Slovenská Technická Univerzita

Fakulta informatiky a informačných technológií

Ilkovičova 3, 842 19 Bratislava 4

Richard Szarka

Analyzátor sieťovej komunikácie

Zadanie č. 1

Prednášajúci: prof. Ing. Ivan Kotuliak, PhD.

Cvičiaci: Ing. Kristián Košťál, PhD.

Cvičenie: Štvrtok 10:00

1 Zadanie úlohy:

Navrhnite a implementujte programový analyzátor Ethernet siete, ktorý analyzuje komunikácie v sieti zaznamenané v .pcap súbore a poskytuje nasledujúce informácie o komunikáciách.

Vypracované zadanie musí spĺňať nasledujúce body:

1) Výpis všetkých rámcov v hexadecimálnom tvare postupne tak, ako boli zaznamenané v súbore.

Pre každý rámec uveďte:

A. Poradové číslo rámca v analyzovanom súbore.
B. Dĺžku rámca v bajtoch poskytnutú pcap API, ako aj dĺžku tohto rámca prenášaného po médiu.
C. Typ rámca – Ethernet II, IEEE 802.3 (IEEE 802.3 s LLC, IEEE 802.3 s LLC a SNAP, IEEE 802.3 – Raw).
D. Zdrojovú a cieľovú fyzickú (MAC) adresu uzlov, medzi ktorými je rámec prenášaný.

Vo výpise jednotlivé bajty rámca usporiadajte po 16 alebo 32 v jednom riadku. Pre prehľadnosť výpisu je vhodné použiť neproporcionálny (monospace) font.

2) Pre rámce typu Ethernet II a IEEE 802.3 vypíšte vnorený protokol. Študent musí vedieť vysvetliť, aké informácie sú uvedené v jednotlivých rámcoch Ethernet II, t.j. vnáranie protokolov ako aj ozrejmiť dĺžky týchto rámcov.

3) Analýzu cez vrstvy vykonajte pre rámce Ethernet II a protokoly rodiny TCP/IPv4:

Na konci výpisu z bodu 1) uveďte pre IPv4 pakety:

A. Zoznam IP adries všetkých odosielajúcich uzlov,
B. IP adresu uzla, ktorý sumárne odoslal (bez ohľadu na prijímateľa) najväčší počet paketov a koľko paketov odoslal (berte do úvahy iba IPv4 pakety).

IP adresy a počet odoslaných / prijatých paketov sa musia zhodovať s IP adresami vo výpise

Wireshark -> Statistics -> IPv4 Statistics -> Source and Destination Addresses.

4) V danom súbore analyzujte komunikácie pre zadané protokoly:

A. HTTP
B. HTTPS
C. TELNET
D. SSH
E. FTP riadiace
F. FTP dátové
G. TFTP, uveďte všetky rámce komunikácie, nielen prvý rámec na UDP port 69
H. ICMP, uveďte aj typ ICMP správy (pole Type v hlavičke ICMP), napr. Echo request, Echo reply, Time exceeded, a pod.
I. i) Všetky ARP dvojice (request – reply), uveďte aj IP adresu, ku ktorej sa hľadá MAC (fyzická) adresa a pri ARP-Reply uveďte konkrétny pár - IP adresa a nájdená MAC adresa. V prípade, že bolo poslaných viacero rámcov ARP-Request na rovnakú IP adresu, vypíšte všetky. Ak sú v súbore rámce ARP-Request bez korešpondujúceho ARP-Reply (alebo naopak ARP-Reply bez ARP-Request), vypíšte ich samostatne.

Vo všetkých výpisoch treba uviesť aj IP adresy a pri transportných protokoloch TCP a UDP aj porty komunikujúcich uzlov. V prípadoch komunikácií so spojením vypíšte iba jednu kompletnú komunikáciu - obsahuje otvorenie (SYN) a ukončenie (FIN na oboch stranách alebo ukončenie FIN a RST alebo ukončenie iba s RST) spojenia a aj prvú nekompletnú komunikáciu, ktorá obsahuje iba otvorenie spojenia.

Pri výpisoch vyznačte, ktorá komunikácia je kompletná. Ak počet rámcov komunikácie niektorého z protokolov z bodu 4 je väčší ako 20, vypíšte iba 10 prvých a 10 posledných rámcov tejto komunikácie. (Pozor: toto sa nevzťahuje na bod 1, program musí byť schopný vypísať všetky rámce zo súboru podľa bodu 1.) Pri všetkých výpisoch musí byť poradové číslo rámca zhodné s číslom rámca v analyzovanom súbore.

5) Program musí byť organizovaný tak, aby čísla protokolov v rámci Ethernet II (pole Ethertype), IEEE 802.3 (polia DSAP a SSAP), v IP pakete (pole Protocol), ako aj čísla portov v transportných protokoloch boli programom načítané z jedného alebo viacerých externých textových súborov. Pre známe protokoly a porty (minimálne protokoly v bodoch 1) a 4) budú uvedené aj ich názvy.

Program bude schopný uviesť k rámcu názov vnoreného protokolu po doplnení názvu k číslu protokolu, resp. portu do externého súboru. Za externý súbor sa nepovažuje súbor knižnice, ktorá je vložená do programu.

6) V procese analýzy rámcov pri identifikovaní jednotlivých polí rámca ako aj polí hlavičiek vnorených protokolov nie je povolené použiť funkcie poskytované použitým programovacím jazykom alebo knižnicou. Celý rámec je potrebné spracovať postupne po bajtoch.

7) Program musí byť organizovaný tak, aby bolo možné jednoducho rozširovať jeho funkčnosť výpisu rámcov pri doimplementovaní jednoduchej funkčnosti na cvičení.

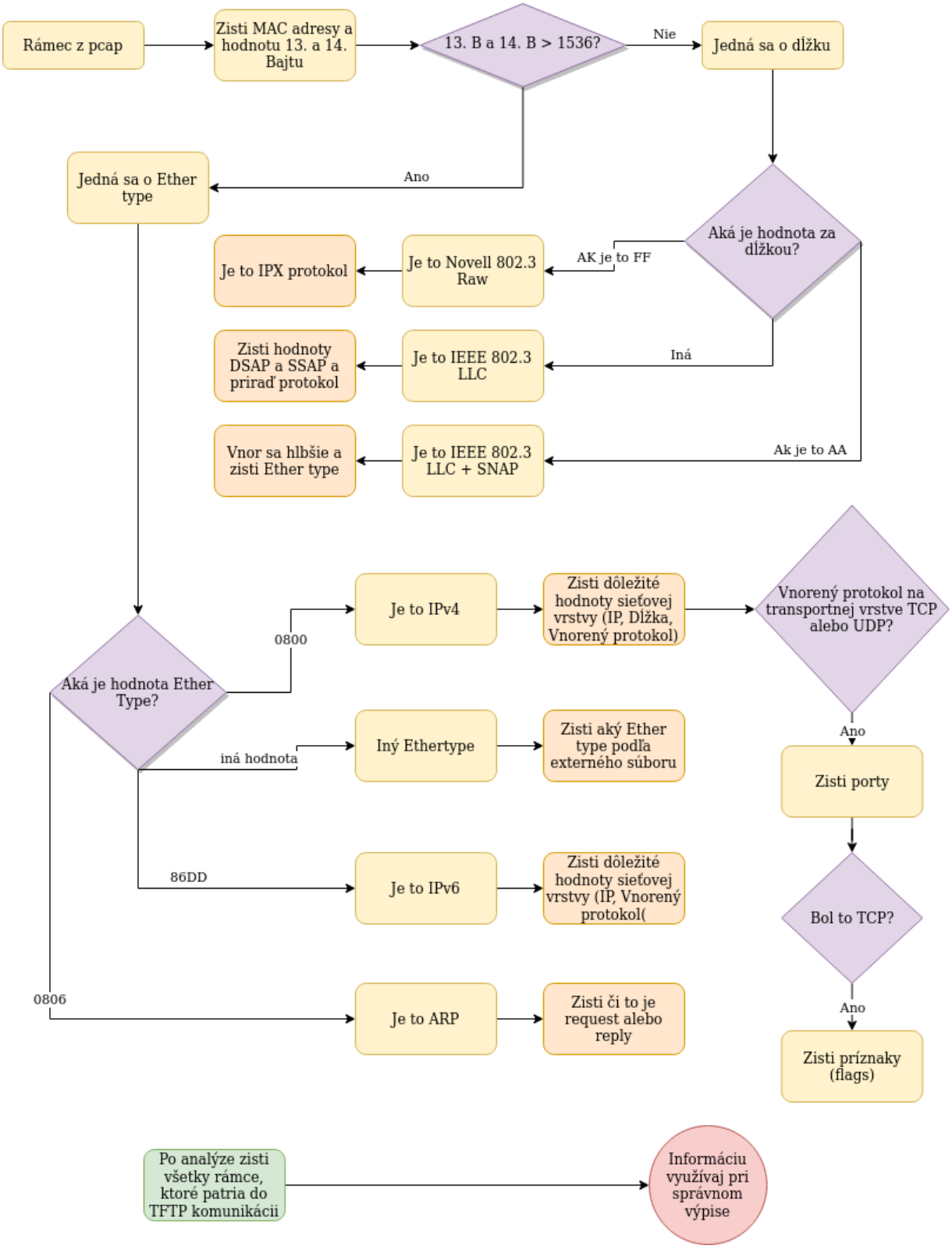
8) Študent musí byť schopný preložiť a spustiť program v miestnosti, v ktorej má cvičenia. V prípade dištančnej výučby musí byť študent schopný prezentovať podľa pokynov cvičiaceho program online, napr. cez Webex.

Obsah

1 Zadanie úlohy: 2
2. Mechanizmus analyzovania jednotlivých protokolov 6
2.1 Blokový návrh fungovania riešenia: 6
2.2 Základ implementácie 7
2.3 Analýza Linkovej vrstvy 8
2.4 Analýza sieťovej vrstvy 9
2.5 Analýza transportnej vrstvy 11
3 Nájdenie najčastejšieho odosielajúceho uzla: 12
4 Hľadanie kompletných a nekompletných komunikácii TCP protokolov 13
5 Hľadanie TFTP komunikácií 19
6 Hľadanie ARP párov 20
7 ICMP komunikácie 21
8 Súbory s pomocnými funkciami a štruktúry externých súborov 22
9 Opis používateľského rozhrania 23
10 Voľba implementačného prostredia 25
Záver 25

2. Mechanizmus analyzovania jednotlivých protokolov

2.1 Blokový návrh fungovania riešenia:



2.2 Základ implementácie

Analýza popísaná ďalej je vykonávaná funkciami:

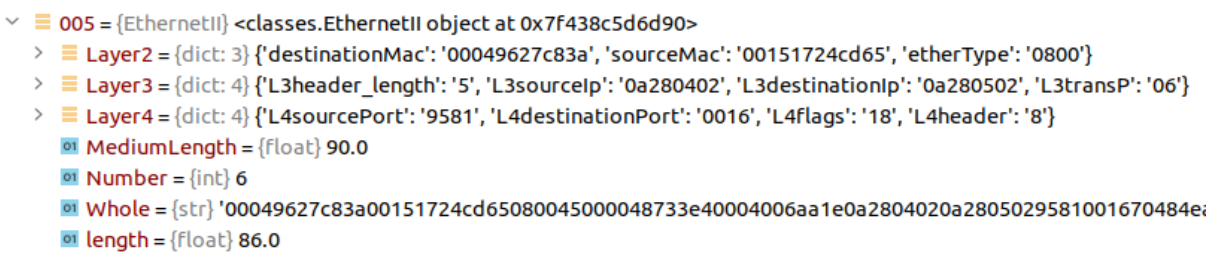
- `def Analyze(frame, Number)`
- `def handle_ETH_IPv4(frame, Number, destinationMac, sourceMac, EtherType, EtherLength)`
- `def handle_ETH_IPv6(frame, Number, destinationMac, sourceMac, EtherType, EtherLength)`

Zistené hodnoty v rámcoch sa ukladajú do štyroch typov objektov:

- `class EthernetII`
- `class IEEE_NovellRaw`
- `class IEEE_802_LLC`
- `class IEEE_802_LLC_SNAP`

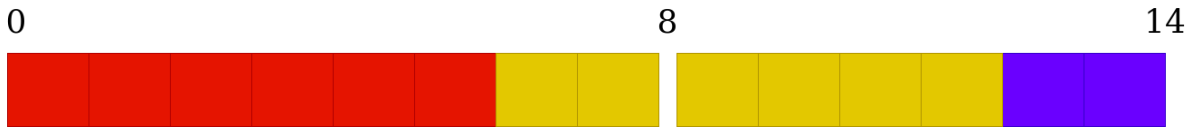
Triedy `class IEEE_NovellRaw`, `class IEEE_802_LLC` a `class IEEE_802_LLC_SNAP` majú podtriedu `class Layer2`. Do nej sa ukladajú všetky parametre ktoré patria do linkovej vrstvy.

Trieda `class EthernetII` je komplexnejšia. Pri niektorých protokoloch ako TCP sa musíme vnoriť hlbšie a analyzovať viac dát. Preto trieda obsahuje slovníky s názvom L2, L3, L4. Do týchto slovníkov sa prerozdelia parametre poslané ako argumenty viazané na kľúč. Do daných slovníkov sa prerozdelia podľa názvu kľúča. Napríklad zdrojová IPv4 adresa patrí do sieťovej vrstvy, takže názov kľúča pre jej premennú je “L3sourceIp”. Nazvaním premenných takýmto štýlom si program efektívne a prehľadne rozdelí dané premenné do potrebných vrstiev a práca s takto vytvorenými objektami bude pohodlnejšia. Taktiež program bude efektívnejší na doimplementáciu nových funkcií.



Trieda EthernetII ktorá obsahuje IPv4 a TCP.

2.3 Analýza Linkovej vrstvy



- Cieľová MAC adresa - Červená
- Zdrojová MAC adresa - Žltá
- Ether type / dĺžka - Modrá

Program načíta rámec z PCAP súboru. Z linkovej vrstvy zistí zdrojovú a cieľovú MAC adresu a zistí hodnoty na 13. a 14. bajte (dĺžka alebo ether type). Ak je hodnota väčšia ako 1536 jedná sa o ether type a ide o Ethernet II.

Ak je hodnota rovná hexadecimálnej hodnote 0800 je to IPv4, ak 0806 ide o ARP, ak 86DD je to IPv6, ináč to je iný protokol (zisťuje z externých súborov).

Ak je hodnota za dĺžkou rovná hexadecimálnej hodnote FF je to protokol Novell 802.3 Raw, ak je rovná AA je to IEEE 802.3 LLC + SNAP ak je iná je to IEEE 802.3 LLC a hodnoty DSAP a SSAP treba zistiť pomocou externých súborov.

Analýzu vykonáva funkcia `def Analyze(frame, Number)` v súbore analyzation.py .

Analýza IEEE 802.3 typov:

A. IEEE Novell 802.3 RAW

Ak program zistil, že za hodnotou dĺžky je hodnota FF, je to typ IEEE Novell 802.3 RAW a hneď sa mu môže priradiť protokol IPX

B. IEEE 802.3 LLC

Ak program zistil, že za hodnotou dĺžky nie je hodnota FF ani AA, je to typ IEEE 802.3 LLC. V ňom treba zistiť príslušné protokoly pre hodnoty DSAP a SSAP (externý súbor LLC_SAP.txt)

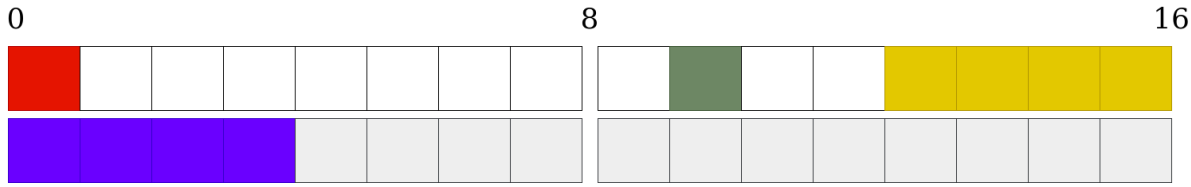
C. IEEE 802.3 LLC + SNAP

Ak program zistil, že za hodnotou dĺžky je hodnota AA, je to typ IEEE 802.3 LLC + SNAP. V tomto prípade sa program vnorí hlbšie a zistí aj vendor code, control a ether type (dôležitý je ten ether type)

Analýzu vykonáva funkcia: `def Analyze(frame, Number)` v súbore analyzation.py

2.4 Analýza sieťovej vrstvy

Protokol IPv4:



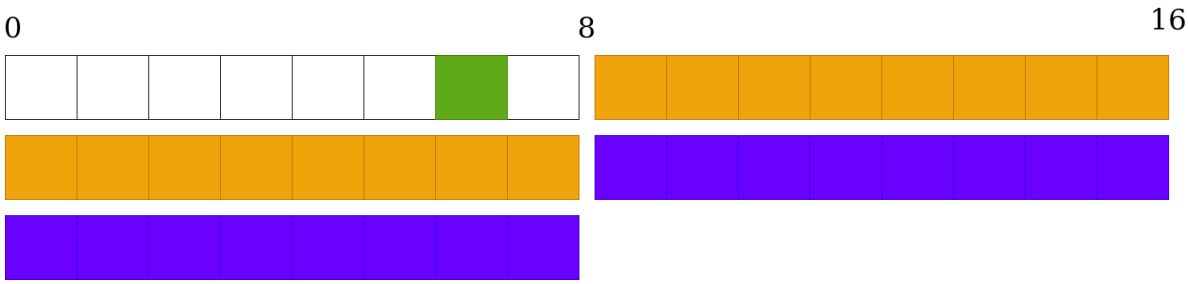
- Veľkosť IPv4 protokolu (prvá polovica prvého bajtu) - Červená
- Vnorený protokol na transportnej vrstve - Zelená
- IPv4 zdrojová adresa - Žltá
- IPv4 cieľová adresa - Modrá
- Dobrovoľné pole (optional) - Sivá

Pri vnorení do sieťovej vrstvy IPv4 program zanalyzuje druhú polovicu prvého bajtu, čím zistí dĺžku IPv4. Hodnota dĺžky znamená štvor-bajtové slová, t.j. hodnota 5 znamená, že vrstva má dĺžku 20 bajtov. Hodnoty sa pohybujú od 5 (20B) do 15 (60B) a indikujú aj prítomnosť poľa Optional. Ďalej na 10 bajte zanalyzuje vnorený protokol na transportnej vrstve (určí ho podľa externého súboru transmission_protocols.txt). Na posledných 8 bajtoch nájde zdrojovú a cieľovú IPv4 adresu.

Analýzu vykonáva funkcia:

`def handle_ETH_IPv4(frame, Number, destinationMac, sourceMac, EtherType, EtherLength)` v súbore eth2handlers.py .

Protokol IPv6:

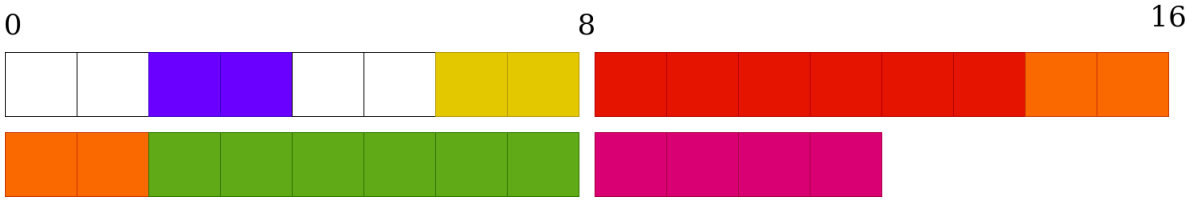


- Vnorený protokol na transportnej vrstve - Zelená
- IPv6 zdrojová adresa - Oranžová
- IPv6 cieľová adresa - Modrá

Pri vnorení na sieťovú vrstvu IPv6 program zanalyzuje typ vnoreného protokolu na transportnej vrstve (určí ho podľa externého súboru transmission_protocols.txt), zdrojovú a cieľovú MAC adresu. Dĺžka protokolu IPv6 je vždy konštantná a to je 40 bajtov.

Analýzu vykonáva funkcia: `def handle_ETH_IPv6(frame, Number, destinationMac, sourceMac, EtherType, EtherLength)` v súbore eth2handlers.py .

Address resolution protokol (ARP):



- Protokol (väčšinou IPv4) - Modrá
- Kód operácie - žltá
- Zdrojová MAC adresa - Červená
- Zdrojová IP adresa - Oranžová
- Cieľová MAC adresa - Zelená
- Cieľová IP adresa - Ružová

Pri analyzovaní sieťovej vrstvy ARP program zanalyzuje protokol, kód operácie, zdrojovú MAC a IP adresa a cieľovú MAC a IP adresu.

Analýzu vykonáva funkcia:

`def Analyze(frame, Number)` v súbore analyzation.py

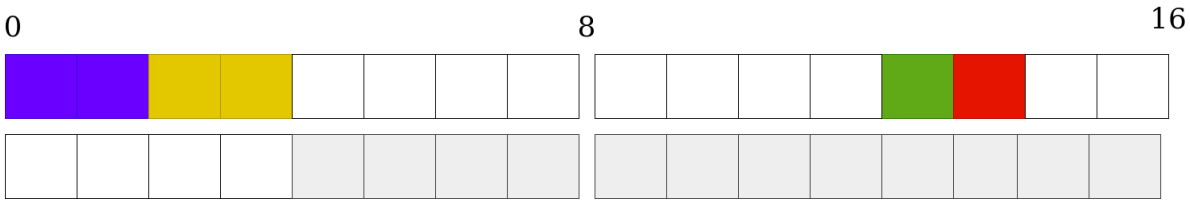
Nájdenie iného ether type-u:

Pri nájdení iného ether type-u ako IPv4, IPv6 a ARP, program sa nevnorí na hlbšie vrstvy ako je linková. Určí, ak vie, o aký ether type ide podľa externého súboru (EtherType_Values.txt).

2.5 Analýza transportnej vrstvy

Analýza transportnej vrstvy prebehne, len ak protokol vo sieťovej vrstve bol IPv4.

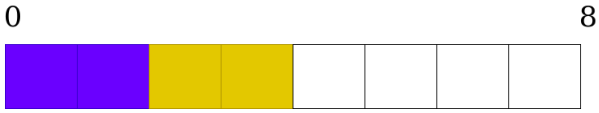
TCP (transmission control protocol):



- Zdrojový port - Modrá
- Cieľový port - Žltá
- Veľkosť TCP vrstvy (prvá polovica bajtu) - Zelená
- Príznaky (Flags) - Červená a druhá polovica Zelenej
- Dobrovoľné pole (optional) - Sivá

Analýza TCP protokolu prebieha tak, že program zistí aký je zdrojový a cieľový port. Taktiež zistí, veľkosť transportnej vrstvy pomocou hodnoty prvej polovici bajtu na 13 mieste. Druhá polovica bajtu na 13. mieste spolu s 14. bajtom tvoria príznaky (Flags). V mojej implementácii ale druhú polovicu z 13. bajtu odignoruje, keďže tie konkrétne príznaky nás nebudú zaujímať. Pomocou externého súboru (TCPports.txt) zistí či hodnota niektorého portov nie je nejaký konkrétny poznaný (http, https, ssh, …).

UDP (user datagram protocol):



- Zdrojový port - Modrá
- Cieľový port - Žltá

UDP protokol sa analyzuje tak, že zistí zdrojový a cieľový port. Pomocou hodnôt portov zistí, či sa jedná o špeciálny typ komunikácií (externý súbor UDPports.txt)

3 Nájdenie najčastejšieho odosielajúceho uzla:

Nájdenie najčastejšieho uzla sa vykoná priamo po vypísaní všetkých rámcov nasledujúcim kódom:

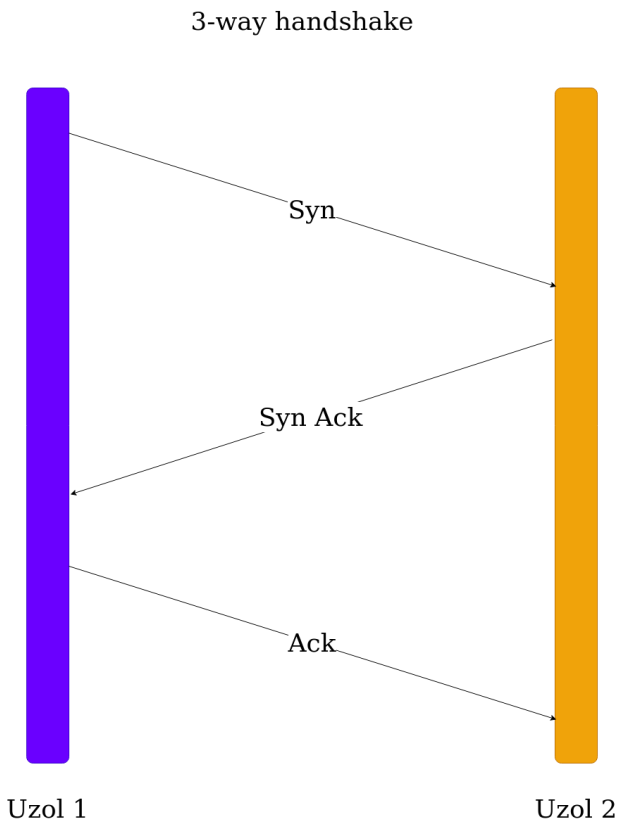
```
counter = 0
uniq_ip = [0]
for ip in ipv4s:  # cyklus zisťujúci najfrekventovanejšiu IP zdrojovú IP
adresu
   if ip not in uniq_ip:
        frequency = ipv4s.count(ip)
        uniq_ip.append(ip)  # pole unikátnych IP adries
        uniq_ip[0] += 1
        if frequency > counter: # ak našiel novú frekventovanejšiu IP
            counter = frequency
            freqIP = ip
        frequency = 0
        print("{number}. IPv4: {ip}".format(number=uniq_ip[0], ip=editIP(ip,
"0800")))

print("Most frequent ip:", editIP(freqIP, "0800")) # výpis
print("Number of sendings:", counter)
```

For cyklom sa prejdú všetky rámce a postupne sa pridávajú jedinečné IP adresy zdrojových uzlov do poľa `uniq_ip`. Ak nájde novú jedinečnú zdrojovú IP adresu, tak zistí jej frekvenciu v celom zázname rámcov. Ak je jej frekvencia väčšia ako posledná najväčšia IP adresa s najväčšou frekvenciou (uložená v premennej counter), tak sa nová najfrekventovanejšia zdrojová IP adresa zapíše do premennej freqIP a jej frekvencia do premennej counter. Jedinečné IP adresy zdrojových uzlov sa pri ich hľadaní vypisujú. Na konci sa vypíše najfrekventovanejšia IP spolu s počtom odoslaní.

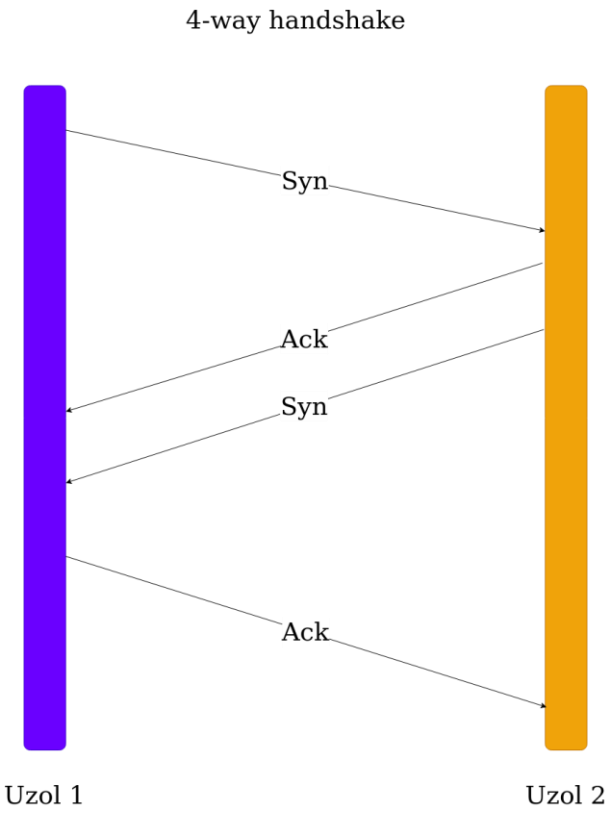
4 Hľadanie kompletných a nekompletných komunikácii TCP protokolov

Program má vedieť nájsť správne ukončené (zároveň aj správne začaté) a nesprávne ukončené - neukončené - ale správne začaté komunikácie. Tieto komunikácie máme vedieť analyzovať pre protokoly: http, https, telnet, ssh, ftp riadiace a ftp dátové. Na to aby analyzátor vedel nasledovné komunikácie nájsť musíme vedieť ako začína a končí TCP komunikácia.

Začatie komunikácie

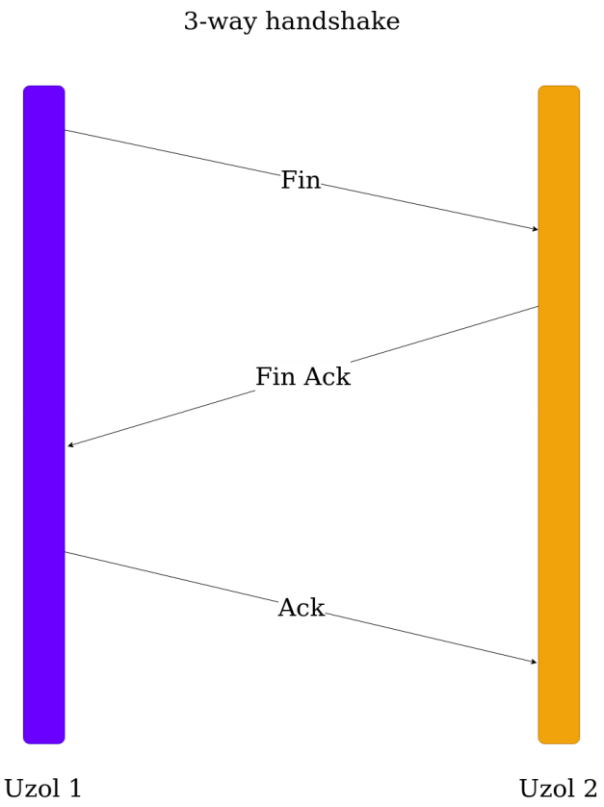


TCP komunikácia môže začať 3-way handshake-om. V tomto prípade uzol 1 pošle príznak Syn uzlu 2 čím signalizuje záujem o nadviazanie spojenia. Nasledovne mu uzol 2 pošle príznaky Syn Ack. Príznakom Ack potvrdzuje prijatie od uzla 1 príznaku Syn, a príznakom Syn prejavuje tiež záujem o začatie komunikácie. Uzol 1 mu na to pošle príznak Ack čím potvrdí prijatie jeho príznaku Syn.



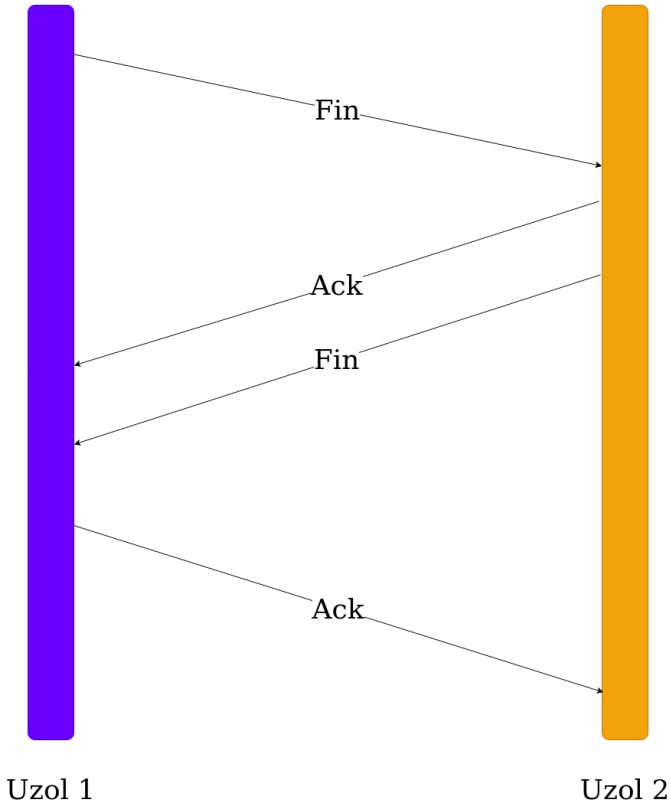
Komunikácia môže začať aj 4-way handshake-om. Princíp tohoto začatia je podobný ako 3-way handshake-u vyššie. Uzol 1 pošle príznak Syn ktorým prejaví záujem o začatie komunikácie. Tu sa ukáže rozdiel medzi 3-way a 4-way handshake-om. Miesto toho aby uzol 2 poslal naraz príznaky Syn a Ack, tak ich pošle zvlášť. Najprv pošle uzol 2 príznak Ack ako potvrdenie prijatia príznaku Syn od uzla 1 a na to hneď posiela príznak Syn. Uzol jedna musí potvrdiť prijatie príznaku Syn poslaním príznaku Ack.

Ukončovanie komunikácie



Ukončenie komunikácie 3-way handshake-om je princípovo rovnaké ako začatie 3-way handshake-om. Uzol 1 pošle príznak Fin aby prejavil záujem o ukončenie komunikácie. Uzol 2 mu pošle príznaky Fin Ack aby taktiež prejavil záujem o ukončenie komunikácie a zároveň potvrdil prijatie príznaku Fin od uzla 1. Uzol 1 následne potvrdí prijatie príznaku Fin od uzla 2 poslaním príznaku Ack.

4-way handshake



Ukončenie komunikácie 4-way handshake-om je zas princípovo rovnaké ako začatie komunikácie 4-way handshake-om. Uzol 1 pošle príznak Fin aby prejavil záujem o ukončenie komunikácie. Následne Uzol 2 pošle príznak Ack čím potvrdí prijatie príznake Fin. Uzol 2 pošle príznak Fin. Uzol 1 mu pošle príznak Ack ako potvrdenie o prijatí príznaku Fin.

Ukončenie komunikácie môže prebehnúť aj keď len jeden z uzlov pošle príznak Reset. Tým sa komunikácia okamžite považuje za ukončenú.

Môže sa ale stať že pri ukončovaní prvý príznak nebude len Fin ale napríklad Fin Ack(uzol potvrdzuje prijatie predošlého rámca ale zároveň prejavuje záujem o ukončenie komunikácie), ba dokonca aj Fin Push Ack (Uzol potvrdzuje prijatie predošlého rámca v komunikácii, ale zároveň prejavuje záujem o ukončenie komunikácie, pričom ešte posiela druhému uzlu dáta - v tomto prípade druhý uzol musí zvlášť potvrdiť príznakmi Ack príznaky Push aj Fin).

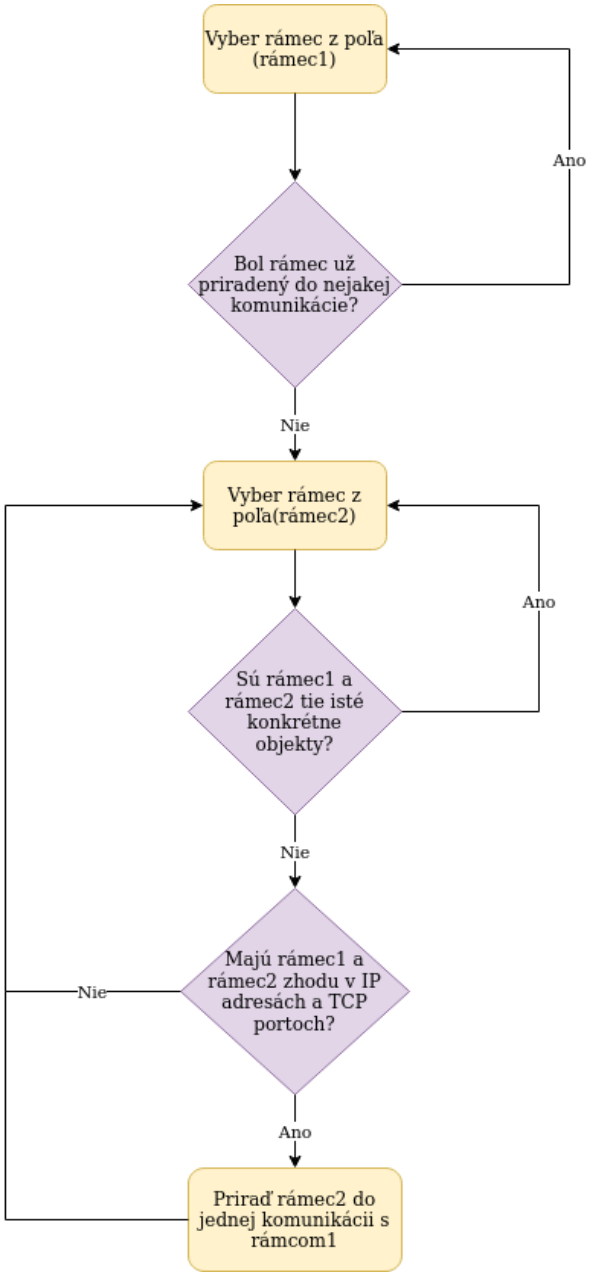
Implementácia

Nájdenie komunikácii a určovanie kompletnosti realizuje program pomocou funkcií:

1. `def completeComunication(frames, type)`
2. `def groupUpFrames(type_frames)`
3. `def isframeused(frame, grouped_frames)`
4. `def findWholeCommunication(grouped_frames)`

Funkcia `def completeComunication(frames, type)` najprv vytvorí pole ktoré obsahuje len objekty (rámce) typu Ethernet-II a zároveň je jeden z portov zhodný s hodnotou argumentu `type` (napríklad komunikácia http má port 80). Následne sa nad týmto poľom zavolá funkcia `def groupUpFrames(type_frames)`, zoskupí jednotlivé rámce, ktoré patria do jednej komunikácie.

Blokový návrh:

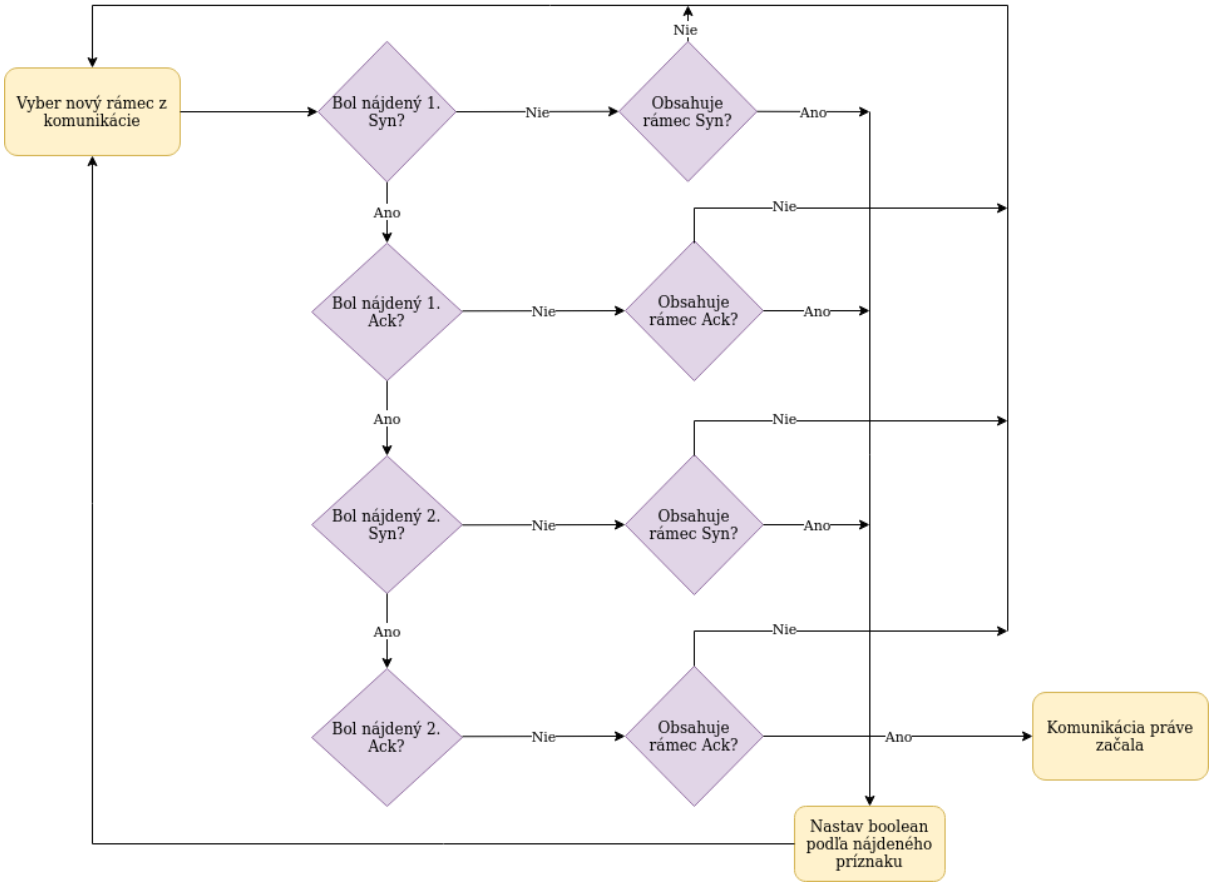


Zisťovanie či prvý porovnávaný rámec už bol pridelený do komunikácie robí funkcia `def isframeused(frame, grouped_frames).`

Dôležité je ale aj zistiť či v našom PCAP súbore môžeme nájdenú komunikáciu považovať za komunikáciu, a ak áno, či bola ukončená správne. Zistenie tejto charakteristiky nad danou komunikáciou robí funkcia:
`def findWholeCommunication(grouped_frames)`

Funkcia prechádza postupne komunikácie a jednotlivé rámce v komunikáciách. Jednotlivé príznaky hľadá podľa nasledovnej blokovej schémy.

Bloková schéma na zistenie či komunikácia správne začala:



V schéme vyššie sa neberie do úvahy podmienka, že konkrétne príznaky musia ísť od konkrétnych uzlov (znázornené na obrázkoch “Začatie komunikácie”

Ak komunikácia začala (uzly nadviazali spojenie), tak sa ďalej rovnakým princípom zisťuje správne ukončenie komunikácii uzlov pomocou príznakov Fin a príznakov Ack. Jediný rozdiel je ten, že ak príde príznak Reset tak komunikácia bola automaticky ukončená, nevyžaduje sa potvrdenie druhého uzla.

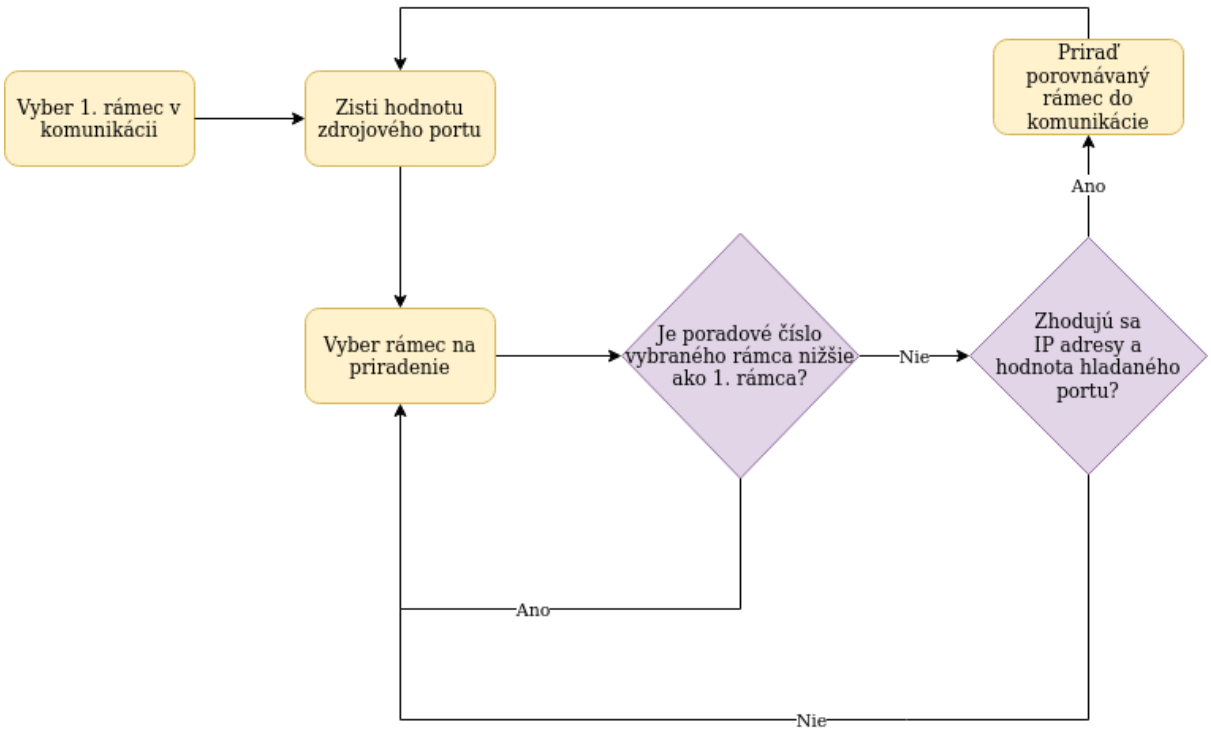
Ak sa jednotlivé komunikácie správne rozdelia na kompletné a nekompletné, program vypíše 1. kompletnú a prvú nekompletnú.

5 Hľadanie TFTP komunikácií

Hľadanie a izolovanie TFTP komunikácii realizuje funkcia `def tftpcommunication(frames, mode).` Vo funkcii sa najprv izolujú rámce ktoré sú začiatkom TFTP komunikácií. Keďže indikátor začatia TFTP komunikácie je to, keď rámec má cieľový port 69, tak program vie jednoducho zistiť rámce začínajúce jednotlivé TFTP komunikácie a tým aj počet komunikácií.

Ďalšie rámce v konkrétnej komunikácii sa rozoznávajú podľa hodnoty zdrojového portu prvého rámca. V tejto implementácii môže nastať chyba ak port bol používaný v inej TFTP komunikácii predtým. Túto chybu ošetríme tak, že ak poradové číslo rámca, ktorý sa priraďuje ku konkrétnej komunikácii je menšie ako poradové číslo prvého rámca v komunikácii, program rozhodne, že do komunikácie určite nepatrí aj kebyže má zhodné IP adresy a používa sa hladaný port.

Stručný blokový opis priraďovania rámcov ku rámcu s hodnotou portu 69 (prvý rámec v komunikácii):



Argument mode funkcie slúži na to, že ak sa mode pošle s hodnotou True, tak funkcia vráti čísla rámcov, ktoré sú TFTP rámce a urobí tak na konci základnej analýzy rámcov. Týmto spôsobom pri výpisoch rámcov TFTP, ktoré nemajú hodnotu protu 69 ale nejakú inú, zaručím, že výpis bude správny.

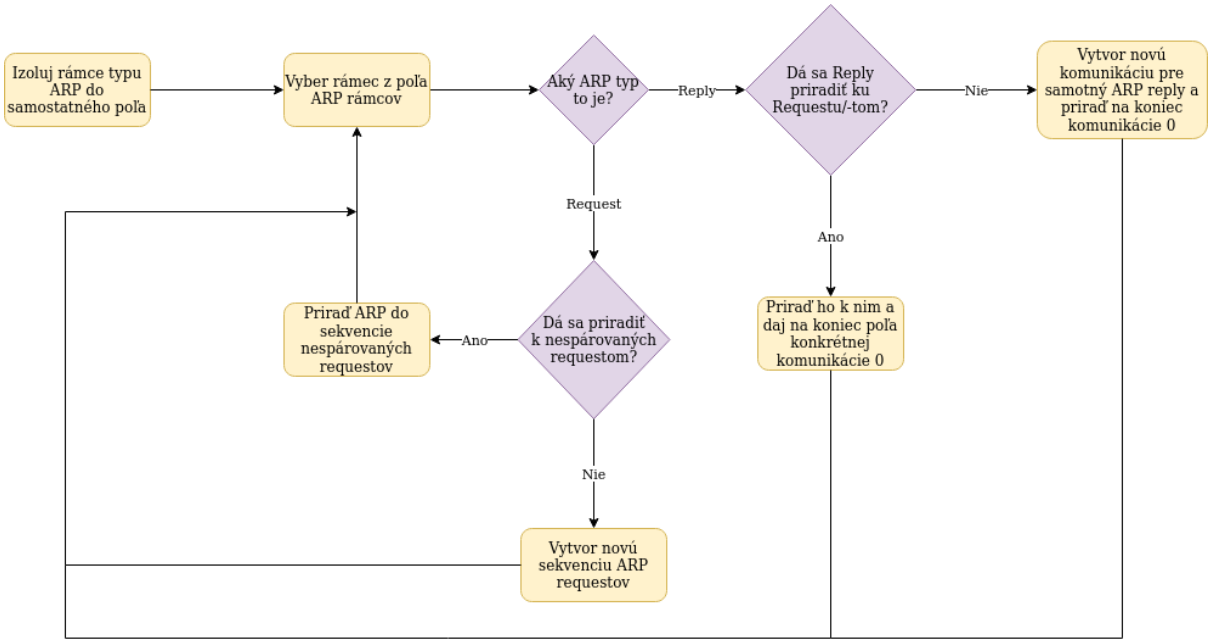
6 Hľadanie ARP párov

Analýza ARP komunikácii a hľadanie ARP párov realizujú funkcie:

- `def arpcom(frames):`
- `def checkArpReply(frame, arp_communication):`
- `def checkArpRequest(frame, arp_communication):`

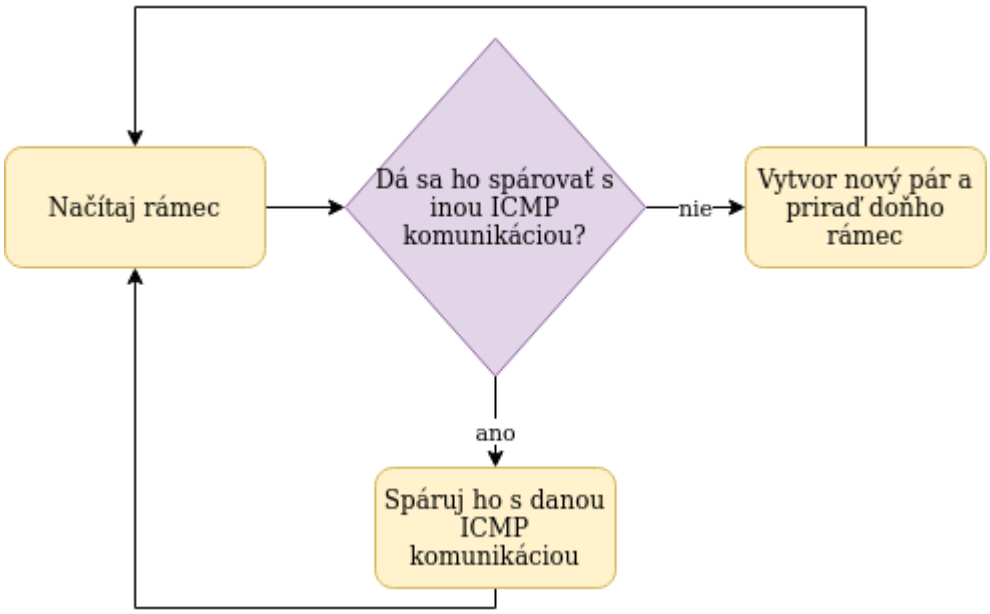
Program si najprv vytvorí pole ktoré obsahuje iba rámce typu ARP. Na to For cyklom prejde všetky ARP rámce, pričom ich združuje do jednotlivých ARP párov (ak bolo poslaných viac ARP requestov toho istého typu, združuje ich do jedného páru). Porovnávajú sa IP adresy a odosielajúce a cieľové MAC adresy. V poli sa pridá na koniec 0, ktorá je indikátor toho, že s týmto ARP párom sa už nedá nič združovať (je to samostatný reply bez requestu alebo requesty dostali reply).

Logika tohto procesu je opísaná na blokovom opise nižšie.



7 ICMP komunikácie

ICMP komunikácia sa páruje podľa zhodných IP adries. Párovanie realizujú funkcie `def icmpCom(frames)` a `def check_pairs(frame, paired_comm)` v súbore icmpCommunication.py. Najprv sa vo funkcii vytriedia všetky rámce len na rámce typu ICMP. Potom sa realizuje párovanie do komunikácií podľa IP adries. Jednoduchosť tohto princípu je zaznamenaná na blokovom diagrame:



Kontrolu či je možné spárovať kontrolovaný rámec s inými sa deje volaním funkcie `def check_pairs(frame, paired_comm)`.

8 Súbory s pomocnými funkciami a štruktúry externých súborov

Medzi súbory obsahujúce pomocné funkcie, ktoré nepatria ku priamej logike spracovania rámcov sú:

- printing.py
- textedits.py

Súbor printing.py obsahuje funkciu `def printPacket(frame)`, ktorou program vypisuje správne všetky spracované a analyzované ramce na užívateľské rozhranie.

Súbor textedits.py obsahuje načítania všetkých externých súborov do slovníkov v programe a zároveň rôzne funkcie, ktoré upravujú výpis IP adries, dát rámca, atď.

Medzi externé súbory patria súbory:

- EtherType_Values.txt - obsahuje rôzne hodnoty Ether type-ov
- flags.txt - obsahuje význam bitov príznakov v TCP komunikácii
- LLC_SAP.txt - obsahuje SAP protokoly
- TCPports.txt - obsahuje známe TCP porty
- UDPports.txt - obsahuje známe UDP porty
- transmission_protocols.txt - obsahuje hodnoty rôzne transportných protokolov
- ICMP_massages - obsahuje typy ICMP správ

Príklad externého súboru (EtherType_Values.txt):

0200=XEROX PUP
0201=PUP Addr Trans
0800=Internet IP (IPv4)
0801=X.75 Internet
0805=X.25 Level 3
0806=ARP (Address Resolution Protocol)
8035=Reverse ARP
809b=Appletalk
80f3=AppleTalk AARP (Kinetics)
8100=IEEE 802.1Q VLAN-tagged frames
8137=Novell IPX
86dd=IPv6
880b=PPP
8847=MPLS
8848=MPLS with upstream-assigned label
8863=PPPoE Discovery Stage
8864=PPPoE Session Stage

2004=DTP

88cc=LLDP

Najprv je napísaná hexadecimálna hodnota daného protokolu. Rovná sa slúži ako odďelovač. Za rovná sa je napísaný názov protokolu, ktorý prislúcha ku danej hexadecimálnej hodnote. Program načíta takýto súbor a vytvorí slovník s kľúčmi hexadecimálnych hodnôt, ktoré odomknú názov daného protokolu. Ak by prokram vypíše “None”, znamená to že pre danú hexadecimálnu hodnotu neexistuje v slovníku protokol.

9 Opis používateľského rozhrania

Pri zapnutí programu sa ukáže v termináli výpis:

```
##############################################
#.......$$$$$$$\  $$\   $$\  $$$$$$\  .......#
#.......$$  __$$\ $$ | $$  |$$  __$$\........#
#.......$$ |  $$ |$$ |$$  / $$ /  \__|.......#
#.......$$$$$$$  |$$$$$  /  \$$$$$$\.........#
#.......$$  ____/ $$  $$<    \____$$\........#
#.......$$ |      $$ |\$$\  $$\   $$ |.......#
#.......$$ |      $$ | \$$\ \$$$$$$  |.......#
#.......\__|      \__|  \__| \______/ .......#
##############################################

Which pcap file do you want to open? (pcap file needs to be in the same directory as this program - format: filename)
```

Program čaká na vstup od užívateľa (pcap súbor). Pcap súbor treba napísať bez .pcap prípony. Po zadaní súboru sa objaví nasledovný výpis:

```
##############################################
#.......$$$$$$$\  $$\   $$\  $$$$$$\  .......#
#.......$$  __$$\ $$ | $$  |$$  __$$\........#
#.......$$ |  $$ |$$ |$$  / $$ /  \__|.......#
#.......$$$$$$$  |$$$$$  /  \$$$$$$\.........#
#.......$$  ____/ $$  $$<    \____$$\........#
#.......$$ |      $$ |\$$\  $$\   $$ |.......#
#.......$$ |      $$ | \$$\ \$$$$$$  |.......#
#.......\__|      \__|  \__| \______/ .......#
##############################################

Which pcap file do you want to open? (pcap file needs to be in the same directory as this program - format: filename)
trace-27
Do you want the output to be written into a file? [1/0]
0
```

Ak je pcap súbor moc dlhý, je možné, že sa do terminálu nezmestí. Preto užívateľ má na výber výpis zapisovať do súboru output.txt (ak zadá 1) alebo do terminálu (ak zadá 1).

Po výbere sa zobrazí nasledovné užívateľské menu:

```
trace-27    stdout to output.txt: 0

What would you like to see?
1) All
2) HTTP
3) HTTPS
4) TELNET
5) SSH
6) FTP control
7) FTP data
8) TFTP
9) ICMP
10) ARP
change) Change source pcap file
toggle) Toogle writing in output.txt
end) End program
```

Zadaním výrazu pred zátvorkou (1, 2, change, end, ...) používateľ zvolí danú možnosť. Hore je názov daného otvoreného súboru a druh výpisu (0 - do terminái, 1 do súboru output.txt)

1 - vypíše všetky rámce a najčastejšiu odosielajúcu IP adresu
2 až 7 - vypíše danú kompletnú a nekompletnú komunikáciu
8 - vypíše TFTP komunikácie
9 - vypíše ICMP rámce
10 - vypíše ARP komunikácie
change - zmena pcap súboru
toggle - vypne/zapne zapisovanie do súboru output.txt
end - koniec programu

10 Voľba implementačného prostredia

Program je napísaný v jazyku python. Verzia pythonu, ktorú som použil je 3.9. Taktiež som použil knižnicu scapy (len na otvorenie pcap súboru). O knižnici a jej nainštalovaní je písané v súbory README.txt. Obratnosť a ľahká doimplementovateľnosť jazyka python ma presvedčili o tom, že python bude najlepšia voľba.

IDE ktoré som použil je PyCharm Professional od spoločnosti JetBrains.

Záver

V tomto zadaní sme mali vytvoriť analyzátor sieťovej komunikácie. Analyzátor je schopný indetifikovať a vypísať najhlavnejšie parametre konkrétného rámca. Komunikácie typu IPv4, TCP, ARP, TFTP a ICMP analyzuje hlbšie. Program vie korektne a efektívne zpárovať potrebné komunikácie. Práca na projekte ma naučila analyzovať rámce ručne, a hlbšiemu chápaniu sieťových komunikácii. Vylepšenie mojej implementácie by sa dalo programovaním v procedurálne zameraných jazykoch ako napríklad C. Avšak doimplementácia a rozširovanie projektu je ľahšie v jazykoch ako python.