

Slovak University of Technology in Bratislava

Faculty of Informatics and Information Technologies

Computer and Communication Networks

Assignment 1

Network communication analyser

Design and implement Ethernet network analyzer for network communications recorded in .pcap file

Table of Contents

The assignment	3
Implementation	5
Flowcharts of network analyser	5
Designing a Mechanism for Analyzing Protocols at Different Layers	7
1. Loading Protocols	7
2. Processing the *. pcap File.....	7
3. Storing Information.....	8
4. Generated Output	8
Example Structure of External Files for Protocol and Port Determination.....	9
Describing the User Interface.....	10
Selecting the Development Environment: PyCharm 2023.2.....	10
Utilized Functions and Libraries in the Environment	10
Evaluation and Potential Extensions	11
Evaluation	11
1. Performance	11
2. Accuracy and Reliability	11
3. Meeting assignments requirements	11
4. User-Friendliness.....	11
Potential Extensions	11
1. New protocols.....	11
2. Error handling.....	11
3. Custom Output format	11
4. Visualization	11

The assignment

Design and implement Ethernet network analyser for network communications recorded in .pcap file and provides the following information about the communications. A fully developed assignment fulfils the following tasks:

1. A listing of all frames in hexadecimal form sequentially as they were recorded in the file. .

For each frame, output should state:

- Sequence number of the frame in the parsed file.
- The length of the frame in bytes provided by the pcap API, as well as the length of this frame carried over the medium.
- Frame type - Ethernet II, IEEE 802.3 with LLC, IEEE 802.3 with LLC and SNAP, IEEE 802.3 - Raw.
- For IEEE 802.3 with LLC, also indicate the Service Access Point (SAP), e.g. STP, CDP, IPX, SAP...
- For IEEE 802.3 with LLC and SNAP, also indicate PID, e.g. AppleTalk, CDP, DTP ...
- The source and destination physical (MAC) addresses of the nodes between which the frame was transmitted.
- In the output, **the frame 16 bytes per line**.. Each line is terminated by a newline character. For the clarity of the output, it is recommended to use a non-proportional (monospace) font.
- The output must be in **YAML**. You should use Ruamel for Python.

2. List of IP addresses and encapsulated protocol on layer 2-4 for Ethernet II frames.

- Encapsulated protocol in frame header. (ARP, IPv4, IPv6)
- Source and destination IP address of the packet.
- For IPv4, also specify the encapsulated protocol. (TCP, UDP...)
- For the 4th layer i.e., TCP and UDP, indicate the source and destination port of the communication and if one of the ports is among the "well-known ports", also include the name of the application protocol. Note that the IP header can range in size from 20B to 60B.
- Protocol numbers within Ethernet II (Ethertype field), in IP packet (Protocol field) and port numbers for transport protocols must be **read from one or more external text files** (Task 2 points a, c, and d). Example of possible external file formatting is at the end of this document.
- Output also names** in addition to numbers for **well-known protocols and ports** (at minimum for the protocols listed in tasks 1) and 2). The program shall output name of encapsulated protocol previously unknown protocol after its name and protocol (or port) number are added to the external file.
- A library file used by the program is not considered an external file.

3. Provide the following statistics for IPv4 packets at the end of output from task 2:

- A list of IP addresses of all sending nodes and number of packets each sent.
- The IP address of the node that sent (regardless of the recipient) the most packets, and number of packets. If there are more that sent the most, list them all.

Your program with communication analysis for selected protocols:

Pre-preparation:

- Implement the -p (as protocol) CLI option, which will be followed by another argument, namely the abbreviation of the protocol taken from the external file, e.g. analyzer.py -p HTTP. If the option is followed by any argument or the specified argument is a non-existent protocol, the program will print an error message and return to the beginning. Alternatively, a menu can be implemented, but the output must be written to a YAML file.
- The output of each frame in the following filters must also meet the requirements set in Tasks 1 and 2 (L2 and L3 analysis). If the argument following "-p" option specifies connection-oriented protocol communication (i.e. encapsulated in TCP):
- List all complete communications with their order number. Complete communication must include establishment (SYN) and termination (FIN on both sides; or FIN and RST; or RST only) of the connection. There are two ways for opening and four ways for closing a complete communication.

- d. List the first incomplete communication that contains only the connection establishment or only termination.
- e. Your analyser must support the following connection-oriented protocols: HTTP, HTTPS, TELNET, SSH, FTP radiation, FTP data.

If the argument following "-p" option specifies connectionless protocol (over UDP):

- f) For the FTP protocol list all frames and clearly list them in communications, not only the first frame on UDP port 69, but identify all frames for each TFTP communication and clearly show which frames belong to which communication. We consider a complete TFTP communication to be one where the size of the last datagram with data is smaller than the agreed block size when the connection is established, and at the same time the sender of this packet receives an ACK from the other side. See chapters: TFTP General and TFTP Detailed Operation.

If the argument following "-p" option specifies ICMP protocol:

- g) The program identifies all types of ICMP messages. Split the Echo request and Echo reply (including Time exceeded) into complete communications based on the following principle. First, you need to identify the source and destination IP pairs that exchanged ICMP messages and associate each pair with their ICMP messages. Echo request and Echo reply contain fields Identifier and Sequence in the header. The Identifier field indicates the communication number within the IP address pair and the Sequence field indicates the sequence number within the communication. Both fields can be the same for different IP source and IP destination pairs. This implies that the new communication is identified by the IP address pair and the ICMP field Identifier. All other ICMP message types and ICMP Echo request/reply messages without a pair will be output as incomplete communications.
- h) For each ICMP frame, also indicate the ICMP message type (Type field in the ICMP header), e.g. Echo request, Echo reply, Time exceeded, etc. For complete communications, also list the ICMP fields Identifier and Sequence.

If the argument following "-p" option specifies ARP protocol:

- i) List all ARP pairs (request – reply), also indicate the IP address for which the MAC address is being sought, for Reply indicate a specific pair - IP address and MAC address found. If multiple ARP-Request frames were sent to the same IP address, first identify all ARP pairs and list them in one complete communication, regardless of the source address of the ARP-Request. If there are ARP-Requests frames without ARP-Reply, list them all in one incomplete communication. Likewise, if it identifies more ARP reply than ARP request messages to the same IP, then list all ARP reply without ARP request in one incomplete communication. Ignore other types of ARP messages within the filter.

If the IP packet is fragmented:

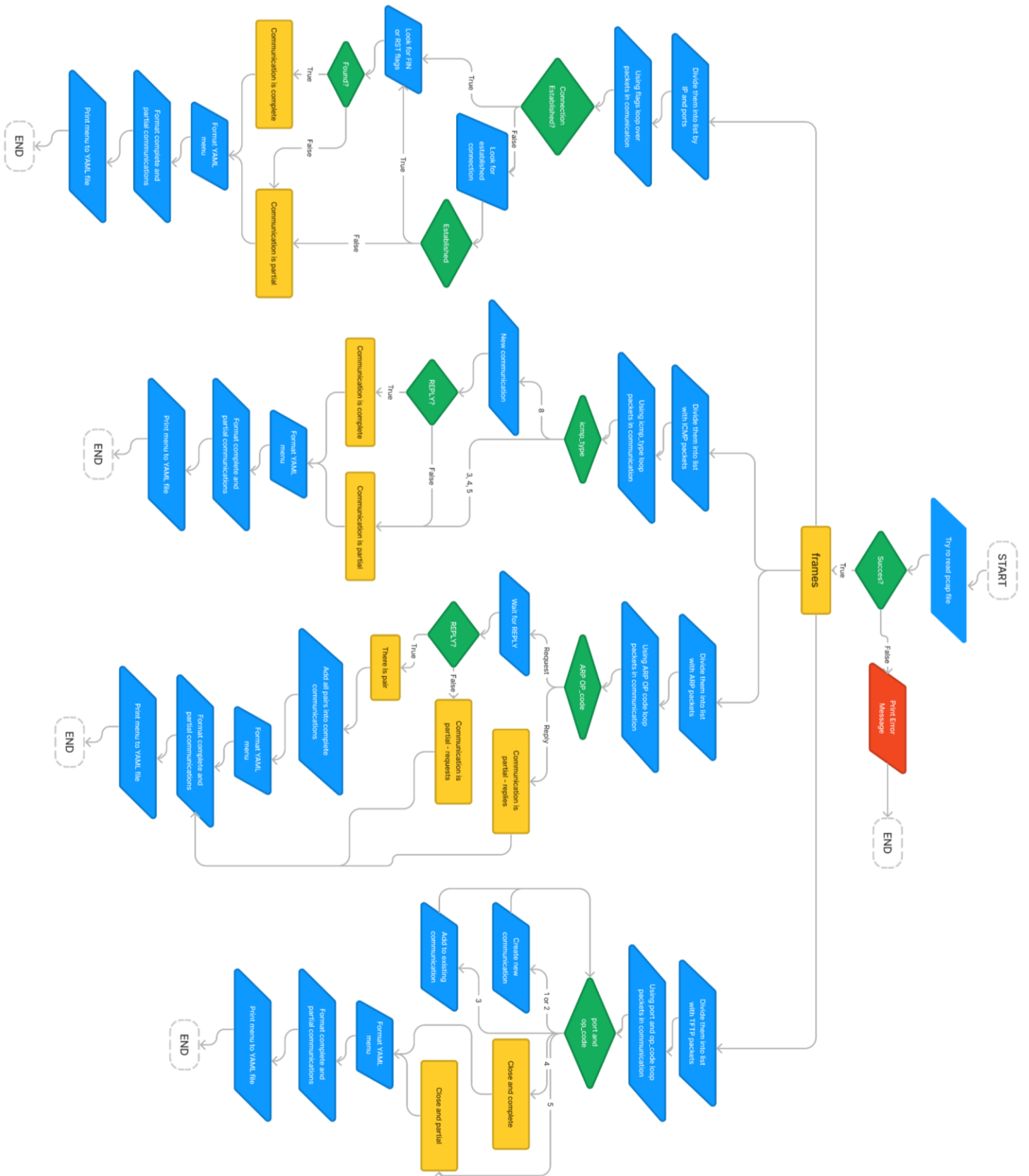
- j) If the size of the IP packet exceeds the MTU, the packet is split into several smaller packets called fragments before being sent and then the whole message is reassembled after receiving all the fragments on the receiver's side. For ICMP filter, identify all fragmented IP packets and list for each such packet all frames with its fragments in the correct order. For fragment merging, study the fields Identification, Flags and Fragment Offset in the IP header and include them also for packets in such communication that contain fragmented packets as id, flags_mf and frag_offset, more details HERE. The task is just an extension of the ICMP filter task, so the input argument for the protocol is same.

The solution must include documentation:

- a. Clarity and comprehensibility of the submitted documentation are required in addition to the overall solution quality. Full marks are awarded only for documents that provide all the essentials about the functioning of their program. This includes the processing diagram *.pcap files and a description of individual parts of the source code (libraries, classes, methods, etc.).
- b. Documentation shall comprise: >- Title page, >- Diagram (activity, flowchart) of processing (concept) and operation of the program, >- The proposed mechanism for protocol analysis on individual layers, >- An example of the structure of external files for specifying protocols and ports, >- Annotated user interface, >- Chosen implementation environment, >- Evaluation and possible extensions.



Task 4:



Designing a Mechanism for Analyzing Protocols at Different Layers

Here's an overview of the proposed analysis mechanism:

1. Loading Protocols

The first step involves loading protocols and ports from text files. These files contain mappings of numeric values to protocol names (ETHER.txt, LLC.txt, IP.txt, L4.txt) and port numbers (L4.txt ICMP.txt). Loading these protocols allows for improved identification and decoding of packets at various layers.

```
protocols_llc = load_protocols_from_file(100)
protocols_ether = load_protocols_from_file(513)
protocols_ip = load_protocols_for_ip()
ports = load_ports()
icmp_codes = load_icmp()
```

Each of these functions opens *.txt file, the use .split() and save protocols and ports into arrays.

2. Processing the *.pcap File

In the next step, the program opens the *.pcap file and processes each packet within it. For each packet, it performs the following actions based on user's input (choosing task to be performed).

Retrieving frame/packet information:

- The program uses a counter to assign frame_number to each frame.
- Calculating the frame's length in the .pcap file and its length in the medium.
- Determining the frame type (ETHERNET II, IEEE 802.3 LLC [& SNAP], IEEE 802.3 RAW.).
- Obtaining the source and destination MAC addresses.
- For Ethernet II frames, further details are gathered, such as IP addresses and transport ports if the frame is ARP or IPv4.
- For IEEE 802.3 LLC & SNAP frames, information from the PID (Protocol Identifier) field is extracted.
- Furthermore, in specific layers, op_codes or arp_operators etc. are extracted from frame.

Example: d) Obtaining the source and destination MAC addresses:

```
# Funkcia na získanie zdrojovej a cieľovej MAC adresy
def get_mac_addresses(packet):
    zdroj_mac = ''
    ciel_mac = ''

    # Získanie zdrojovej a cieľovej MAC adresy
    for i in range(6):
        zdroj_mac += str(hexlify(packet[i:i + 1]))[2:-1] + ':'
    for i in range(6, 12):
        ciel_mac += str(hexlify(packet[i:i + 1]))[2:4] + ':'

    # Upravenie MAC adres do požadovaného formátu odstránenie posledného ':'
    zdroj_mac = zdroj_mac[:-1]
    ciel_mac = ciel_mac[:-1]

    return zdroj_mac.upper(), ciel_mac.upper()
```

3. Storing Information

The acquired information about each packet is stored in structured data, which is subsequently used in creating the output YAML file.

In this example, the structure of each packet is stored in variable called info:

```
info["len_frame_pcap"] = len_frame_pcap
info["len_frame_medium"] = len_frame_medium
info["frame_type"] = frame_type
info["src_mac"] = src_mac
info["dst_mac"] = dst_mac

if frame_type == "IEEE 802.3 LLC & SNAP":
    info["pid"] = pid_sap
elif frame_type == "IEEE 802.3 LLC":
    info["sap"] = pid_sap

if frame_type == "ETHERNET II":
    ether_type = int(str(hexlify(packet[12:14]))[2:-1], 16)
    if ether_type in protocols_ether:
        info["ether_type"] = protocols_ether[ether_type]
        if protocols_ether[ether_type] == 'ARP':
            arp_operation = int(str(hexlify(packet[20:22]))[2:-1], 16)
            if arp_operation == 1:
                info["arp_opcode"] = "REQUEST"
            elif arp_operation == 2:
                info["arp_opcode"] = "REPLY"

        if protocols_ether[ether_type] == 'IPv4':
            info["src_ip"] = src_ip
            info["dst_ip"] = dst_ip
            protocol_l4 = int(str(hexlify(packet[23:24]))[2:-1], 16)
            if protocol_l4 in protocols_ip:
                info["protocol"] = protocols_ip[protocol_l4]
                if protocols_ip[protocol_l4] == 'TCP' or protocols_ip[protocol_l4] == 'UDP':
                    info["src_port"] = src_port
                    info["dst_port"] = dst_port
                    if app_protocol != '':
                        info["app_protocol"] = app_protocol
                elif protocols_ip[protocol_l4] == 'ICMP':
                    icmp_type = int(str(hexlify(packet[34:35]))[2:-1], 16)
                    if icmp_type in icmp_codes:
                        info["icmp_type"] = icmp_codes[icmp_type]
```

4. Generated Output

At the end of the analysis, the program generates an output file in YAML format, where information about all processed packets is saved.

a) Varied Output Based on Selected Task

The output of the program dynamically adjusts according to the selected task. To illustrate, when Task 3 is chosen, the output encompasses a comprehensive list of all IPv4 addresses, along with the number of packets sent by each. The file concludes by highlighting the IPv4 address that sent the highest number of packets.

b) Maintaining Output Structure to Meet Assignment Schemas

The structure of the output file adheres rigorously to the schemas prescribed by this assignment. Each output undergoes validation using the 'validator.py' program, ensuring that every piece of information within the output conforms to the specified format.

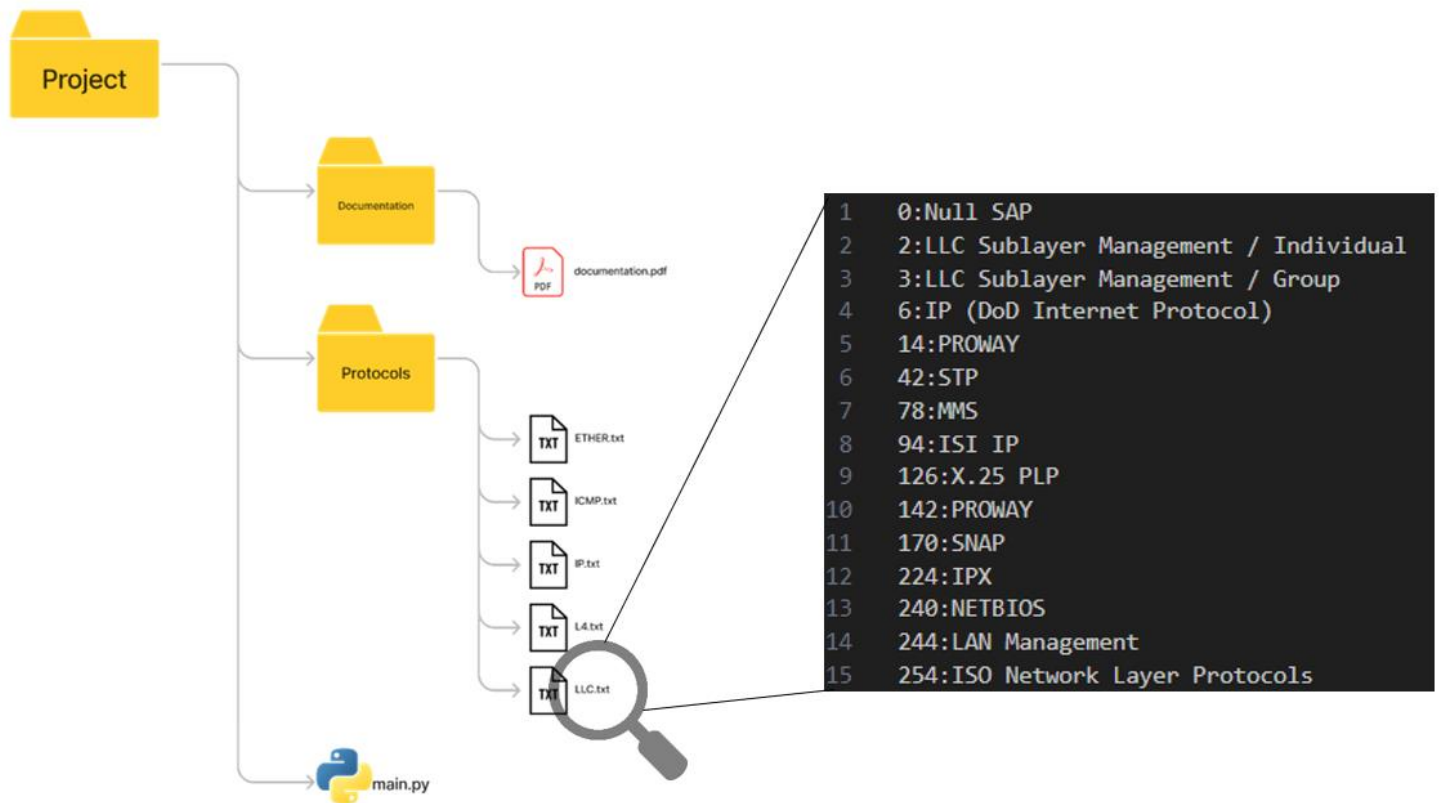
Example: output in YAML format

```
name: PKS2023/24
pcap_name: trace-27.pcap
packets:
...
- frame_number: 2141
  len_frame_pcap: 60
  len_frame_medium: 64
  frame_type: ETHERNET II
  src_mac: FF:FF:FF:FF:FF:FF
  dst_mac: 84:B8:02:66:72:34
  ether_type: ARP
  arp_opcode: REQUEST
  hexa_frame: |
    FF FF FF FF FF FF 84 B8 02 66 72 34 08 06 00 01
    08 00 06 04 00 01 84 B8 02 66 72 34 93 AF 90 01
    00 00 00 00 00 00 93 AF 91 95 00 00 00 00 00 00
    00 00 00 00 00 00 00 00 64 45 6C 8D

ipv4_senders:
- node: 201.43.83.60
  number_of_sent_packets: 1
- node: 10.91.0.1
...
...
  number_of_sent_packets: 590
max_send_packets_by:
- 147.175.144.1
```

Example Structure of External Files for Protocol and Port Determination

In this section, we provide an example of the structured format for external files used to define protocols and ports within the program. These external files serve as crucial references for protocol and port identification. The structured format ensures ease of access and interpretation for both the program and the user.



Describing the User Interface

In this section, we'll delve into a detailed description of the user interface. This aspect is pivotal as it forms the bridge between the user and the program. A clear understanding of the user interface is essential for effectively utilizing the program's features and functionalities.

The user is prompted to provide input, which can be either the complete file path to a "*.pcap" file or, if the file is located in the same directory, simply the file name. Subsequently, the user is presented with a menu of tasks to select from, numbered 1, 2, 3, or the fourth task, which enables the user to specify a filter name based on a variety of network protocols. Additionally, in cases of incorrect input, the program gracefully guides the user towards the next appropriate step.

```
Dávid Truhlář - 120897 - PKS Zadanie číslo 1
🌐🔍 Analýzátor sieťovej komunikácie 🌐🔍

-----
👉 Zadáj názov súboru: ➡ eth-1.pcap
📁 test_pcap_files/eth-1.pcap 📁

📄 1 a 2 Výpis informácií o pakete
📊 3 Zobrazenie štatistiky - IP
🔍 4 Zadáj názov filtra:
    HTTP | HTTPS | TELNET | SSH
    FTPcontrol | FTPdata | TFTP | ARP | ICMP
👉 exit

👉 Vyber úlohu: ➡ 3
✅ Výstup bol uložený do 📄 output-all.yaml 📄
Quit? (y/n) ➡ y
🌐🔍
```

```
👉 Vyber úlohu: ➡ 4

Úloha 4 - Zadáj meno filtra:
HTTP | HTTPS | TELNET
SSH | FTPcontrol | FTPdata
TFTP | ARP | ICMP

👉 Vyber úlohu:
```


```
👉 Zadáj názov súboru: ➡ badexample
❌ 📁 test_pcap_files/badexample neexistuje
👉 Skús znova alebo zadaj 📁 exit

👉 Zadáj názov súboru:
➡ |
```

```
👉 Vyber úlohu: ➡ 5
❌ Zadal si nesprávny vstup, skús znova / exit

👉 Vyber úlohu:
```

Selecting the Development Environment: PyCharm 2023.2

As the chosen IDE, PyCharm 2023.2 by JetBrains was selected for its rich set of -debugging features and exceptional user-friendliness. PyCharm excels in providing a versatile and developer-centric environment, simplifying the process of writing, testing, and maintaining code. However, it's worth noting that the trade-off for this user-friendliness is a slightly slower response time compared to 'VS Code'.



Utilized Functions and Libraries in the Environment

Within the program's environment, several essential functions and libraries were employed to facilitate its functionality:

```
from binascii import hexlify
from os.path import exists
import ruamel.yaml
from scapy.compat import raw
from scapy.utils import rdpcap
```

Evaluation and Potential Extensions

Evaluation

1. Performance

The program's performance may vary across different computers. In my testing, I'd rate the performance as 8/10. It's worth noting that analyzing .pcap files with thousands of packets naturally takes more time but is well within the program's capabilities.

2. Accuracy and Reliability

The foremost priority during the code development process was ensuring the accuracy and correctness of the program's output. Each step of the analysis, including communication matching and protocol details, was meticulously validated against real-world analysis performed with Wireshark.

3. Meeting assignments requirements

To guarantee that all the stipulated assignment requirements were met, I relied on examples and schemas to validate the correctness of the program's output. Furthermore, a critical aspect of testing involved the use of 'validator.py' to ensure that the results adhered precisely to the specified output format.

4. User-Friendliness

Enhancing user-friendliness further could have been achieved with more time. Despite time constraints, the program boasts an intuitive menu that offers users a clear and interactive console application. User requirements are explicitly stated within the console, ensuring that tasks are always transparent. The program provides immediate feedback after each action, whether it's a successful execution or an error.

Potential Extensions

1. New protocols

Consider expanding the program's capabilities by adding support for additional network protocols and standards. For instance, support for IPv6 could be added and its associated extension headers, broadening the program's scope to accommodate evolving network technologies.

2. Error handling

Implement advanced error handling mechanisms to enhance user experience and program robustness. With improved error handling, the program can provide more informative error messages, making it easier for users to troubleshoot issues, leading to a smoother analysis process.

3. Custom Output format

Enhance the program's flexibility by allowing users to define custom output formats or templates. This feature would enable users to tailor the reporting and analysis output to their specific needs.

4. Visualization

Consider incorporating data visualization tools into the program. Visual representations of network traffic patterns can make it easier for users to interpret and analyze the results.