Slovenská technická univerzita v Bratislave

Fakulta informatiky a informačných technológii

1. Október 2021

Počítačové a komunikačné siete

Analyzátor sieťovej komunikácie

Roland Vdovják
id: 110912

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 pomédiu.

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) **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, Meet, etc.

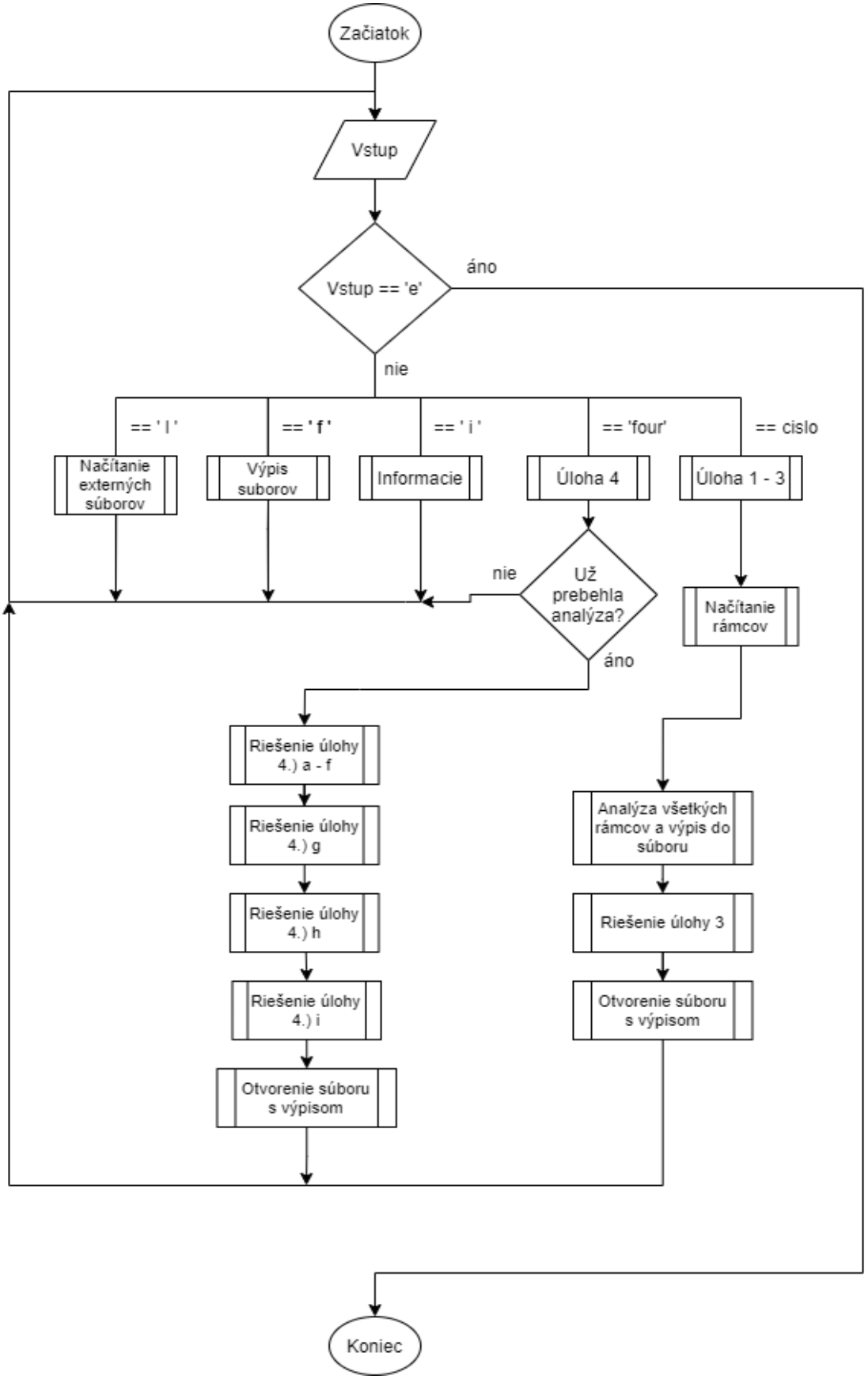
V danom týždni, podľa harmonogramu cvičení, musí študent priamo na cvičení doimplementovať do funkčného programu (podľa vyššie uvedených požiadaviek) ďalšiu prídavnú funkčnosť.

Program musí mať nasledovné vlastnosti (minimálne):

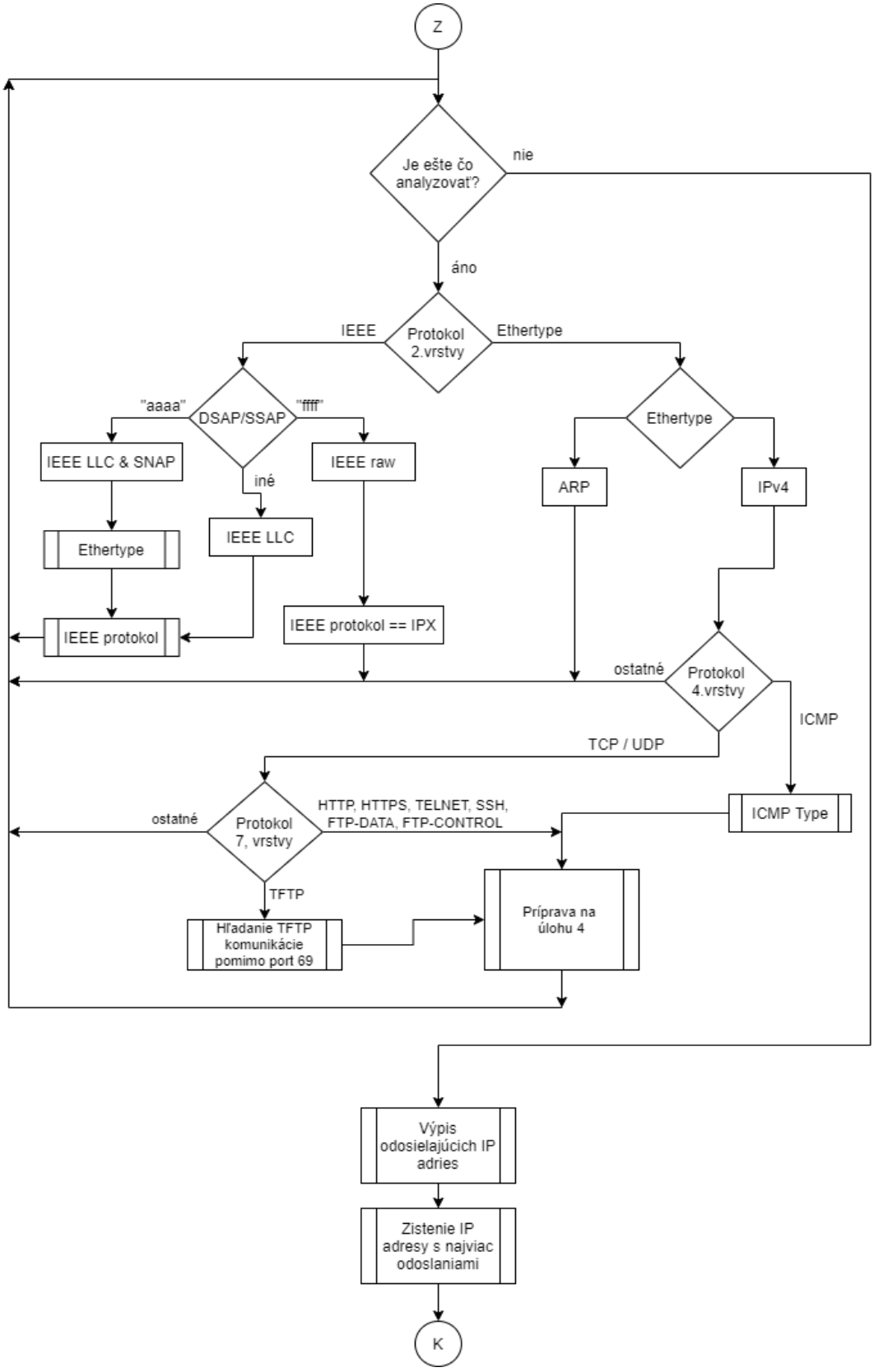
1) Program musí byť implementovaný v jazykoch C/C++ alebo Python s využitím knižnice pcap, skompilovateľný a spustiteľný v učebniach. Na otvorenie pcap súborov použite knižnice libpcap pre linux/BSD a winpcap/ npcap pre Windows. Použité knižnice a funkcie musia byť schválené cvičiacim. V programe môžu byť použité údaje o dĺžke rámca zo struct pcap_pkthdr a funkcie na prácu s pcap súborom a načítanie rámcov:
   pcap_createsrcstr()
   pcap_open()
   pcap_open_offline()
   pcap_close()
   pcap_next_ex()
   pcap_loop()
   Použitie funkcionality libpcap na priamy výpis konkrétnych polí rámca (napr. ih->saddr) bude mať za následok nulové hodnotenie celého zadania.
2) Program musí pracovať s dátami optimálne (napr. neukladať MAC adresy do 6x int).
3) Poradové číslo rámca vo výpise programu musí byť zhodné s číslom rámca v Analyzovanom súbore.
4) Pri finálnom odovzdaní, pre každý rámec vo všetkých výpisoch uviesť použitý protokol na 2. 4. vrstve OSI modelu. (ak existuje)
5) Pri finálnom odovzdaní, pre každý rámec vo všetkých výpisoch uviesť zdrojovú a cieľovú adresu / port na 2. - 4. vrstve OSI modelu. (ak existuje)

Nesplnenie ktoréhokoľvek bodu minimálnych požiadaviek znamená neakceptovanie riešenia cvičiacim.

Blokový návrh (koncepcia) fungovania riešenia



Navrhnutý mechanizmus analyzovania protokolov na jednotlivých vrstvách



Pre reprezentovanie mechanizmu analyzovania protokolov na vrstvách sme sa rozhodli použiť blokový diagram, ktorý opisuje cestu analýzy testovacieho *.pcap* súboru. Tento diagram je reprezentácia bloku s procesom „Analýza všetkých rámcov a výpis do súboru“ v Blokovej koncepcii riešenia.

Pri analýze rámca zistené informácie rovno zapisujem do súboru výpis. Program si nezapamätáva všetky rámce, len tie rámce potrebné na riešenie úlohy (*HTTP, HTTPS, TELNET, SSH, FTP-DATA, FTP-CONTROL, TFTP, ICMP, ARP*). Rámce s týmito protokolmi vkladáme do dátových štruktúr už v komunikačných pároch.

Tento screenshot zobrazuje štruktúru pre ARP, je to zoznam, ktorý obsahuje ďalšie zoznamy, môžeme si to predstaviť ako nepravidelné 2D pole.

Jeden rozmer obsahuje všetky komunikačné dvojive, Druhý rozmer už jednotlivé rámce

Objekt „*FRAME*“ = Rámec, je nami vytvorený objekt obsahujúci všetko, čo analyzujeme.

Vďaka takto načítaným štruktúram je riešenie úlohy štyri nenáročné na čas. Nie je potrebné prídavné triedenie rámcov na základe komunikačných dvojíc.

Príklad štruktúry externých súborov pre určenie protokolov a portov

Externý súbor sme konštruovali nasledovne:

ETH_prot.txt

```
0200$XEROX PUP
0201$PUP Addr Trans
0800$IPv4
0801$X.75 Internet
0805$X.25 Level 3
0806$ARP
8035$Reverse ARP
809b$Appletalk
80f3$AppleTalk AARP
8100$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$PPPoE Session Stage
88cc$Link Layer Discovery Protocol (LLDP)
```

IEEE_prot.txt

```
00$Null SAP
02$LLC Sublayer Management / Individual
03$LLC Sublayer Management / Group
06$IP (DoD Internet Protocol)
0e$PROWAY
42$BPDU
4e$MMS EIA-RS 511
5e$ISI IP
7e$X.25 PLP
8e$PROWAY
aa$SNAP
e0$IPX
f4$LAN Management
fe$ISO Network Layer Protocols
ff$Global DSAP
```

Externé súbory obsahujú prvé 4 alebo 2 znaky hexadecimálnu hodnotu, oddeľovač ‚$' a meno protokolu rovnajúceho sa danej hodnote. Oddeľovač slúži na spadnutie programu, ak je zle naprogramované čítanie alebo formát súboru. Čítajú sa prvé 2 alebo 4 znaky, ak sa tu vyskytne dolár, čo nie je znak hexadecimálnej hodnoty, vieme že súbor je naformátovaný zle. Súbory sú tvorené tak, aby IEEE a ICMP boli 2-znakové, ostatné 4-znakové

Z externých súborov sa dáta čítajú do polí. Každý súbor má svoje vlastné pole.

Používateľské rozhranie

Používateľské rozhranie sme sa snažili urobiť intuitívne a efektívne. Keď program pracuje, používateľ o tom vie. Keď používateľ zadá chybný vstup, číslo mimo rozsah, bude upozornený.

Po spustení programu sa vypíšu informácie, resp. menu, vyzerá nasledovne (rovnaký výpis je aj pri vstupe ‘ i ’ (information).):

```
                    I N F O R M A T I O N
e- Exit
f- Files
i- Information
l- Load ETH & IEE & IP & PORT
four- Make task 4


Please enter file number:
```

Program čaká na vstup od používateľa, jednotlivé vstupy aj s funkciami sú vypísané. Ak používateľ chce, môže rovno napísať číslo *pcap* súboru. **Je potrebné si dať pozor na číslo, tie sa nezhodujú s číslami v menách súborov.** Program si načíta celý priečinok so všetkými súbormi a očísluje si ich sám. Používateľ zistí číslo súboru napísaním ‘ *f* ’ (files).

Vyzerá to nasledovne:

```
Please enter file number:
f
    1 -          eth-1.pcap    2 -          eth-2.pcap    3 -          eth-3.pcap    4 -          eth-4.pcap
    6 -          eth-6.pcap    7 -          eth-7.pcap    8 -          eth-8.pcap    9 -          eth-9.pcap
   11 -        trace-1.pcap   12 -       trace-10.pcap   13 -       trace-11.pcap   14 -       trace-12.pca
   16 -       trace-14.pcap   17 -       trace-15.pcap   18 -       trace-16.pcap   19 -       trace-17.pca
   21 -       trace-19.pcap   22 -        trace-2.pcap   23 -       trace-20.pcap   24 -       trace-21.pca
   26 -       trace-23.pcap   27 -       trace-24.pcap   28 -       trace-25.pcap   29 -       trace-26.pca
   31 -        trace-3.pcap   32 -        trace-4.pcap   33 -        trace-5.pcap   34 -        trace-6.pca
   36 -        trace-8.pcap   37 -        trace-9.pcap   38 - trace_ip_nad_20_B.pcap
```

Písmeno ‘ *l* ’ (load) slúži na znovu načítanie externých súborov, aby sa aj za behu programu vedeli pridávať protokoly do externých súborov.

Vstup ‘ *four* ’ spustí vykonávanie úlohy štyri.

Počas pracovania programu používateľ vidí progres analýzy.

```
10
Loading frames from pcap folder
Progress
Start                        End
|                             |
|.............
```

Fungovanie programu

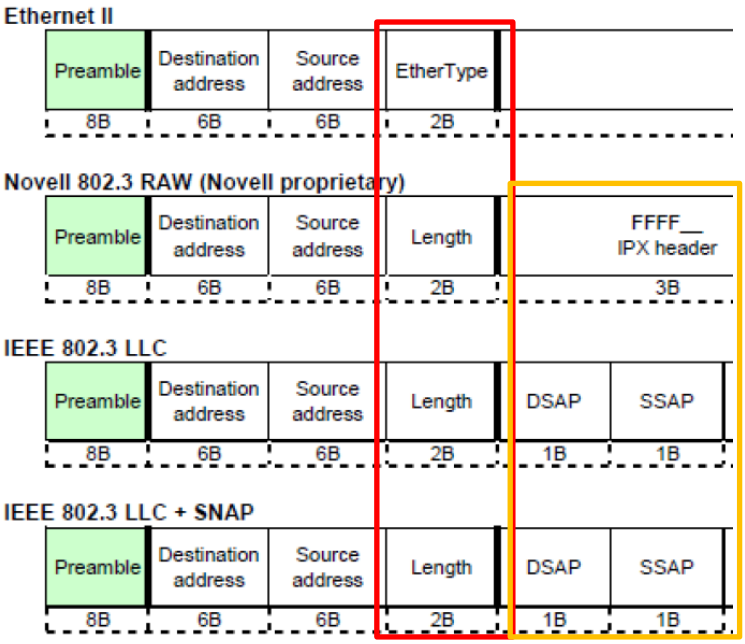
Z blokového návrhu je vidieť, ako je navrhnutý program. V tejto časti dokumentu popíšeme hlavné funkcie a main funkciu programu *main.py*. Kód máme len v tomto jednom súbore, pre správne fungovanie potrebujeme aj priečinok *Pcap* s testovacími súbormi a priečinok Prot s externými súbormi pomocou ktorých určujeme protokoly.

Dôležité funkcie

Všetky funkcie sú opodstatnené, všetky sa využívajú a sú okomentované aj v zdrojovom súbore, tu popíšeme len tie najpodstatnejšie.

def frame_type(frame):

Funkcia dostáva ako parameter celý rámec, zameriava sa na nižšie zvýraznenú oblasť rámca.



Červené zvýraznenie je oblasť, pri ktorej sa program rozhoduje, či ide o Ethernet II alebo IEEE.

Oranžová oblasť slúži na zistenie typu IEEE.

Ukážka kódu:

```
def frame_type(frame):
    f_type = ""
    global IEEE_p

    if int(frame[24:28], 16) >= 1536:
            # ETH
        f_type = "Ethernet II"
        find_ETH_p(frame[24:28]) # Zistovanie Protokolu 3. vrstvy
    else:
            #IEEE
        if str(frame[28:32]) == "ffff":
            f_type = "IEEE 802.3 - raw"
            IEEE_p = "IPX"
```

```
        elif str(frame[28:32]) == "aaaa":
            f_type = "IEEE 802.3 LLC & SNAP"

                # Zistovanie Protokolu 3. vrstvy
            find_IEEE_p(frame[32:34])
            find_ETH_p(frame[40:44])

        else:
            f_type = "IEEE 802.3 LLC"
            find_IEEE_p(frame[32:34]) # Zistovanie Protokolu 3. vrstvy

    return f_type
```

Po zistení typu rámca (protokolu druhej vrstvy) program zisťuje aj protokol tretej vrstvy, to podľa toho, ako určil druhú vrstvu. (podfarbené žlto).

def loadProt(prot):

Program deserializuje pomocou tejto funkcie, v časti Príklad štruktúry externých súborov pre určenie protokolov a portov sme opísali, ako záleží na počte miest pred oddeľovačom „ *$* “. V programe toto číslo reprezentuje premenná *n*.

```
if prot == "IEEE" or prot == "ICMP":
    n = 2
else:
    n = 4
```

Ďalej načítava program do polí, pole zvolí podľa parametra funkcie, keďže názvy polí sú vo formáte *typ_poľa*+prot (ETHprot, IEEEprot atď.)

```
list = eval(prot + "prot")

for line in lines:
    list[0].append(line[:n])              -hodnota
    list[1].append(line[n + 1:-1])                -protokol
```

def find_ETH_p(h):

Funkcia prehľadáva polia a vracia protokol prislúchajúci danej hodnote alebo prepisuje rovno globálnu premennú (prípady ETH_p a IEEE_p).

Tak ako táto funkcia fungujú aj iné a sem si vedomí toho, že by sa to dalo zjednodušiť do jednej, kvôli časovým nárokom to je však v samostatných funkciách (find_IEEE, find_PORT atď.)

Kód:

```
i = 0

while p == "" and i < len(ETHprot[0]):
    if ETHprot[0][i] == h:
```

```
            p = ETHprot[1][i]
        i += 1
    global ETH_p
    ETH_p = p
```

def add_to_list_4(list, frame):

Ako parametre sú list a rámec, rámec sa má zapísať do pod-listu listu ak splní podmienku. Podmienkou je zhoda IP adries a portov komunikačnej dvojice (vkladaného rámca a nultého rámca pod-listu), ak sa aspoň jeden údaj nezhoduje, vytvorí sa nový pod-list a ako nultý prvok sa pridá vkladaný rámec.

```
    for i in range(len(list)):
        if (set([frame.d_ip, frame.s_ip]) == set([list[i][0].s_ip,
    list[i][0].d_ip]) and set([frame.d_port, frame.s_port]) ==
    set([list[i][0].s_port, list[i][0].d_port])):
            list[i].append(frame)

            assigned = True

    if assigned == False:
        list.append([])
        list[len(list) - 1].append(frame)
```

Analogicky sa napĺňa aj ARP list, len as nekontrolujú porty, ale *flagy*. Úlohou tejto kontroly je zabránenie pripojenia rámcov na už ukončenú komunikáciu.

def check_start_tcp(list) a **def check_end_tcp(list):**

V štvrtej úlohe od a až po f máme za úlohu zistiť či ej komunikácia ukončená neukončená alebo ani jedno. Zisťujeme to podľa vracajúcich hodnôt vyššie uvedených funkcií. Ak sú neprázdne, tak v danom pod-liste (všetky rámce komunikačného páru [zhodujúce IP a porty]). Ako začiatok hľadáme 3-way-handshake:

```
    for i in range(2, len(list)-1):
        if (list[i-2].flag[1] == '1' and list[i-1].flag[1] == '1' and
    list[i-1].flag[4] == '1' and list[i].flag[4] == '1' ):
            start = i-2
            continue
```

Koniec má tri spôsoby:

```
    for i in range(3, len(list)):

            # RST
        if list[i].flag[2] == '1':
            end = list[i]
            continue

        if i > 4 :
                # FIN ACK, ACK, FIN ACK, ACK
```

```
            if (list[i-3].flag[0] == '1' and list[i-3].flag[4] == '1' and
    list[i-2].flag[4] == '1' and  list[i-1].flag[0] == '1' and list[i-
    1].flag[4] == '1' and list[i].flag[4] == '1'):

                end = list[i - 3]
                continue

                # FIN, FIN ACK, ACK
            if (list[i - 2].flag[0] == '1' and list[i - 1].flag[0] == '1'
    and list[i - 1].flag[4] == '1' and list[i].flag[4] == '1'):
                end = list[i - 3]
                continue
```

Porovnávať môžeme bit po bite vďaka tomu, že flag je string otočeného čísla v binárnej sústave.
(0h11 -> int(17) -> 0b10001 -> 10001b0 -> "1000100")

def check_tftp_communication(frame):

Keďže nie všetka TFTP komunikácia sa rozoznáva podľa portu, ale len prvý rámec, je potrebné odsledovať aj tie nasledujúce, na to slúži táto funkcia.

Každý prvý TFTP rámec uložíme do pod-listu listu s TFTP komunikáciou. Funkcia kontroluje zhodu IP adries a zhodu nultého zdrojového portu s zdrojovým alebo cieľovým portom rámca, ktorý kontrolujeme.

```
for i in range(len(TFTP_l)):
    if set([TFTP_l[i][0].s_ip, TFTP_l[i][0].d_ip ]) ==
set([frame.d_ip, frame.s_ip]) and (TFTP_l[i][0].s_port ==
frame.d_port or TFTP_l[i][0].s_port == frame.s_port):
        TFTP_l[i].append(frame)
        return True
```

def four_g(out_file):

Výpis z úlohy štyri je analogický pre každý bod úlohy. Pri potrebe dodatočného výpisu (*Target MAC* pri ARP, *Complete or Incomplete* pri TCP[4.)a-f]) sú funkcie poupravené.

Zo zadania vieme, že vypisujeme len prvých a posledných 10 rámcov komunikácie (ak má pod 20 rámcov, celú), toto zabezpečuje podmienka zvýraznená žltou farbou.

```
if len(TFTP_l)>0:
    out_file.write("TFTP".center(80, "_"))

    for i in range(len(TFTP_l)):
        out_file.write("\n")
        out_file.write("Communication: {}".center(80, "_").format(i +1))

        for j in range(len(TFTP_l[i])):
            if j < 10 or j > len(TFTP_l[i]) - 11:
                out_file.write(four_write(TFTP_l[i][j], "TFTP"))

else:
    out_file.write("\nNo TFTP communication\n")
```

four_write je funncia, ktorá vráti string rámca, ktorý sa ako celok zapíše.

def main():

Main funkcia je opísaná v časti **Blokový návrh (koncepcia) fungovania riešenia.** Obsahuje cyklus, ktorý čaká na vstup, podľa vstupu program pracuje.

Ostatné časti programu sú bližšie okomentované v zdrojovom kóde.

Implementačné prostredie

Implementačným prostredím bol program PyCharm. Rozhodol som sa zadanie robiť v pythone, tak PyCharm bola jasná voľba. Vstavaný debugger, kontrola nad programom, inteligentné dopĺňanie a mnoho ďalších vecí pomáhalo s programovaním keďže toto boli moje prvé skúsenosti s pythonom.