

PKS  
Zadanie č.1  
Analyzátor sieťovej komunikácie  
2023-24

## Table of Contents

<b>Zadanie .....</b>	<b>2</b>
<b>Implementácia .....</b>	<b>3</b>
<b>Používanie programu.....</b>	<b>7</b>
<b>.....</b>	<b>7</b>
<b>Zhodnotenie .....</b>	<b>7</b>

## Zadanie

Navrhnuť a implementovať programový analyzátor Ethernet siete, ktorý analyzuje komunikácie v sieti zaznamenané v .pcap súbore a poskytuje informácie o rámcoch:

1. Typ rámca
2. Dĺžka rámca poskytnutá pcap API a dĺžka rámca prenášaného po médiu
3. Zdrojová a cieľová MAC adresa
4. Zdrojová a cieľová IP adresa
5. Vnorený protokol v hlavičke rámca
6. Vnorený protokol v IPv4
7. Zdrojový a cieľový port
8. Aplikačný protokol
9. Iné informácie špecifické pre rôzne typy rámcov a jednotlivé protokoly

Výstup musí byť v .yaml súbore. 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.** Napríklad použitie funkcionality *libpcap* na priamy výpis konkrétnych polí rámca (napr. `ih->saddr`) bude mať za následok nulové hodnotenie celého zadania.

Čísla protokolov v rámci Ethernet II (pole `Ethertype`), v IP pakete (pole `Protocol`) a čísla portov pre transportné protokoly musia byť **načítané z jedného alebo viacerých externých textových súborov.**

## Implementácia

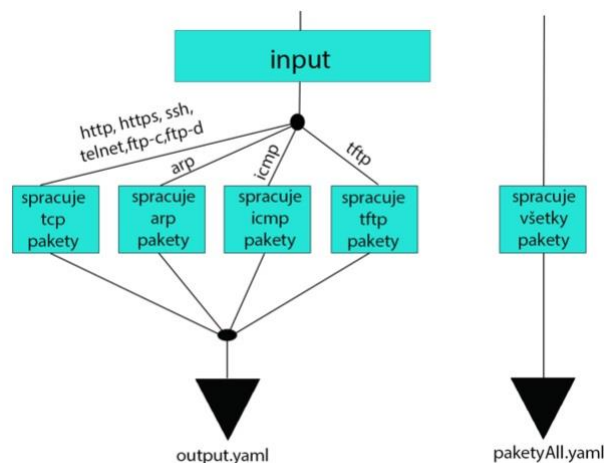
Zadanie som robil v jazyku Python, verzii 3.11 v prostredí JetBrains PyCharm 2023.2.2, na prácu s .pcap súbormi som použil knižnicu scapy a na prácu s .yaml súbormi som použil knižnicu ruamel.yaml.

Protokoly v rámci Ethernet II, v IP pakete a čísla portov pre transportné protokoly som si uložil do slovníkov v externom súbore MyDictionaries.py.

Môj program má dva výstupy:

1. paketyAll.yaml – všeobecná analýza rámcov (úlohy 1, 2 a 3)
2. output.yaml – analýza komunikácie podľa filtra (úloha 4)

### Diagram spracovania .pcap súborov



Pre každý paket program zistí typ paketu (premenná EternetType) . Ak rámec nie je typu Ethernet II, podľa premennej TypLLC určí či je rámec typu IEEE 802.3 LLC & SNAP, IEEE 802.3 RAW alebo IEEE 802.3 LLC.

Ďalej zisťuje zdrojovú a cieľovú MAC adresu, dĺžku rámca poskytnutú pcap API aj manuálne vypočíta dĺžku rámca – premenná „len\_frame\_medium“

```
if int(EthernetType, 16) > 1500: #ak je
    packet_info["frame_type"] = "Ethernet II"
    if EthernetType in ETHERtypes:
        packet_info["ether_type"] = ETHERtypes[EthernetType]
    else:
        packet_info["ether_type"] = "unknown"
    if packet_info["ether_type"] == "IPv4":
        SIP = ""
        for i in range(52, 60, 2): # zapise Source IP
            part = f"{int(hex_data[i:i + 2], 16)}"
            SIP = SIP + part + "."
        SIP = SIP[:-1]
        DIP = ""
        for i in range(60, 68, 2): # zapise Destination IP
            part = f"{int(hex_data[i:i + 2], 16)}"
            DIP = DIP + part + "."
        DIP = DIP[:-1]
        packet_info["src_ip"] = SIP
        IpStatistics[SIP] += 1
        packet_info["dst_ip"] = DIP
```

Zapíše jednotlivé protokoly na transportnej vrstve a ku nim prislúchajúce porty, tieto informácie tiež ťahá z externého súboru.

```
elif packet_info["ether_type"] == "IPv6":
    SIP = ""
    for i in range(44, 76, 4): #zapise Source IP
        part = f"{hex_data[i:i + 4]}"
        if part[0] == "0":
            part = part[1:]
        SIP = SIP + part + ":"
    SIP = SIP[:-1]
    SIP = SIP.replace(_old: "000:", _new: "")
    SIP = SIP[0:4] + ":" + SIP[4:]
    DIP = ""
    for i in range(76, 108, 4): #zapise Destination IP
        part = f"{hex_data[i:i + 4]}"
        if part[0] == "0":
            part = part[1:]
        DIP = DIP + part + ":"
    DIP = DIP[:-1]
    DIP = DIP.replace(_old: "000:", _new: "")
    DIP = DIP[0:4] + ":" + DIP[4:]
    packet_info["src_ip"] = SIP
    packet_info["dst_ip"] = DIP
```

```
for packet in packets: #na zaciatku prejde vsetky
    hex_data = packet.build().hex() #vyrobi aj ip
    EthernetType = hex_data[24:28]
    TypLLC = f"{hex_data[28:30]}"
    DMAC = ""
    for i in range(0, 12, 2): #zapise Destination MAC
        DMAC = DMAC + hex_data[i:i+2] + ":"
    DMAC = DMAC[:-1]
    SMAC = ""
    for i in range(12, 24, 2): #zapise Source MAC
        SMAC = SMAC + hex_data[i:i+2] + ":"
    SMAC = SMAC[:-1]
    packet_info = {"frame_number": counter} #vytvori
    packet_info["len_frame_pcap"] = len(packet.payload) #z
    manual_len = 0
    for layer in packet:
        manual_len += len(layer)
    if manual_len < 64:
        packet_info["len_frame_medium"] = 64
    else:
        packet_info["len_frame_medium"] = manual_len + 4 #
```

Ak je rámec typu Ethernet II, zapíše vnorený protokol cez slovník z externého súboru. Ďalej zapíše IP adresy rámca.

```
if IPtypes[hex_data[46:48]] == "TCP":
    packet_info["protocol"] = IPtypes[hex_data[46:48]]
    if f"{int(hex_data[68:72], 16)}" in portsTCP:
        packet_info["src_port"] = portsTCP[f"{int(hex_data[68:72], 16)}"]
    else:
        packet_info["src_port"] = int(hex_data[68:72], 16)
    if f"{int(hex_data[72:76], 16)}" in portsTCP:
        packet_info["dst_port"] = portsTCP[f"{int(hex_data[72:76], 16)}"]
    else:
        packet_info["dst_port"] = int(hex_data[72:76], 16)
elif IPtypes[hex_data[46:48]] == "UDP":
    packet_info["protocol"] = IPtypes[hex_data[46:48]]
    if f"{int(hex_data[68:72], 16)}" in portsUDP:
        packet_info["src_port"] = portsUDP[f"{int(hex_data[68:72], 16)}"]
    else:
        packet_info["src_port"] = int(hex_data[68:72], 16)
    if f"{int(hex_data[72:76], 16)}" in portsUDP:
        packet_info["dst_port"] = portsUDP[f"{int(hex_data[72:76], 16)}"]
    else:
        packet_info["dst_port"] = int(hex_data[72:76], 16)
else:
    packet_info["protocol"] = IPtypes[hex_data[46:48]]
```

Ak má rámec v hlavičke protokol IPv6, zápis IP adresy je trochu iný.

```
for pismeno in hex_data:      #podeli hexa gulas na 2B casti a pri
    hexDataFinal += pismeno
    if counter2 % 2 == 0 and counter2 % 32 != 0:
        hexDataFinal += ' '
    elif counter2 % 32 == 0:
        hexDataFinal += '\n'
    counter2 += 1
packet_info["hexa_frame"] = PreservedScalarString(hexDataFinal)
output.append(packet_info)
```

Tento cyklus vyrobí „hexa\_frame“, tak ako ho vidíme v programe Wireshark – dáta sú rozdelené do dvojíc po bytoch a po 16tich takých dvojiciach do riadkov.

```
maximum = 0
for i in IpStatistics:
    if IpStatistics[i] > maximum:
        maximum = IpStatistics[i]
    ipstat.append({"node": i, "number_of_sent_packets": IpStatistics[i]})
for i in IpStatistics:
    if IpStatistics[i] == maximum:
        ipwin.append(i)
```

V slovníku IpStatistics sa nachádzajú všetky unikátne zdrojové IP adresy a k nim prislúchajúce počty rámcov. V prvom cykle sa tieto informácie zapíšu do finálnej podoby, v ktorej potom pôjdu do .yaml výstupu a navyše sa nájde tá, z ktorej sa odoslalo najviac rámcov.

```
fileOutput = {"name": "PKS2023/24", "pcap_name": "trace-26.pcap", "packets": output, "ipv4 senders":
with open("paketyAll.yaml", "w") as yaml_file:
    ruamel.yaml.dump(fileOutput, yaml_file, Dumper=ruamel.yaml.RoundTripDumper)
```

Takto spracované údaje o rámcoch sa potom zapíšu do súboru paketyAll.yaml.

V rámci úlohy 4 sa program ďalej rozvetvuje. Pre aplikačné protokoly nad TCP najprv zanalyzujem TCP komunikácie a podelím ich na kompletne a nekompletné podľa TCP Flags, na konci z takto spracovaných komunikácií vyberiem do výstupu tie, ktoré obsahujú konkrétny aplikačný protokol podľa filtra. Filtre TFTP, ARP a ICMP fungujú podobne, odlišnosti sú len malé – prípadná špecifická informácia o rámci navyše, špecifické zadeľovanie komunikácií podľa zadania.

```
if filter == "HTTP" or filter == "HTTPS" or filter == "Telnet" or filter == "SSH" or
elif filter == "tftp":...
elif filter == "icmp":...
elif filter == "arp":...
```

```
for comm in zaciatky: #pozera ci je komunikacia kompletne (flag SYN, ACK, SYN, ACK, ... FIN, ACK, FI
    zaciatok = False
    koniec = False
    del comm["packets"][0]
    if len(comm["packets"]) > 4:
        if "SYN" in comm["packets"][0]["tcp_flags"] and "ACK" in comm["packets"][1]["tcp_flags"] and "
            zaciatok = True
        if "SYN" in comm["packets"][0]["tcp_flags"] and "SYN" in comm["packets"][1]["tcp_flags"] and "
            zaciatok = True
        if "FIN" in comm["packets"][-4]["tcp_flags"] and "ACK" in comm["packets"][-3]["tcp_flags"] and
            koniec = True
        if "FIN" in comm["packets"][-4]["tcp_flags"] and "FIN" in comm["packets"][-3]["tcp_flags"] and
            koniec = True
        if "FIN" in comm["packets"][-3]["tcp_flags"] and "ACK" in comm["packets"][-3]["tcp_flags"] and
            koniec = True
        if "RST" in comm["packets"][-1]["tcp_flags"]:
            koniec = True
    if zaciatok == True and koniec == True:
        complete_comms.append(comm)
```

Tento cyklus určuje, či je komunikácia kompletná. Kompletná je ak má platný začiatok, jedna z dvoch možností:

1. je v prvom pakete „SYN“, druhom „SYN“ a „ACK“ a v treťom „ACK“
2. je v prvom pakete „SYN“, druhom „ACK“, treťom „SYN“, štvrtom „ACK“

a zároveň má aj platný koniec, jedna zo štyroch možností:

1. v 4. pakete od konca je „FIN“, v 3. od konca „ACK“, 2. od konca „FIN“ a v poslednom pakete je „ACK“
2. v 4. pakete od konca je „FIN“, v 3. od konca „FIN“, 2. od konca „ACK“ a v poslednom pakete je „ACK“
3. V 3. pakete od konca je „FIN“ a „ACK“, v 2. od konca je „FIN“ a „ACK“ a v poslednom pakete je „ACK“
4. V poslednom pakete je „RST“

Napravo je príklad štruktúry externého súboru pre určenie protokolov a portov. Jednotlivé protokoly a porty a ich číselné hodnoty sú uložené v slovníku s príslušným menom. Tieto slovníky potom importujem do hlavného súboru s algoritmom.


```
portsUDP = {
    "7": "echo",
    "19": "chargen",
    "37": "time",
    "53": "DNS",
    "67": "DHCP",
    "68": "DHCP",
    "69": "TFTP",
    "137": "NetBIOS-ns",
    "138": "NetBIOS-dgm",
    "161": "SNMP",
    "162": "SNMP-trap",
    "500": "ISAKMP",
    "514": "syslog",
    "520": "RIP",
    "33434": "traceroute",
}
```

## Používanie programu

Program si po spustení v konzole vypýta filter pre úlohu 4. Možné filtre sú:

- ICMP
- TFTP
- ARP
- HTTP, HTTPS, SSH, Telnet, FTP-control, FTP-data

Tieto vstupy treba napísať do konzoly, program potom zbehne príslušný algoritmus. Ak používateľ zadá nejaký iný filter, alebo sa pomýli v nejakom písmenku, program to nerozpozna a zbehne iba algoritmus pre úlohy 1 až 3.



```
Zadajte filter: HTTP
```



```
zaznam = "/Users/peterbrenkus/Desktop/skola/sem3/pks/Z1/pcap/trace-12.pcap"  
packets = rdpcap(zaznam)
```

Súbor s rámcami na čítanie sa dá nastaviť hneď na začiatku programu, treba uviesť úplnú cestu k nemu.

## Zhodnotenie

Program funguje pre všetky filtre, na všetkých .pcap súboroch, ktoré nám boli poskytnuté na testovanie. Splnil som všetky úlohy vrátane fragmentácie. Fragmentované ICMP rámce program vypíše do súboru output.yaml na koniec – najprv vypíše klasicky kompletne komunikácie, potom nekompletne a za nimi v poli fragmented\_packets sa po skupinách nachádzajú jednotlivé príslušné fragmenty.

Čo sa týka možností rozšírenia, tento projekt je môj prvý v jazyku Python a aj tak vyzerá. Určite by bolo fajn kód upratať, opakujúce sa bloky kódu oddeliť do samostatných funkcií, prípadne implementovať triedy na reprezentáciu rámca a komunikácie. Výsledkom by bol prehľadnejší a efektívnejší algoritmus. Tieto úpravy som nerobil z jednoduchého dôvodu – nestíhal som. Na projekte som pracoval priebežne, nezanedbával som ho, no vzhľadom na deadlines z iných predmetov aj to že som dostal covid s ťažkým priebehom a bol som PN, som ledva stihol do termínu odovzdania vytvoriť aspoň takéto riešenie.

Podľa mňa by bolo super keby zadanie bolo aspoň do cvika (o 3 dni viac), určite by som kód stihol upraviť do viac prezentovateľnej podoby.