Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Počítačové a komunikačné siete

Zadanie 1 - Analyzátor sieťovej komunikácie

Jakub Hrnčár
AIS ID: 110 800, ak. rok 2021/2022

1. Úvod do zadania

Úlohou tohto zadania bolo vytvoriť program podobný nástroju Wireshark. Jeho podstatou je zachytávanie sieťovej komunikácie na sieťovej karte v PC, s rozdielom, že náš program len číta zachytenú komunikáciu z pcap súborov.

V zadaní figuruje 8 bodov, z ktorých 4 udávajú programovacie úlohy, ďalšie 4 udávajú organizačné pokyny. Medzi programovacie úlohy patrí:

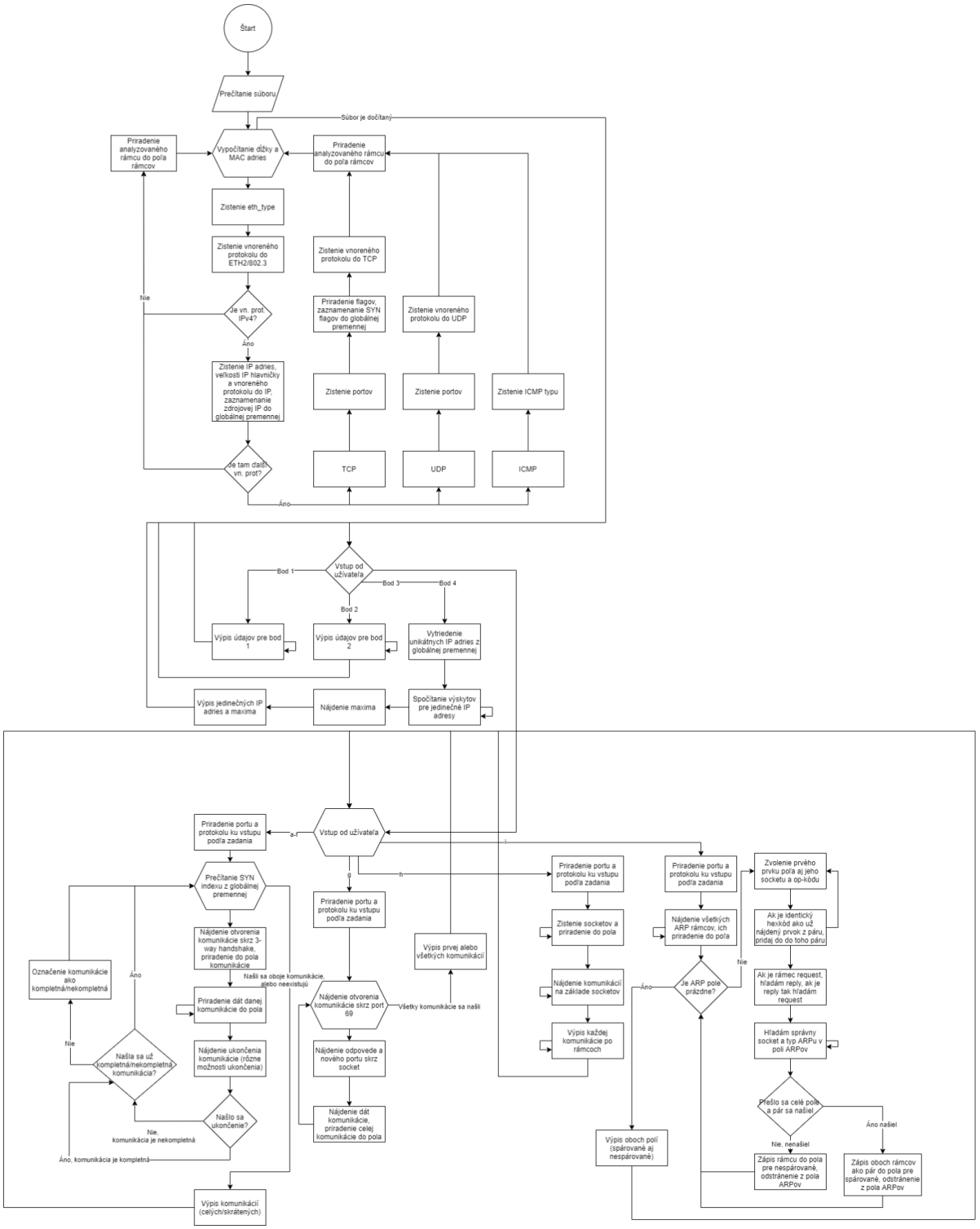
- výpis všetkých rámcov a základné info o nich (MAC adresa, dĺžka, protokol)
- výpis vnorených protokolov a info o nich (IP adresy, porty)
- výpis štatistiky odoslaných rámcov z IP adresy; ktorá adresa odoslala najviac rámcov
- konkrétne analyzovanie komunikácií pre špecifické protokoly

Zadanie musí byť pomerne robustné a užívateľsky prívetivé, musí mať možnosť voľby medzi jednotlivými časťami programu.

Zadanie som vytvoril v programovacom jazyku Python 3.9, ako implementačné prostredie som zvolil PyCharm, čisto kvôli skusenostiam z minulosti. Program sa spúšťa po nainštalovaní knižnice scapy a pridaní konfiguračného súboru a pcap súborov do spoločného adresára s týmto programom.

2. Blokový diagram celého programu

V prílohe sa nachádza tento diagram v plných rozmeroch, tu je iba zmenšený. Je tu opísaný celý program, jeho základný algoritmus. Podrobnejším detailom sa budem venovať v ďalšej sekcií.



3. Stručný opis algoritmov

V zadaní sa nachádza veľké množstvo algoritmov, boli vytvárané procedurálne. Postupne budem prechádzať jednotlivými časťami programu, tak ako ich užívateľ bude stretávať:

a. Trieda Frame

V prvom rade je veľmi dôležité vysvetliť, ako sa ukladá jeden rámec. Vytvoril som triedu Frame, ktorá nemá žiadne metódy a správa sa len ako štruktúra. Jej atribúty sú takmer všetky podrobnosti o jednom rámci (MAC a IP adresy, TCP flagy a pod.). Ich prednastavená hodnota je None a postupne, ak daný atribút má relevanciu v danom rámci, sa menia a priraďuje sa im hodnota. Tieto rámce sú následne ukladané do premennej *frames*, ktorá sa následne posiela do funkcií.

b. Konfiguračný súbor config.cfg

Program používa externý súbor config.cfg, bez ktorého sa nedá program riadne spustiť. Jeho formátovanie a čítanie je podmienené natívnou knižnicou configparser, ktorá udáva nasledovné formátovanie:
[Názov_sekcie]
kľúč = hodnota

Sekcia Ethertypes v mojom súbore vyzerá nasledovne:

```
[EtherTypes]
512 = XEROX PUP
513 = PUP Addr Trans
2048 = IPv4
2049 = X.75 Internet
2053 = X.25 Level 3
2054 = ARP
8192 = CDP
8196 = DTP
32772 = Cronus Direct
32821 = Reverse ARP
32923 = Appletalk
33011 = AppleTalk AARP (Kinetics)
33024 = IEEE 802.1Q VLAN-tagged frames
33079 = Novell IPX
34525 = IPV6
34827 = PPP
34887 = MPLS
34888 = MPLS with upstream-assigned label
34915 = PPPoE Discovery Stage
34916 = PPPOE Session Stage
35020 = IEEE Std 802.1AB - Link Layer Discovery Protocol (LLDP)
36864 = Loopback
```

Dáta zo súboru ziskavam pomocou metódy *.get(sekcia :str, kľúč :str)*, ktorá vráti hodnotu v stringu. V prípade, že nenájde daný kľúč, vráti *NoOptionError*.

c. Menu

Po spustení programu sa užívateľ dostane do menu, ktoré si od neho vypýta názov pcap súboru, ktorý bude analyzovať. Po chvíli, kedy program automaticky analyzuje každý rámec, sa objaví možnosť zvoliť si z bodov zadania 1 - 4, vypísať len konkrétny rámec alebo ukončiť program.

d. Bod 1

Zavolanie bodu 1 okamžite začne vypisovať základné informácie o každom rámci. Vzorový výpis:

1.: 802.3 LLC:
Length API: 104 Length medium: 108
Destination: 00:04:76:13:97:df Source: 00:00:c0:d7:80:c2
00 04 76 13 97 df 00 00 c0 d7 80 c2 00 5a e0 e0 03 ff ff 00 56 00 04 00 00 00 98 00 04 76 13 97
df 04 55 00 00 00 98 00 00 c0 d7 80 c2 04 55 41 06 25 40 ff ff 00 00 26 00 00 00 26 00 00 00 01
00 41 41 2d 44 44 34 50 5a 32 56 31 50 47 35 56 00 45 4e 49 47 4d 41 20 20 20 20 20 20 20 20 20
20 a9 05 15 00 09 00 00

2.: 802.3 LLC:
Length API: 104 Length medium: 108
Destination: 00:00:c0:d7:80:c2 Source: 00:04:76:13:97:df
00 00 c0 d7 80 c2 00 04 76 13 97 df 00 5a e0 e0 03 ff ff 00 56 00 04 00 00 00 98 00 00 c0 d7 80
c2 04 55 00 00 00 98 00 04 76 13 97 df 04 55 81 06 10 60 25 40 00 00 26 00 00 00 26 00 01 00 06
00 45 4e 49 47 4d 41 20 20 20 20 20 20 20 20 20 20 41 41 2d 44 44 34 50 5a 32 56 31 50 47 35 56
00 a9 05 15 00 09 00 00

Vo výpise vidíme hexadecimálny zápis obsahu celého rámca, z ktorého sú čerpané všetky informácie pre celý program. MAC adresy sú ukladané do stringu, napríklad *frame.src_mac = "0000c0d780c2"*, takže až vo výpise sú naformátované do správneho tvaru, aby sa predošlo plýtvaniu pamäte.

Algoritmus na analyzovanie rámcu a získanie potrebných informácií pre tento výpis pozostáva z viacerých častí a je formátovaný do viacerých čiastkových funkcií. V tomto bode sa zisťuje iba dĺžka, protokol na linkovej vrstve a MAC adresy. Protokol je zistený na základe dĺžky 3. políčka v prvej hlavičke, či je číslo ktoré obsahuje

väčšie ako 1500. Ak je protokol 802.3, tak sa pozerá, ma 4. políčko a na základe jeho hodnoty určujem vnorený protokol. Dĺžku celého rámca v API zisťujem skrz funkciu *len()* a následne dĺžku po médiu počítam z nej, pripočítam k nej 4 alebo ju zaokrúhlim na 64, ak je API dĺžka menšia ako 60. MAC adresy vyberám z prvého a druhého políčka a formátujem ich.

e. Bod 2

Zavolanie bodu 2 okamžite začne vypisovať informácie o vnorenom protokole v každom rámci, kde sa nachádza. Vzorový výpis:

5.: IPv4
Zdrojová IP: 192.168.1.102
Cieľová IP: 192.168.1.104
UDP
Zdrojový port: 4306
Cieľový port: 161
snmp

6.: IPv4
Zdrojová IP: 192.168.1.104
Cieľová IP: 192.168.1.102
UDP
Zdrojový port: 161
Cieľový port: 4306

7.: IPv4
Zdrojová IP: 192.168.1.102
Cieľová IP: 128.119.245.12
TCP
Zdrojový port: 4307
Cieľový port: 80
http

8.: IPv4
Zdrojová IP: 128.119.245.12
Cieľová IP: 192.168.1.102
TCP
Zdrojový port: 80
Cieľový port: 4307

Rovnako ako aj v bode 1, všetky informácie boli najprv analyzované, po zavolaní bodu 2 sa len vypíše výpis z informácií o vnorených protokoloch, žiadne dopočítavanie sa už nedeje.

Algoritmus na analyzovanie rámcu a získanie potrebných informácií pre tento výpis sa zavolá bezprostredne po volaní predošlého bodu. V tomto bode sa môžu nachádzať rôzne vnorené protokoly, avšak na ich analyzovanie sa dá uplatniť jeden algoritmus. Začína sa na sieťovej vrstve, kde sa môžu nachádzať tieto protokoly:

EtherType values
google for "IANA Ether Types" for up-to-date list

Hex	Dec	Description
0200	0512	XEROX PUP
0201	0513	PUP Addr Trans
0800	2048	Internet IP (IPv4)
0801	2049	X.75 Internet
0805	2053	X.25 Level 3
0806	2054	ARP (Address Resolution Protocol)
8035	32821	Reverse ARP
809B	32923	Appletalk
80F3	33011	AppleTalk AARP (Kinetics)
8100	33024	IEEE 802.1Q VLAN-tagged frames
8137	33079	Novell IPX
86DD	34525	IPv6
880B	34827	PPP
8847	34887	MPLS
8848	34888	MPLS with upstream-assigned label
8863	34915	PPPoE Discovery Stage
8864	34916	PPPoE Session Stage

802.2 LLC Service Access Points (SAPs)
IEEE SAPs

Hex	Dec	Function
00	0	Null SAP
02	2	LLC Sublayer Management / Individual
03	3	LLC Sublayer Management / Group
06	6	IP (DoD Internet Protocol)
0E	14	PROWAY (IEC 955) Network Management, Maintenance and Installation
42	66	BPDU (Bridge PDU / 802.1 Spanning Tree)
4E	78	MMS (Manufacturing Message Service) EIA-RS 511
5E	94	ISI IP
7E	126	X.25 PLP (ISO 8208)
8E	142	PROWAY (IEC 955) Active Station List Maintenance
AA	170	SNAP (Sub-Network Access Protocol / non-IEEE SAPs)
E0	224	IPX (Novell NetWare)
F4	244	LAN Management
FE	254	ISO Network Layer Protocols
FF	255	Global DSAP

Ak je na spojovej vrstve Ethernet 2, pozerám sa na *EtherType values* a 3. políčko, ak je tam 802.3 LLC, tak na 4. políčko a tabuľku *SAPs*. Hodnoty tabuliek sú uložené v config.cfg. Iba v prípade existencie IPv4 protokolu na tejto vrstve sa pozerám ďalej a priradzujem IP adresy do objektu rámca. Môže sa tu nachádzať aj protokol ARP, ale tento protokol analyzujem samostatne neskôr.

```
def nested_protocols(f):
    if f.eth_type == "Ethernet II":
        f.nested_mac = nested_mac(int(f.hex_value[24:28], 16), 'EtherTypes')
        if int(f.hex_value[24:28], 16) == 2048:
            f.src_ip, f.dest_ip, f.nested_ip, f.size_ip = ip(f.hex_value[28::])
            if f.nested_ip is not None:
                nested_internet(f)

    elif "802.3 LLC" == f.eth_type:
        f.nested_mac = nested_mac(int(f.hex_value[28:30], 16), 'SAPs')

    elif "802.3 - raw" == f.eth_type:
        f.nested_mac = "IPX"

    elif "802.3 LLC + SNAP" == f.eth_type:
        f.nested_mac = nested_mac(int(f.hex_value[40:44], 16), 'EtherTypes')
```

Nasleduje transportná vrstva, kde sa môžu nachádzať tieto protokoly:

Protocol					
1	ICMP	17	UDP	57	SKIP
2	IGMP	47	GRE	88	EIGRP
6	TCP	50	ESP	89	OSPF
9	IGRP	51	AH	115	L2TP

Protokoly ICMP, TCP a UDP ďalej analyzujem, ostatné len vložím do objektu rámca. Vyššie vymenované protokoly majú aj svoje ďalšie údaje, ktorými sú:

ICMP - typ (viď. bod 4 h))

UDP - porty

TCP - porty a flagy (viď. bod 4 a-f))

```
def nested_internet(f):
    try:
        if f.nested_ip == "TCP":
            f.src_port = int(f.hex_value[(28 + (f.size_ip * 2)): (32 + (f.size_ip * 2))],
16)
            f.dest_port = int(f.hex_value[(32 + (f.size_ip * 2)): (36 + (f.size_ip *
2))], 16)
            tcp_analyze(f)
            f.nested_port = config.get('TCP_ports', str(f.dest_port))
        elif f.nested_ip == "UDP":
            f.src_port = int(f.hex_value[(28 + (f.size_ip * 2)): (32 + (f.size_ip * 2))],
16)
            f.dest_port = int(f.hex_value[(32 + (f.size_ip * 2)): (36 + (f.size_ip *
2))], 16)
            f.nested_port = config.get('UDP_ports', str(f.dest_port))
        elif f.nested_ip == "ICMP":
            icmp_analyze(f)
    except configparser.NoOptionError:
        f.nested_port = None
```

Ako posledné ma v tomto bode zaujímajú protokoly na prvej, aplikačnej vrstve:

UDP Header Information

Common UDP Well-Known Server Ports			
7	echo	138	netbios-dgm
19	chargen	161	snmp
37	time	162	snmp-trap
53	domain	500	isakmp
67	bootps (DHCP)	514	syslog
68	bootpc (DHCP)	520	rip
69	tftp	33434	traceroute
137	netbios-ns		

TCP Header Contents

Common TCP Well-Known Server Ports

7	echo	110	pop3
19	chargen	111	sunrpc
20	ftp-data	119	nntp
21	ftp-control	139	netbios-ssn
22	ssh	143	imap
23	telnet	179	bgp
25	smtp	389	ldap
53	domain	443	https (ssl)
79	finger	445	microsoft-ds
80	http	1080	socks

Tieto protokoly majú aj svoje hlavičky a údaje, avšak s nimi už nepracujem, ich názvy len vložím do objektu rámca.

f. Bod 3

Bod 3 vypisuje štatistickú informáciu o IP adresách odosielajúcich uzlov a IP adresu, ktorá poslala najviac paketov. Vzorový výpis vyzerá nasledovne:

IP adresy vysielajúcich uzlov:
128.238.38.2
128.238.38.160
128.238.29.22
128.238.38.194
Adresa uzla s najväčším počtom odoslaných paketov:
128.238.38.2 3

Zdrojový kód tohto bodu je:

```
def three():
    unique, count = [], []

    for ip_ in ips:
        if ip_ not in unique:
            unique.append(ip_)
    count = [0] * len(unique)
    for u in range(len(unique)):
        for ip_old in ips:
            if unique[u] == ip_old:
                count[u] += 1
    file.write("IP adresy vysielajúcich uzlov:")
    for u in unique:
        file.write("\n" + u)
    maximum = max(count)

    file.write("\nAdresa uzla s najväčším počtom odoslaných
paketov:\n" + unique[count.index(maximum)] + "\t" + str(maximum))
```

ips je globálna premenná, v ktorej sú uložené všetky zdrojové IP adresy, ukladám ich tam počas analyzovania vnorených protokolov (viď. bod 2). Tento bod je vykonaný na základe redukovania pola na unikátne prvky a následné počítanie týchto prvkov a nájdenie maxima.

g. Bod 4

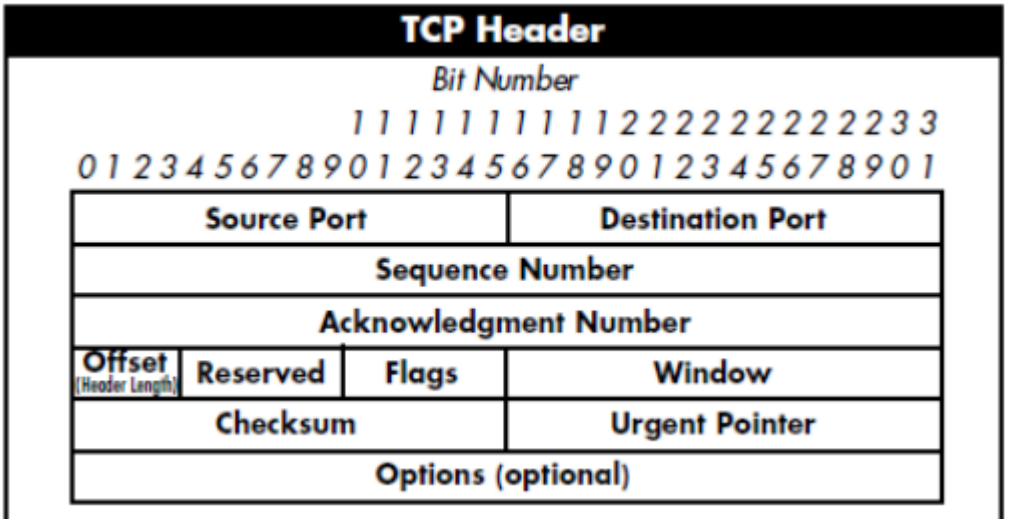
Bod 4 má veľa pod-bodov, ktoré sú veľmi odlišné a preto sa im budem venovať konkrétne vo vlastných bodoch:

- **TCP komunikácie**

Analyzovanie tejto komunikácie delím do piatich štádií. Táto komunikácia musí začínať 3-way handshake-om, ktorý vyzerá nasledovne:

SYN
SYN ACK
ACK

Pokiaľ nezačína týmito konkrétnymi flagmi, nemá zmysel sa ňou zaoberať. Preto som vytvoril globálnu premennú *syns*, do ktorej sú ukladané indexy rámcov, ktoré majú priradený (iba) SYN flag. Flagy analyzujem prevodom na binárny kód.




```
for x in syns:
   if frms[x].dest_port == port or frms[x].src_port == port:
       syn = frms[x]

       for u in range(x, len(frms)):
           if (syn.src_ip == frms[u].dest_ip) and (syn.dest_ip == frms[u].src_ip) and (
                   syn.src_port == frms[u].dest_port) and (syn.dest_port ==
frms[u].src_port):
               if (frms[u].syn_flag == True) and (frms[u].ack_flag == True):
                   sync_ack = frms[u]
                   break
       for u in range(sync_ack.index + 1, len(frms)):
           if (syn.src_ip == frms[u].src_ip) and (syn.dest_ip == frms[u].dest_ip) and (
                   syn.src_port == frms[u].src_port) and (syn.dest_port ==
frms[u].dest_port):
               if frms[u].ack_flag:
                   ack = frms[u]
                   break
```

Následne hľadám tento 3-way handshake, pričom používam podmienky striedania IP adries a portov.

Podobný algoritmus je aj pri čítaní komunikácie. Zvolím si socket, ktorým je pole IP adries a portov, ktorými sa komunikácia otvorila. Prechádzam celým súborom a hľadám všetky rámce, ktoré vyhovujú socket-u, nezastavím sa ani pri ukončení komunikácie, pretože po ukončení ešte môžu prejsť niektoré rámce, o ktoré by som prišiel.

Na identifikáciu ukončenej a neukončenej komunikácie mám algoritmus, ktorý používa princíp stavového automatu. Za validné ukončenie som zvolil:

3-way handshake:

FIN
FIN
ACK

4-way handshake:

FIN
ACK
FIN
ACK

Reset:

reset môže prísť kedykoľvek

Komunikáciu považujem za ukončenú, ak v nej nachádza istá forma jedného z ukončení. Naopak, neukončená je vtedy, keď sa nenachádza - toto bolo ťažšie na opatrenie, keďže je tu veľa možností. Vyriešil som to tak, že ak sa nájde nedokončené ukončenie, tak sa komunikácia automaticky zvolí za neukončenú. Ďalej, ak sa prečíta celá komunikácia a ešťe nebola označená za neukončenú a ešte som žiadnu ukončenú nenašiel, tka ju označím.

Následne iba vypíšem obsah komunikácií.

- **TFTP komunikácie**

TFTP komunikácia je veľmi špecifická, pretože nemá konzistentne daný port pre celú komunikáciu. Princíp jej analyzovania spočíva v nájdení jej začiatku - port 69, nájdení odpovede na tento začiatok - sedí socket (IP adresy a port), pridanie nového portu do socket-u a hľadanie rámcov, ktoré patria do komunikácie na základe socket-u.

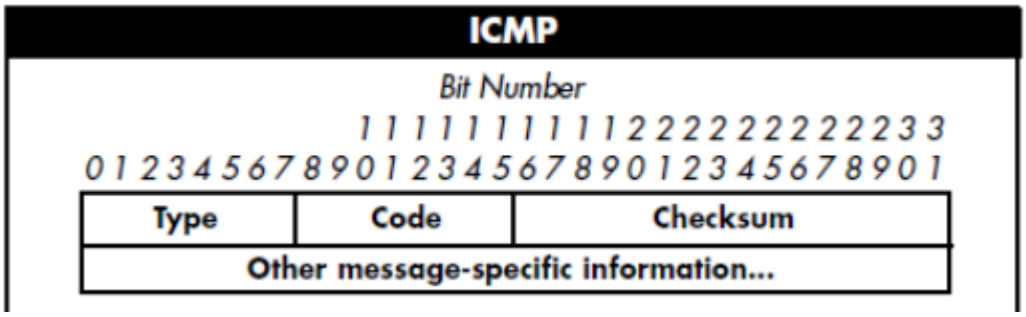
```
for fram in frams:
   if fram.dest_port == 69:
       start_ = fram

       if start_ is not None:
           for u in range(start_.index + 1, len(frams)):
               if (frams[u].src_ip == start_.dest_ip) and (frams[u].dest_ip ==
start_.src_ip) and (
                       frams[u].dest_port == start_.src_port):
                   response = frams[u]
                   break
```

Na jej ukončenie sa dajú využiť rôzne metódy, avšak ja som zvolil metódu istoty a najmenšieho odboru, teda ćítam vždy celý súbor a triedim komunikácie.

Pri výpise jednoducho vypisujem buď prvú, alebo všetky komunikácie, záleží na vstupe od užívateľa.

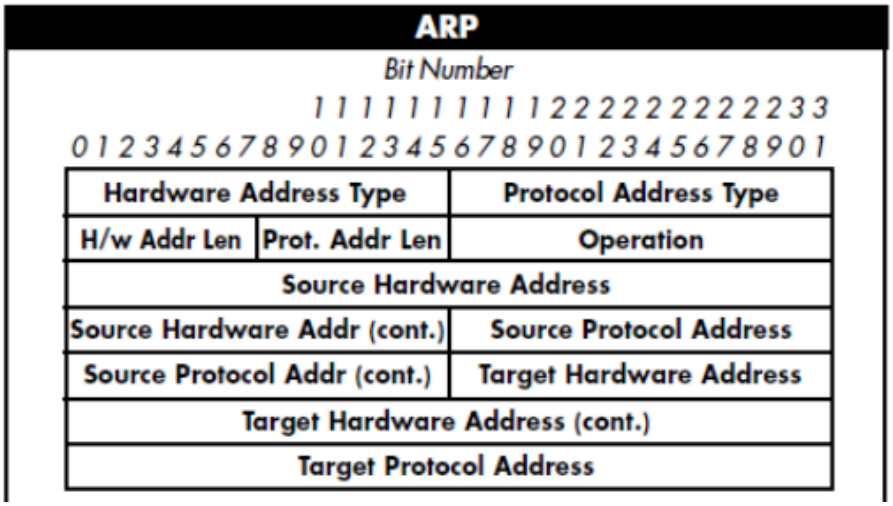
- **ICMP komunikácie**

Tento typ komunikácie nemá nutne žiadny jasný začiatok a koniec, preto na identifikáciu komunikácie sa používa len *socket*, ktorý najprv analyzujem a všetky druhy *socket*-u pre ICMP komunikácie najprv vložím do poľa pre *socket*-y. Následne si cyklom v cykle prechádzam poľom *socket*-ov a poľom rámcov a triedim rámce do komunikácií. Sledujem aj ICMP kód a vypisujem ho; kódy sú uložené v config.cfg




```
for frm in frms:
   if frm.nested_ip == "ICMP":
       if [frm.src_ip, frm.dest_ip] not in sockets and [frm.dest_ip, frm.src_ip] not in
sockets:
           sockets.append([frm.src_ip, frm.dest_ip])

for socket in sockets:
   data_ = []
   for frm in frms:
       if frm.nested_ip == "ICMP":
           if frm.src_ip in socket and frm.dest_ip in socket:
               data_.append(frm)
   icmp.append(data_)
```

- **ARP páry**



Ako prvú akciu vykonám triedenie všetkých ARP rámcov, aby som nemusel neustále prechádzať celým súborom.

Tento protokol má vlastnú hlavičku, ktorá nahrádza IP hlavičku a je od nej veľmi odlišná, preto som zvolil postup, že tieto rámce sa analyzujú až po zavolaní ARP funckie. V časti *operation* sa nachádza kód, ktorý určuje, či ide o požiadavku alebo o odpoveď, na základe ktorej sa vytvárajú páry. IP adresy sú uložené v *protocol adress*, odkiaľ ich prečítam a vytvorím z nich *socket.*

Prechádzam súbor, ak nájdem ARP, analyzujem ho a hľadám k nemu pár, ktorý má opačný kód a vyhovujúci *socket*. Ak som skontroloval všetky ARP rámce a nenašiel som pár, označím rámec ako nepárový a pokračujem. Ak sa pár nájde celý pár pridám do poľa pre páry.

Nakoniec párové a aj nepárové pole vypisujem.

4. Záver

Zadanie som vypracoval samostatne, na základe poznatkov na internete a prednáške, cvičeniach. Snažil som sa dôsledne sledovať zadanie a nepridávať veci navyše. Kde sa dalo, tak tam sa snažil pracovať čo najefektívnejšie, aby nevznikli vysoké hardvérové a pamäťové požiadavky. Zadanie bolo vypracované na Windows 10, s procesorom Intel i3-10100f (4 jadrá, 8 vlákien) a 16 GB DDR4 RAM pamäte.