_pckt["receiver_port"] and
open_pckt["sender_port"] == end_pckt["sender_port"]:
                #nastavim potrebne atributy na najdenie komunikacie
                first_packet,last_packet,src_ip, dst_ip, src_port, dst_port =
set_atributes_for_complete(open_pckt, end_pckt)
                #vrati list paketov ktore si medzi sebou zariadenia vymenili
                packets = find_packets_for_com(first_packet, last_packet, src_ip,
dst_ip, src_port, dst_port, list)
                complete.append(cr_yaml_node(open_pckt, packets, count))
                openings.remove(open_pckt)
                endings.remove(end_pckt)
                count += 1

    if len(openings) == 0 and len(endings) == 0:
        return complete, uncomplete
    if len(openings) != 0:
        open_pckt = openings.pop()
        first_packet,src_ip, dst_ip, src_port, dst_port =
set_atributes_for_uncomplete(open_pckt)
        packets = find_packets_for_uncomplete_open(list, first_packet, src_ip,
dst_ip, src_port, dst_port)
```

```
        uncomplete.append(cr_yaml_node(open_pckt, packets, 1))
        return complete, uncomplete
    elif len(endings) != 0:
        end_pckt = endings.pop()
        last_packet,src_ip, dst_ip, src_port, dst_port =
set_atributes_for_uncomplete(end_pckt)
        packets = find_packets_for_uncomplete_end(list, last_packet, src_ip,
dst_ip, src_port, dst_port)
        uncomplete.append(cr_yaml_node(end_pckt, packets, 1))
        return complete, uncomplete
    else:
        return False, False
```

Komunikacia bez spojenia

Flagy pre udp komunikaciu

```
"flags_for_udp": {
        "0002": "WRITE REQUEST",
        "0001": "READ REQUEST",
        "0005": "ERROR CODE",
        "0003": "LAST PACKET",
        "0004": "ACKNOWLEDGMENT"
    },
```

Vytvaranie class paketov a pridavanie do listu je stale rovnake. Jediny rozdiel je pridanie nasledovneho ifu do funkcie

```
 if "protocol" in frame.__dict__ and frame.protocol == "UDP" and "app_protocol"
in frame.__dict__ and frame.app_protocol == "TFTP":
            frame.set_raw_frame(hex_packet)
            ipv4_udp_list_start.append(frame)
```

Tento if zabezpeci filtraciu udp komunikacii.

Postup riesenia

Odfiltrujem si udp komunikacie. Prechadzam pole paketov v udp_liste a ak najdem paket ktoruy ma opcode typu WRITE REQUEST alebo READ REQUEST a destination port je 69 ulozim si tento **paket ako zaciatok komunikacie** a zacnem hladat pre dane ipcky a porty packety ktore patria do komunikacie. Toto vyhladavanie realizujem vo funkcii find_com_udp. Z listu vsetkych paketov si zistim index kde sa nachadza **prvy paket** a zacnem pole prehladavat od daneho indexu. Hladam dokym nenarazim na packet s dst portom 69=zaciatok novej komunikacie. Ako prve sa snazim zistit port z ktoreho server odpoveda. Request pride na port 69 ale response uz z neho nepride. Toto som vyriesil tak ze kontrolujem ip adresy s **prvym packetom** ak sa zhoduju a zhoduje sa aj port z ktoreho prisla komunikacia zistim tak novy port

na ktorom bude prebiehat komunikacia. Nasledne uz len prehladavam pole a pridavam pakety ak sa zhoduje ip a porty.

```
def find_tftp_comunications(udp_list, all_list, udp_dic):
    comunications = []
    #84-88 b = opcode
    tmp = {
        "sender_ip": "x",
        "receiver_ip": "x",
        "sender_port": "x",
        "receiver_port": "x",
        "packets": []
    }

    #prva komunikacia musi prist od sendera na 69 port, potom z opacnej strany z hociakeho portu na senderov port
    #postup: 1. najdi zaciatok komunikacie
    #2. najdi vsetky pakety az dokym nenajdes port 69

    for frame in udp_list:
        frame.set_op_code_udp(udp_dic)
            #najdem request hladam pre neho komunikaciu
        if (frame.opcode == "WRITE REQUEST" or frame.opcode == "READ REQUEST") and frame.dst_port == 69:

            tmp["packets"].append(frame)
            tmp = find_com_udp(frame, tmp, all_list)
            comunications.append(tmp)
            tmp = {
                "packets": []
            }
    return comunications
```

```
def find_com_udp(start, tmp, all_list):
    find_start = False
    tmp["sender_ip"] = start.src_ip
    tmp["receiver_ip"] = start.dst_ip
    tmp["sender_port"] = start.src_port
    idx_l = all_list.index(start) + 1
    #ip adresy sa musia rovnat
    #a frame dst_port == tmp["sender_port"]
    for idx in range(idx_l, len(all_list)):
        pckt = all_list[idx]
        if "protocol" in pckt.__dict__ and pckt.protocol == "UDP":
            #nova komunikacia
            if pckt.dst_port == 69:
                return tmp
            #uz poznam port servera, ak sa zhoduju ipcky a porty pridam do
komunikacie
            if find_start and are_ports_ips_same(pckt, tmp):
                tmp["packets"].append(pckt)
            #nastavenie portu na ktorom bude prebiehat komunikacia
            if are_ip_same(pckt, tmp) and pckt.dst_port == tmp["sender_port"] and
not find_start:
                tmp["receiver_port"] = pckt.src_port
                tmp["packets"].append(pckt)
                find_start = True
    return tmp
```

ARP

Flagy pre ARP protokoly

```
"arp_flags":{
        "0002": "reply",
        "0001": "request"
    },
```

Vytvaranie class paketov a pridavanie do listu je stale rovnake. Jediny rozdiel je pridanie nasledovneho ifu do funkcie

```
if "ether_type" in frame.__dict__ and frame.ether_type == "ARP":
            arp_list.append(frame)
            frame.set_raw_frame(hex_packet)
            frame.set_op_code_arp(arp_dic)
```

Zabezpeci aby som v liste mal len arp komunikaciu

Postup riesenia

Ako prve si najdem v udo_liste vsetky requesty a pridam ich do listu requests. Nasledne prechadzam polom requests. Ako prve zistim index kde v poli sa nachadza request(setri cas). A zacnem prechadzat list vsetkych packetov mozu nastat dve moznosti

1. Packet ma rovnake ipcky a opcode je request, to znamena ze je sucastou komunikacie tak ho pridam do zoznamu paketov
2. Packet ma rovnake ipcky ako request a opcode je reply to znamena ze som nasiel completnu komunikaciu

Find_com_arp

```
def find_com_arp(list, arp_dic):
    # musim najst request a vyparsovat z nej data

    list_requests = []
    complete_com = []
    #find requests
    for idx, pckt in enumerate(list):
        if pckt.opcode == "request":
            list_requests.append(pckt)
    count = 1
    for request in list_requests:
        idx = list.index(request) + 1
        packets = []
        packets.append(request)
        for idx_l in range(idx, len(list)):
            pckt = list[idx_l]
             #rovnaka request
            if "opcode" in pckt.__dict__ and pckt.opcode == "request":
                 if pckt.src_ip == request.dst_ip and pckt.dst_ip ==
request.src_ip:
                     packets.append(pckt)
             #nasiel som reply na request
            if "opcode" in pckt.__dict__ and pckt.opcode == "reply":
                if pckt.src_ip == request.dst_ip and pckt.dst_ip ==
request.src_ip:
                     packets.append(pckt)

                     complete_com.append({
                         "number_comm":count,
                         "src_comm": request.src_ip,
                         "dst_comm": request.dst_ip,
                         "packets": packets
                     })
                     count += 1
                     break

    return complete_com
```

Pouzivatelske rozhranie

Main.py

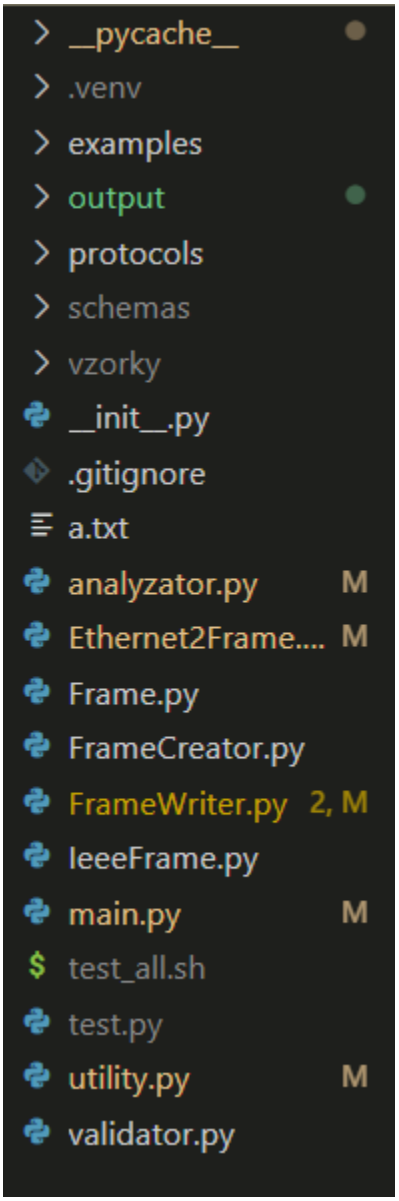
Pouzivatelske rozhranie je jednoducha konzolova aplikacia. Program si vypyta meno suboru, ak pouzivatel zada neexistujuci subor program si vypyta meno suboru znova. Ak program uspesne dokonci pracu vypise hlasku o stave. Pouzivatel moze ukoncit program prikazom exit

Analyzator.py

```
(.venv) ~/school/zs_2022/pks/zad1/PKS_packet_analyzer/src (master X)* ▷ python analyzator.py
write name of trace: trace-26.pcap
write a protocol: a
Wrong protocol
try again: a
try again: a
try again: a
try again: a
try again: a
try again: a
try again: a
try again: ARP
run succesfull
write name of trace:
```

Pouzivatelske rozhranie je jednoducha konzolova aplikacia. Ako prve si program vypyta meno suboru z ktoreho chce pouzivatel cerpat data o paketoch Ak zada neexistujuci subor program ho informuje o chybe a vypyta si meno suboru znova. Nasledne zada typ filtra ktory chce aplikovat. Ak zada neexistujuci protokol program vypyse chybovu hlasku a poziada pouzivatela o znovu zadanie filtra. Ak vsetko zbehne v poriadku program vypise hlasku a informuje pouzivatela o uspesnom skonceni. Program sa vypne prikazom exit

Suborova struktura



- Output: ukladam vysledne yaml file-s
- Protocols: obsahuje key:value pre jednotlive protokoly

Pouzite kniznice

1. Kniznica json:
    a. Pouzivam na zapisanie hodnot pre jednotlive protokoly v ext. suboroch, pomocou tejto kniznice nacitavam data zo suboru
2. Ruamel.yaml
    a. Pouzivam na zapis slovnikov do yaml formatu
3. Scapy
    a. Na citanie .pcap suborov
4. Binascii
    a. Prevedenie raw packetu do hexa tvaru

Zaver

Zo zadania som sa naucil rozpoznavat pakety, lepsie som pochopil komunikaciu na sieti a naucil som sa nieco nove o protokoloch, ich strukture a vyznamu. Zadanie som vypracoval samostatne najlepsie ako som vedel. Keby mam viacej casu urcite by som doimplementoval ICMP komunikaciu nakolko to je jedina cast zo zadania ktoru som nestihol kompletne dokoncit. Program by som vylepsil refaktorizaciou existujuceho kodu.