Adam Miškove

Počítačové a komunikačné siete

Zadanie 1: Analyzátor sieťovej komunikácie
FIIT STU

Cvičenie: Štvrtok 10:00-11:40, Košťál

Zadanie 1: **Analyzátor sieťovej komunikácie**

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) **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 byťschopný 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 súborsa 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 byťschopný 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

Koncept programu:

Pre začiatok môjho riešenia zadania je potrebné vypísať analyzované údaje celého otvoreného súboru, a preto som zvolil použiť vnímanie hlavných funkcií v zmysle jedného rámca/paketu.

Tieto funkcie riadi funkcia **fullLoop()**, ktorá inicializuje program a vo for cykle volá funkciu **onePacketLoop()**, ktorá sa stará o analýzu jedného, konkrétneho rámca.

Táto funkcia **onePacketLoop()** si otvorí daný rámec a prevedie ho do bytes() a potom do str(), čím sa nám viac sprístupňujú jeho informácie. Tieto informácie čiastočne analyzuje, konkrétne zistí (pre nás) staticky dané informácie, ako dĺžka rámca a MAC adresy. Ich výpisy sú vykonané na miestach, ktoré udáva naše zadanie.

Môžeme si všimnúť, že tieto funkcie si posúvajú isté premenné a listy, pričom niektoré slúžia ako poskytnutie informácií na ich analýzu(**strpacket**), výpis na správnom mieste v priebehu programu(mac adresy) alebo "komunikáciu"/interakciu rôznych rámcov medzi sebou(potrebné pri ARP komunikácii, tftp komunikácii, a určovaní polôh rámcov s požadovanými portmi/protokolmi zo zadania).

Funkcia, ktorá je volaná z funkcie **onePacketLoop()** pre hlbšiu analýzu a riadenie analytického chodu programu je **analyzeFrame().** Táto funkcia organizuje a riadi všetku ďalšiu analýzu našich paketov.

Pre začiatok, táto funkcia získa z analyzovaného paketu hodnotu len/type, podľa ktorej zisťuje, či sa jedná o IEEE 802.3 alebo Ethernet II.
Podľa tejto informácie sa rozhoduje, čo má robiť ďalej. Tento jav rozhodnutia podľa získanej informácie o rámci sa opakuje v mnohých miestach tohto programu.

Ak je to Ethernet II, musí zistiť protokol na ďalšej vrstve, a podľa zadania tieto informácie vypíše. Ak je tento protokol IPv4 alebo ARP, musí riešiť ďalšie veci, ako napríklad protokoly pod IPv4, pričom pre TCP, UDP, a ICMP protokoly sa analýza nekončí. Pre ARP protokol treba na základe zadania zistiť typ operácie, teda Request alebo Reply, a spárovať ich medzi sebou, pričom requesty alebo reply rovnakej povahy, ale bez páru (requesty bez reply-ov, reply-e bez requestov) je potrebné rozdeliť do samostatných komunikácií, ako je v zadaní napísané.
Pre dané protokoly TCP, UDP a ICMP pod protokolom IPv4 je potrebné ďalej analyzovať ich porty, alebo v prípade ICMP treba analyzovať jeho typ a kód. Súčasťou tejto analýzy je prirodzene nevynechateľnou časťou zistiť niektoré vysoko dôležité informácie, ako napríklad internet header length IPv4 protokolu, ako aj adresy a porty všetkých spomenutých protokolov.

Pri IEEE 802.3 program zisťuje, či sa jedná o LLC, LLC+SNAP, alebo raw IPX protokol. V prípade čisto LLC protokolu skúmam a analyzujem jeho DSAP/SSAP, pričom zisťujem protokol na ďalšej vrstve.

Po skončení analýzy všetkých rámcov je vypísaná štatistika zdrojových IP adries podľa kritérií, ktoré boli spomenuté v zadaní.
Takisto je možné vypísať konkrétny typ protokolu z úlohy číslo 4, teda “http”, “https ssl”, “telnet”, “ssh”, “ftp-data”, “ftp-control”, “tftp”, “ICMP”. Výskyty týchto protokolov sa zapisujú do **listFilter** počas vykonávania funkcie **analyzeFrame()**.

Príklad štruktúry externých súborov pre určenie protokolov a portov:

Pre určenie konkrétnych protokolov a portov z dokumentu “Analyza_ramcov_hlavicky”, ktorý nám bol poskytnutý na dokumentovom serveri, som v ňom spomenuté hodnoty pre SAPy, Ethertypy, IPv4, TCP a UDP protokoly uložil v externom súbore. Pre ICMP som uložil hodnoty v druhom externom súbore, pričom sú ohľadom na výskyt typu a kódu rozdelené rovnakým princípom ako predošlé.

Súbor je vo formáte:
#vyhľadávaný protokol(pod ktorým sa vyhľadáva)
vyhľadávaná hodnota
názov k príslušnej hodnote
vyhľadávaná hodnota
názov k príslušnej hodnote
vyhľadávaná hodnota
názov k príslušnej hodnote
vyhľadávaná hodnota
názov k príslušnej hodnote
(a tak ďalej, až do konca daného protokolu, potom nasleduje opakovanie rovnakého princípu, pre ďalší protokol, pričom po konci posledného protokolu je riadok EXIT).

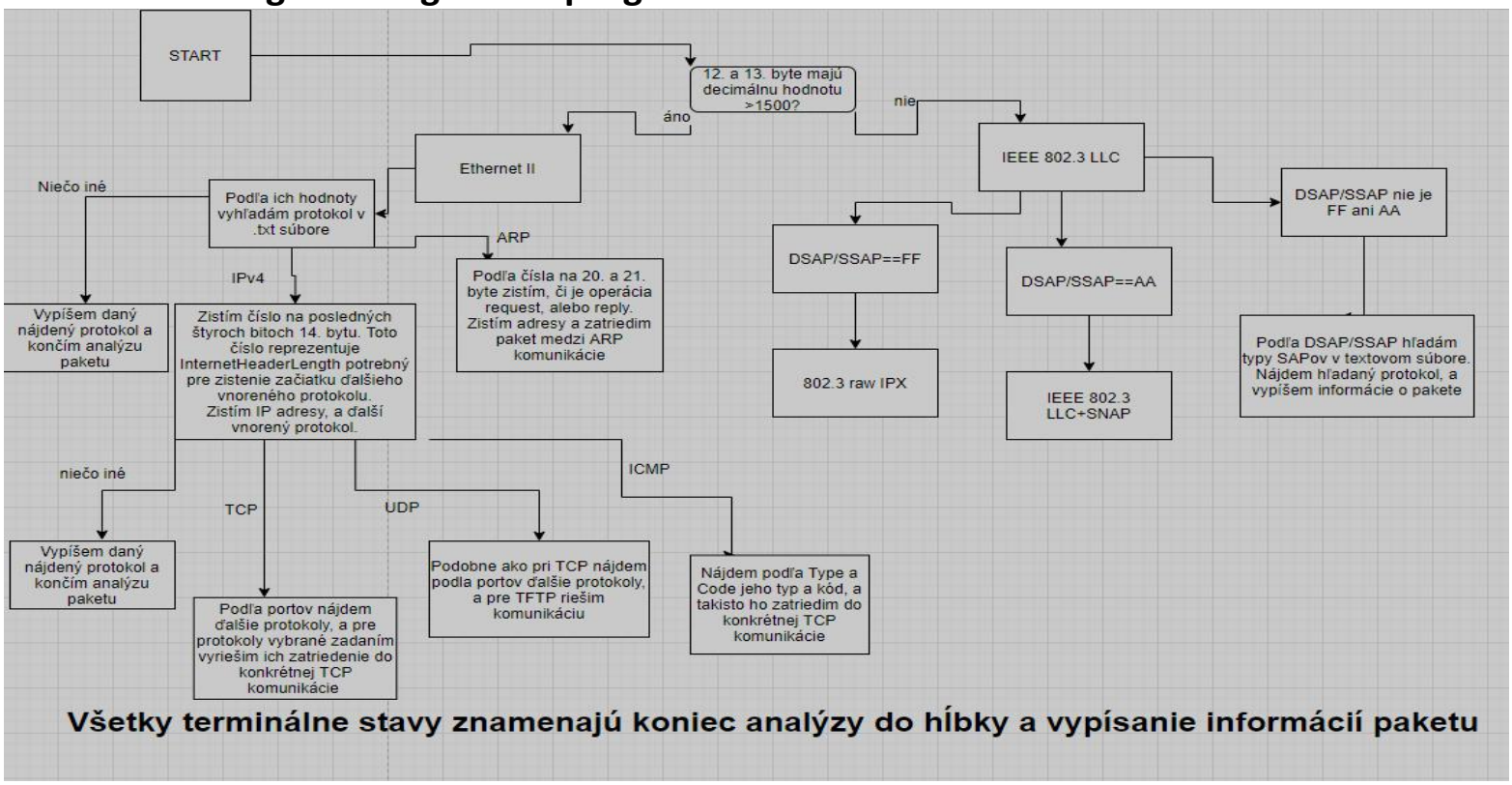
Konkrétny príklad:
#TCP
7
echo
19
chargen
20
ftp-data
21
ftp-control
22
ssh
23
telnet
25
smtp
53
domain
79
finger
80
http
110
pop3
111
sunrpc
119
nntp
139
netbios-ssn
143
imap
179
bgp
389
ldap
443
https ssl
445
microsoft-ds
1080
socks
#UDP
7

echo
19
chargen
37
time
53
domain
67
bootps DHCP
68
bootpc DHCP
69
tftp
137
netbios-ns
138
netbios-dgm
161
snmp
162
snmp-trap
500
isakmp
514
syslog
520
rip
33434
traceroute
EXIT

Opísané používateľské rozhranie:

Pre tento program som sa rozhodol využiť klasické, jednoduché konzolové textové rozhranie, pričom dá program používateľovi možnosti vykonania akcií, napríklad vypísať rámce s konkrétnym protokolom z úlohy 4.
Taktiež si používateľ vyberá súbor s paketmi na otvorenie.
Program sa začína kompletným prejdením všetkých paketov a ich výpisom pre úlohy 1-3.
Po tomto dostáva užívateľ možnosť vyfiltrovať konkrétny výpis pre úlohu 4, a program skončí.

Diagram fungovania programu:



Objasnenie funkcií a fungovania programu/spôsobu analýzy rámcov a protokolov:

Pomocné funkcie:

```
def getFrameLengthAPI(pkt):
    return int((len(pkt)-3)/2)
def getFrameLengthMedium(pkt):
    if((len(pkt)-3)/2<=60):
        return 64
    else:
        return int(4+(len(pkt)-3)/2)
def getDestMAC(strpkt):
    return strpkt[2:14]
def getSrcMAC(strpkt):
    return strpkt[14:26]
def getSrcIP(strpkt,start):   #strpkt[54:62]
    ip = ""
    for i in range(0, 4):
        ip += str(hexToDec(strpkt[(start+(i*2)):(start+2+(i*2))]))
        if i < 3:
            ip += "."
    return ip
def getDestIP(strpkt,start):   #strpkt[62:70]
    ip = ""
    for i in range(0, 4):
        ip += str(hexToDec(strpkt[(start+(i*2)):(start+2+(i*2))]))
        if i < 3:
            ip += "."
    return ip
def printMACAddresses(dest,src):
    if(len(dest)!=len(src)):
        return "error1"
    print("Zdrojová MAC adresa: ",end="")
    for x in range(0,len(src), 2):
        print("{}{} ".format(src[x], src[x + 1]), end="")
    print("")
    print("Cieľová MAC adresa: ", end="")
    for x in range(0, len(dest), 2):
        print("{}{} ".format(dest[x], dest[x + 1]), end="")
    print("")
def printIPv4Addresses(src,dest):
    print("zdrojová IP adresa: {}".format(src))
    print("cieľová IP adresa: {}".format(dest))
```

Prvé dve funkcie zistia a vypočítajú požadované dĺžky rámca.

Funkcie **getDestMAC** a **getSrcMAC** vrátia hodnoty MAC adries, ktoré sa nachádzajú na statickom mieste a sú potrebné pri každom rámci.

Podobne fungujú funkcie **getSrcIP** a **getDestIP**, s rozdielom, že IP adresy nachádzajú podľa argumentu **start.**

Ďalej nasledujú funkcie slúžiace na výpis informácií a ich formátovanie.

```
def printMACAddresses(dest,src):
    if(len(dest)!=len(src)):
        return "error1"
    print("Zdrojová MAC adresa: ",end="")
    for x in range(0,len(src), 2):
        print("{}{} ".format(src[x], src[x + 1]), end="")
    print("")
    print("Cieľová MAC adresa: ", end="")
    for x in range(0, len(dest), 2):
        print("{}{} ".format(dest[x], dest[x + 1]), end="")
    print("")
def printIPv4Addresses(src,dest):
    print("zdrojová IP adresa: {}".format(src))
    print("cieľová IP adresa: {}".format(dest))
def printFrameLengths(first,second):
    print("dĺžka rámca poskytnutá pcap API - {} B".format(first))
    print("dĺžka rámca prenášaného po médiu - {} B".format(second))
def printIPCount(listIP):
    if(len(listIP)==0):
        print("Nenašli sa žiadne IPv4 pakety!")
        return
    print("IP adresy vysielajúcich uzlov:")
    max=0
    idx = 0
    for i in range(0,len(listIP)):
        if(max<listIP[i][1]):
            max=listIP[i][1]
            idx=i
        print(listIP[i][0])
    print("\nAdresa uzla s najväčším počtom odoslaných paketov:\n{}       {} paketov\n\n".format(listIP[idx][0],listIP[idx][1]))
def printBytes(strpkt):
    n = 0
    for x in range(0, len(strpkt)):
        if(x==0 or x==1 or x==len(strpkt)-1):
            continue
        else:
            n += 1
            print(strpkt[x], end="")
            if(n%32 == 0):
                print("")
            elif(n%16 == 0):
```

Pre tento program sú dôležité aj elementárne funkcie, slúžiace na prevod čísel z hexadecimálnej do decimálnej sústavy.

```
def htdChar(c):#assuming its hexadec
    if c.isalpha() and len(c) == 1:
        if c.isupper():
            return int((ord(c)) - 55)
        else:
            return int((ord(c)) - 87)
    elif len(c) == 1:
        return int(c)
def hexToDec(str):
    num = 0
    for i in range(len(str)-1,0-1,-1):
        num += int(htdChar(str[i])*(16**(1-i)))
    return num
```

Funkcia **htdChar** sa stará o premenu písmenok v hexadecimálnom čísle na decimálne.

Funkcia **hexToDec** pracuje spolu s funkciou **htdChar** pre prevod hexadec. čísla v stringu do intu, volá sa keď potrebujem z paketu získať konkrétne číslo, napríklad čísla protokolov/portov.

```
def getProtocol(n, typeStr):
    if(typeStr == "Ethernet II"):
        nestedProtocol = findInTxt("#ETHERTYPE\n", n)
        return nestedProtocol
    elif(typeStr == "IEEE 802.3 LLC"):
        nestedProtocol = findInTxt("#SAP\n", n)
        return nestedProtocol
```

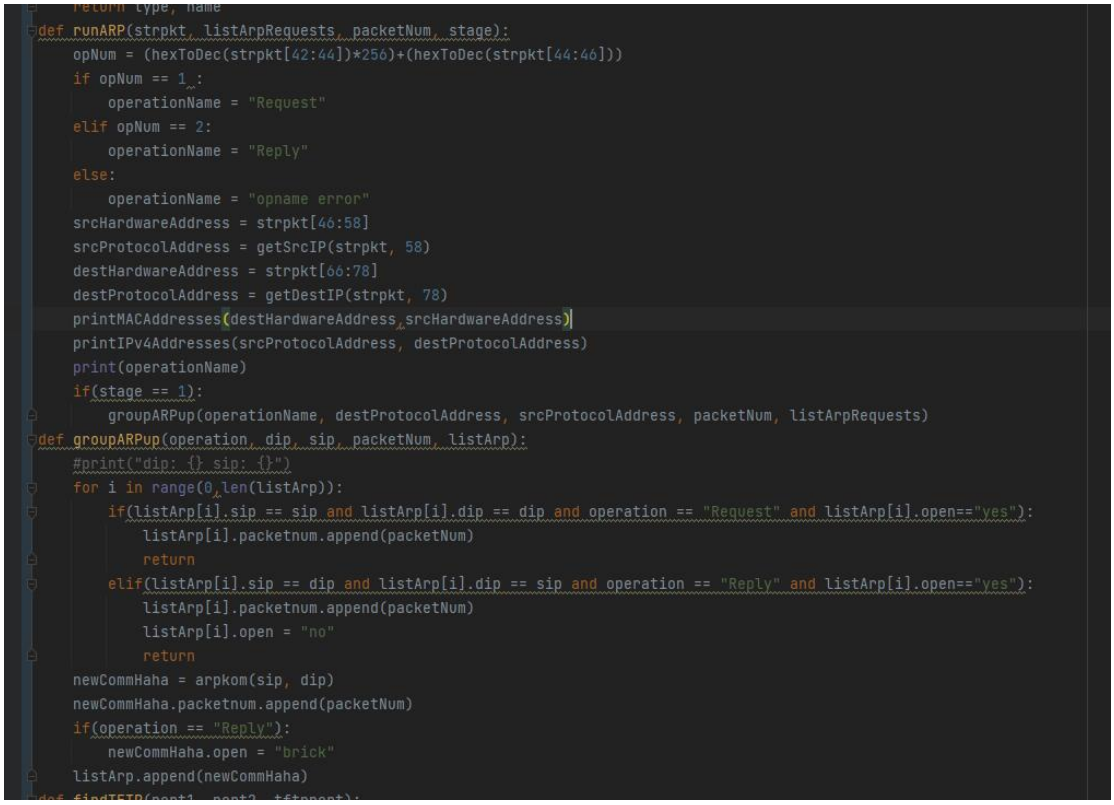
Funkcia **getProtocol** vráti prvý analyzovaný protokol pod Ethernet 2/IEEE 802.3 LLC-> IPv4, ARP, SPT a podobne.... (protokoly z Ethertypov a SAPov)

```
def findInTxt(categoryStr, num):
    f = open('C:/Users/adam/Desktop/FIIT/treti rocnik/__PKS/externysubor.txt', "r")
    numStr = str(num)
    numStr += "\n"
    for i in range(0, 1000):
        if (f.readline() == categoryStr):
            for y in range(0, 1000):
                cmp = f.readline()
                if (cmp == "EXIT" or cmp[0] == "#"):
                    #print("-----------------protocol not found!!!----------------------")
                    f.close()
                    return "NotFound"
                if (cmp == numStr):
                    nestedProtocol = f.readline().replace("\n", "")
                    #print("found!!!!!!!!!!")
                    #print(nestedProtocol)
                    f.close()
                    return nestedProtocol
```

Funkcia **findInTxt** slúži na nájdenie hľadaného protokolu v jeho príslušnom externom súbore. Štruktúra externého súboru už bola objasnená vyššie.
Existuje aj podobná funkcia **findInTxtICMP,** ktorá funguje podobne, no pre ICMP a v inom súbore.

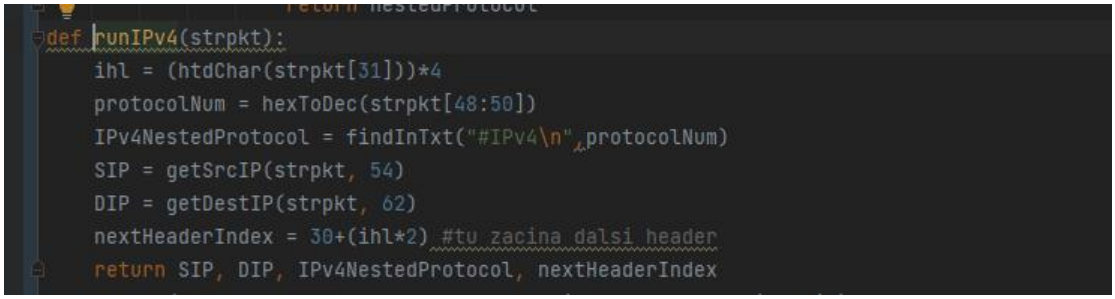
Jedna z najhlavnejších funkcií tohto programu, **analyzeFrame,** sa stará o takmer celý proces analýzy paketov. Je rozdelená do dvoch hlavných častí, prvá pre Ethernet II a druhá pre IEEE 802.3.

Tieto časti fungujú veľmi špecificky pre ich povinnosti. Analýza oboch častí je odlišná a je ju potrebne ovládať rozdielne.



```
def runARP(strpkt, listArpRequests, packetNum, stage):
    opNum = (hexToDec(strpkt[42:44])*256)+(hexToDec(strpkt[44:46]))
    if opNum == 1:
        operationName = "Request"
    elif opNum == 2:
        operationName = "Reply"
    else:
        operationName = "opname error"
    srcHardwareAddress = strpkt[46:58]
    srcProtocolAddress = getSrcIP(strpkt, 58)
    destHardwareAddress = strpkt[66:78]
    destProtocolAddress = getDestIP(strpkt, 78)
    printMACAddresses(destHardwareAddress, srcHardwareAddress)
    printIPv4Addresses(srcProtocolAddress, destProtocolAddress)
    print(operationName)
    if(stage == 1):
        groupARPup(operationName, destProtocolAddress, srcProtocolAddress, packetNum, listArpRequests)
def groupARPup(operation, dip, sip, packetNum, listArp):
    #print("dip: {} sip: {}")
    for i in range(0,len(listArp)):
        if(listArp[i].sip == sip and listArp[i].dip == dip and operation == "Request" and listArp[i].open=="yes"):
            listArp[i].packetnum.append(packetNum)
            return
        elif(listArp[i].sip == dip and listArp[i].dip == sip and operation == "Reply" and listArp[i].open=="yes"):
            listArp[i].packetnum.append(packetNum)
            listArp[i].open = "no"
            return
    newCommHaha = arpkom(sip, dip)
    newCommHaha.packetnum.append(packetNum)
    if(operation == "Reply"):
        newCommHaha.open = "brick"
    listArp.append(newCommHaha)
```

Pre Ethernet II musí zistiť, či to je IPv4, alebo ARP. Pre ARP sa volá funkcia **runARP,** ktorá analyzuje a zatriedi ARP pakety do ARP komunikácií.
Analýza začne zistením typu operácie podľa poľa Operation v pakete(1= request, 2= reply) a ďalej zistí IP a MAC adresy podľa polí Hardware Address a Protocol Address.
Vo funkcii **groupARPup** zgrupujem pakety do komunikácií podľa ich IP adries, a v prípade začatia novej komunikácie danú komunikáciu vytvorím.



```
def runIPv4(strpkt):
    ihl = (htdChar(strpkt[31]))*4
    protocolNum = hexToDec(strpkt[48:50])
    IPv4NestedProtocol = findInTxt("#IPv4\n", protocolNum)
    SIP = getSrcIP(strpkt, 54)
    DIP = getDestIP(strpkt, 62)
    nextHeaderIndex = 30+(ihl*2) #tu zacina dalsi header
    return SIP, DIP, IPv4NestedProtocol, nextHeaderIndex
```

Pre protokol IPv4 sa volá funkcia **runIPv4**, ktorá slúži na jeho analýzu.
Podľa ťaháku z dokumentového serveru je spravené hľadanie Internet Header Length-u, ktorý diktuje, aký dlhý tento IP header je. S touto informáciou viem zistiť, kde sa začína ďalší protokol, ak taký je/ak ho potrebujem neskôr riešiť.
Taktiež hľadám IPv4 adresy a názov vnoreného protokolu.

V rámci IPv4 protokolov treba ďalej analyzovať TCP, UDP a ICMP protokoly.

```
if(first_nested_protocol == "IPv4"):
    SIP, DIP, second_nested_protocol, nextHeaderIndex = runIPv4(strpkt)
    checkDIP(listIP,SIP)
    printIPv4Addresses(SIP, DIP)                                #print IP adries
    print(second_nested_protocol)                               #print vnoreneho protokolu
    if(second_nested_protocol == "TCP"):
        tcpPort = runTCP(strpkt, nextHeaderIndex, packetNum, listTCP, stage, DIP, SIP)
        if (tcpPort == "http" or tcpPort == "https ssl" or tcpPort == "telnet" or tcpPort == "ssh" or tcpPort == "ftp-data" or tcpPort == "ftp-control"):
            listFilter[packetNum-1] = tcpPort
    elif(second_nested_protocol == "UDP"):
        udpPort, tftpport = runUDP(strpkt, nextHeaderIndex, tftpport)
        if(tftpport != tftpportstart):
            tftpNum+=1
            if(stage==1):
                newTFTPComm = tftpkom()
                listTFTP.append(newTFTPComm)
        if(udpPort == "tftp"):
            if(stage == 1):
                listTFTP[tftpNum-1].packetnum.append(packetNum)
            print("Číslo tftp komunikácie: {}".format(tftpNum))
            listFilter[packetNum - 1] = udpPort
    elif(second_nested_protocol == "ICMP"):
        listFilter[packetNum - 1] = second_nested_protocol
        ICMPtype, ICMPname = runICMP(strpkt,nextHeaderIndex)
        print("ICMP type: {}".format(ICMPtype))
        if(ICMPname != "NotFound"):
            print("ICMP name: {}".format(ICMPname))
```

Pre TCP protokoly volám funkciu **runTCP**, ktorá sa postará o analýzu TCP protokolov a ich portov, a volá ďalšie funkcie, ktoré ovládajú TCP komunikácie medzi paketmi a ich vytváranie/pridávanie.

```
def runTCP(strpkt, startIndex, packetNum, listTCP, stage, dip, sip):
    SPortNum = (hexToDec(strpkt[startIndex:startIndex + 2]) * 256) + hexToDec(strpkt[startIndex + 2:startIndex + 4])
    DPortNum = (hexToDec(strpkt[startIndex+4:startIndex + 6]) * 256) + hexToDec(strpkt[startIndex + 6:startIndex + 8])
    SPort = findInTxt("#TCP\n", SPortNum)
    DPort = findInTxt("#TCP\n", DPortNum)
    realPort = 0
    if (SPort == "NotFound" and DPort != "NotFound"):
        print(DPort)
        realPort = DPort
    elif(SPort != "NotFound" and DPort == "NotFound"):
        print(SPort)
        realPort = SPort
    elif (SPort != "NotFound" and DPort != "NotFound" and DPort == SPort):
        print(DPort)
        realPort = DPort
    print("zdrojový port: {}".format(SPortNum))
    print("cieľový port: {}".format(DPortNum))
    if(stage == 1):
        TCPflag = findTCPFlag(strpkt, startIndex + 26)
        checkTCPcomm(packetNum, dip, sip, DPortNum, SPortNum, listTCP, TCPflag, realPort)
    return realPort
```

Podľa ťaháku s TCP Headerom klasicky zistí porty, a získava TCP flagy z funkcie **findTCPFlag.**

Pre ovládanie TCP komunikácie sa volá funkcia **checkTCPcomm,** ktorá manuálne hľadá three-way-handshake, pakety data streamu, a ukončenie komunikácie rôznymi spôsobmi(finy, finacky, resety).

Pre UDP protokoly volám funkciu **runUDP**, ktorá sa postará o analýzu daných paketov a ich komunikáciu.

```
def runUDP(strpkt, startIndex, tftpport):
    SPortNum = (hexToDec(strpkt[startIndex:startIndex + 2]) * 256) + hexToDec(strpkt[startIndex + 2:startIndex + 4])
    DPortNum = (hexToDec(strpkt[startIndex + 4:startIndex + 6]) * 256) + hexToDec(strpkt[startIndex + 6:startIndex + 8])
    SPort = findInTxt("#UDP\n", SPortNum)
    DPort = findInTxt("#UDP\n", DPortNum)
    realPort = 0
    checktftp, tftpport = findTFTP(SPortNum, DPortNum,tftpport)
    if(checktftp == "yestftp"):                                    #ak najdem tftp port tak sa to musi handlovat jednotli
        realPort = "tftp"
        print(realPort)
        print("zdrojový port: {}".format(SPortNum))
        print("cieľový port: {}".format(DPortNum))
        return realPort, tftpport
    if (SPort == "NotFound" and DPort != "NotFound"):
        print(DPort)
        realPort = DPort
    elif(SPort != "NotFound" and DPort == "NotFound"):
        print(SPort)
        realPort = SPort
    elif (SPort != "NotFound" and DPort != "NotFound" and DPort == SPort):
        print(DPort)
        realPort = DPort
    print("zdrojový port: {}".format(SPortNum))
    print("cieľový port: {}".format(DPortNum))
    return realPort, tftpport
```

Znova podľa ťaháku z dokumentového serveru zisťujem UDP porty a rozhodujem sa, o aké typy ide, napríklad pri TFTP porte 69 sa začína TFTP komunikácia, pre ktorú potrebujem funkciu **findTFTP**, ktorá sa stará o sledovanie danej komunikácie, jej vývin a zmenu TFTP portov.

```
def findTFTP(port1, port2, tftpport):
    if(port1 == 69 or port2 == 69):
        if(port1!=69):
            tftpport = port1
            return "yestftp", tftpport
        if(port2!=69):
            tftpport = port2
            return "yestftp", tftpport
    else:
        if((port1 == tftpport or port2 == tftpport) and tftpport!=-1):
            return "yestftp", tftpport
        else:
            return "notftp", tftpport
```

Posledný typ analyzovaných protokolov pod IPv4 je ICMP, pričom tento protokol sa analyzuje podobne ako predošlé. Zavolá sa jeho funkcia **runICMP**, ktorá z paketu vytiahne zakódovaný Type a Code/Name, ktoré v externom súbore cez funkciu **findInTxtICMP** nájde a pridelí k ním ich mená.

```
def runICMP(strpkt, startIndex):
    typeNum = hexToDec(strpkt[startIndex:startIndex+2])
    nameNum = hexToDec(strpkt[startIndex+2:startIndex+4])
    type = findInTxtICMP("#TYPE\n", typeNum)
    if(typeNum == 3):
        name = findInTxtICMP("#NAME3\n", nameNum)
    elif(typeNum == 5):
        name = findInTxtICMP("#NAME5\n", nameNum)
    elif (typeNum == 11):
        name = findInTxtICMP("#NAME11\n", nameNum)
    elif (typeNum == 12):
        name = findInTxtICMP("#NAME12\n", nameNum)
    else:
        name = "NotFound"
    return type, name
```

Pre IEEE 802.3 máme riešiť iba LLC, totižto RAW má IPX, a LLC SNAP nie je náplňou tohto zadania. Tieto dve iba jednoducho vypíšem a protokol pod LLC sa s pomocou DSAP/SSAP hodnôt hľadá v SAPoch a vypíše.

```
        runARP(strpkt, listArpRequests, packetNum, stage)
else:
    frameType = "IEEE 802.3"
    ieeeDecider = hexToDec(strpkt[30:32])
    if(ieeeDecider == 255):
        frameType+= " raw IPX"
        print(frameType)                                    #print typu ramca
        printMACAddresses(dmac, smac)                       #print MAC adries
    elif(ieeeDecider == 170):
        frameType+= " LLC SNAP"
        print(frameType)                                    #print typu ramca
        printMACAddresses(dmac, smac)                       #print MAC adries
    else:
        frameType+= " LLC"
        print(frameType)                                    #print typu ramca
        printMACAddresses(dmac, smac)                       #print MAC adries
        first_nested_protocol = getProtocol(ieeeDecider, frameType)
        print(first_nested_protocol)                        #print vnoreneho protokolu
        # print(first_nested_protocol)
```

Ostatné funkcie fungujú ako filter výpisu pri štvrtom bode zadania, a asi nie je veľmi dôležité ich opisovať.

Voľba implementačného prostredia:

Moje riešenie tohto zadania je naprogramované v programovacom jazyku **Python**, pričom som používal IDE **PyCharm** od JetBrains.