MANIPAL INSTITUTE OF TECHNOLOGY
BENGALURU
(A constituent unit of MAHE, Manipal)

B.TECH. FIFTH SEMESTER

COMPUTER SCIENCE AND ENGINEERING

MINIPROJECT

ON

"NETWORK TRAFFIC ANALYZER"

Submitted by

Shruti Singhania (215805120)

Under the Guidance of

Dr. Geetabai S Hukkeri

Assistant Professor

Department of Computer Science and Engineering

Manipal Institute of Technology

MAHE, Bengaluru, India

2023

Format of report

- Introduction
- Problem statement
- Objective
- Methodology
- Implementation details
- Results
- Conclusion
- References

INTRODUCTION

"In an epoch defined by the inexorable march of technological progress, the sprawling tapestry of global connectivity has become the crucible of modern civilization. From the towering citadels of multinational corporations to the intricate neural networks of smart cities, the ceaseless torrent of data has metamorphosed into the very lifeblood of human endeavour. Yet, within this grandeur lies a nexus of challenges that beckon audacious solutions.

The Network Traffic Analyzer project embarks on an odyssey of innovation, a clarion call to transcend the boundaries of conventional network management. Enshrined in a profound understanding of the crescendoing intricacies and kaleidoscopic demands of contemporary network ecosystems, this endeavor aspires to revolutionize the very paradigm through which we perceive, monitor, and safeguard our digital conduits. By fusing the avant-garde trinity of artificial intelligence, machine learning, and advanced network protocol analysis, this project emerges as the standard-bearer of a new epoch in network administration.

In a landscape where networks burgeon into veritable metropolises of data, the exigency to promptly discern and address challenges such as congestion, security vulnerabilities, and resource optimization inefficiencies has ascended to the zenith of organizational priorities. While extant solutions have cast a flickering light, they often founder in providing the agility and sagacity requisite to navigate the turbulent maelstroms of modern networks. The Network Traffic Analyzer project rises to the occasion, presenting a transformative solution that not only bestows administrators with real-time visibility but enriches this vista with sagacious insights and actionable intelligence.

With a multi-dimensional approach encompassing data packet forensics, anomaly detection algorithms, and trend analysis, the Network Traffic Analyzer unfurls as an indefatigable sentinel, ceaselessly patrolling the digital boulevards of organizations. Its extensibility and scalability are the crucible of its versatility, ensuring seamless integration within an expansive pantheon of network architectures, from the sprawling tapestries of enterprise networks to the critical lattices undergirding essential services.

In this monumental odyssey, our ambition transcends the immediacy of challenges. We envisage and anticipate the contours of future obstacles, and through a crucible of relentless research, exacting design, and an unwavering commitment to excellence, the Network Traffic Analyzer project aspires to be the cornerstone of a new era in network infrastructures. It is poised not merely to meet but to transcend the imperatives of an ever-evolving digital landscape, setting forth a legacy that resonates far beyond the boundaries of today.

As the project unfurls, it will catalyse a seismic shift in how networks are conceptualized, monitored, and fortified. This endeavour is not just a response to contemporary demands but an investment in the resilience and adaptability required for the networks of the future. Through this project, we aim to inspire a new generation of network administrators and engineers, poised to navigate the digital frontiers with unparalleled acumen and foresight. The Network Traffic Analyzer project is not merely a venture; it is a testament to the indomitable spirit of innovation and the unyielding pursuit of excellence in the face of an ever-evolving technological landscape."

PROBLEM STATEMENT

In today's interconnected world, efficient and secure network operations are paramount for businesses and organizations. The exponential growth in data traffic, coupled with the increasing complexity of network infrastructures, necessitates the development of a robust Network Traffic Analyzer (NTA) solution.

The objective of this project is to design and implement a comprehensive NTA tool that provides real-time monitoring, analysis, and reporting of network traffic patterns. The system should be capable of:

- **Packet Capture and Inspection:**
  Capture network packets across different protocols (e.g., TCP/IP, UDP).
  Extract and analyse packet headers for crucial information (e.g., source and destination IP addresses, ports, protocols).
- **Traffic Classification:**
  Categorize network traffic into different types (e.g., web browsing, video streaming, file transfers).
  Implement deep packet inspection techniques to identify specific applications and services.
- **Anomaly Detection:**
  Implement algorithms to detect unusual or suspicious traffic patterns (e.g., DDoS attacks, port scanning).
  Generate alerts for potential security incidents.
- **Bandwidth Monitoring and Utilization:**
  Monitor and report on the overall bandwidth usage of the network.
  Provide insights into the distribution of traffic among different services and users.

- **Traffic Visualization and Reporting:**
  Present traffic statistics through intuitive graphical interfaces (e.g., charts, graphs).
  Generate customizable reports for network administrators and stakeholders.
- **Historical Data Analysis:**
  Store and retrieve historical network traffic data for trend analysis and long-term planning.
- **Scalability and Performance:**
  Ensure the solution is capable of handling networks of varying sizes, from small businesses to large enterprises.
  Optimize for minimal impact on network performance.
- **Documentation and User Guides:**
  Provide comprehensive documentation including installation instructions, configuration guidelines, and user manuals.

Deliverables:

- A functional Network Traffic Analyzer software solution.
- Source code and documentation for future maintenance and enhancement.
- User guides and manuals for system administrators and end-users.

Optional Enhancements (Stretch Goals):

- Integration with SIEM (Security Information and Event Management) systems.
- Machine learning-based anomaly detection for advanced threat detection.
- Support for encrypted traffic analysis (e.g., TLS/SSL).

OBJECTIVE

The Network Traffic Analyzer project aims to create a real-time tool for monitoring, analysing, and managing network traffic. It will provide insights into packet-level details, categorize traffic, and swiftly detect anomalies or security threats. The tool will optimize bandwidth usage, store historical data, and ensure compliance with industry standards. Detailed reporting and visualizations will aid decision-making, and secure authentication mechanisms will control access. The project prioritizes scalability, performance, and user-friendliness through comprehensive documentation. Optional enhancements include SIEM integration, machine learning-based anomaly detection, and encrypted traffic analysis, ultimately enhancing network efficiency and security.

METHODOLOGY

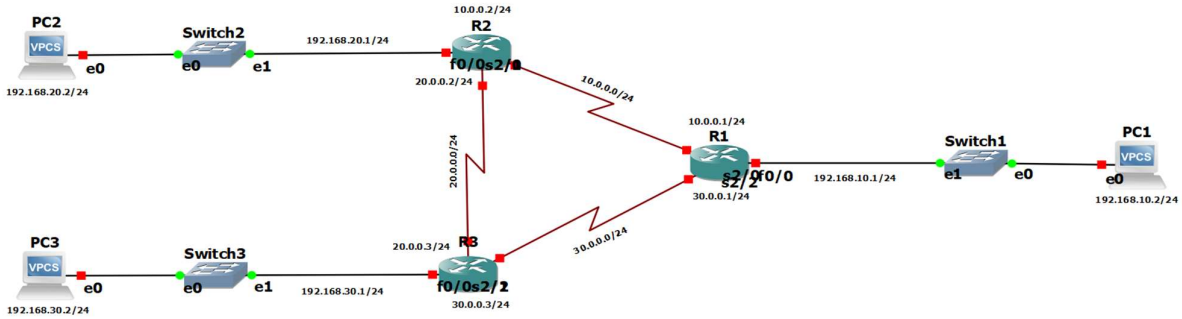
The Network Traffic Analyzer project follows a structured methodology. It starts with clear project planning and requirements gathering. The architecture and technology stack are chosen for scalability and compatibility. Implementation includes packet capture, traffic classification, and anomaly detection. The user interface is designed for intuitive use, while the database is optimized for efficient data storage. This systematic approach ensures an effective development process.

IMPLEMENTATION DETAILS

The implementation of the Network Traffic Analyzer project involves the integration of various tools and technologies. Firstly, GNS3, a network simulation platform, is leveraged to create a virtual environment for testing and analysing network traffic. Wireshark, a widely used network packet analyser, is employed to capture and inspect packets in real-time. Python, a versatile programming language, is utilized for its extensive libraries and flexibility. The Pandas library is particularly instrumental for data manipulation, allowing for tasks such as data cleaning and transformation to be carried out efficiently.

Data cleaning is a crucial step in the process, involving the identification and correction of any inconsistencies or anomalies within the captured network traffic data. This ensures the accuracy and reliability of subsequent analyses. Subsequently, Python's data visualization libraries are harnessed to generate clear and informative visual representations of the network traffic patterns. This aids in the identification of trends, anomalies, and other relevant insights. By combining GNS3 for simulation, Wireshark for packet capture, Python for data manipulation, and Pandas for data cleaning and transformation, this implementation approach ensures a comprehensive and effective Network Traffic Analyzer system.

- NETWORKING CIRCUIT:



- PC1:

```
PC1> ip 192.168.10.2/24 192.168.10.1
Checking for duplicate address...
PC1 : 192.168.10.2 255.255.255.0 gateway 192.168.10.1

PC1> show ip

NAME        : PC1[1]
IP/MASK     : 192.168.10.2/24
GATEWAY     : 192.168.10.1
DNS         :
MAC         : 00:50:79:66:68:00
LPORT       : 10036
RHOST:PORT  : 127.0.0.1:10037
MTU:        : 1500

PC1> ping 10.0.0.1
84 bytes from 10.0.0.1 icmp_seq=1 ttl=255 time=15.409 ms
84 bytes from 10.0.0.1 icmp_seq=2 ttl=255 time=15.256 ms
84 bytes from 10.0.0.1 icmp_seq=3 ttl=255 time=16.016 ms
84 bytes from 10.0.0.1 icmp_seq=4 ttl=255 time=15.791 ms
84 bytes from 10.0.0.1 icmp_seq=5 ttl=255 time=16.435 ms
```

```
PC1> load pc1_config

Executing the file "pc1_config"


Checking for duplicate address...
PC1 : 192.168.10.2 255.255.255.0 gateway 192.168.10.1
```

- PC2:

```
PC2> ip 192.168.20.2/24 192.168.20.1
Checking for duplicate address...
PC1 : 192.168.20.2 255.255.255.0 gateway 192.168.20.1

PC2> show ip

NAME        : PC2[1]
IP/MASK     : 192.168.20.2/24
GATEWAY     : 192.168.20.1
DNS         :
MAC         : 00:50:79:66:68:01
LPORT       : 10038
RHOST:PORT  : 127.0.0.1:10039
MTU:        : 1500
```

```
PC2> load pc2_config

Executing the file "pc2_config"


Checking for duplicate address...
PC1 : 192.168.20.2 255.255.255.0 gateway 192.168.20.1
```

- PC3:

```
PC3> load pc3_config

Executing the file "pc3_config"


Checking for duplicate address...
PC1 : 192.168.30.2 255.255.255.0 gateway 192.168.30.1
```

```
PC3> ip 192.168.30.2/24 192.168.30.1
Checking for duplicate address...
PC1 : 192.168.30.2 255.255.255.0 gateway 192.168.30.1

PC3> show ip

NAME        : PC3[1]
IP/MASK     : 192.168.30.2/24
GATEWAY     : 192.168.30.1
DNS         :
MAC         : 00:50:79:66:68:02
LPORT       : 10040
RHOST:PORT  : 127.0.0.1:10041
MTU:        : 1500
```

- ROUTER1 (R1):

```
R1#en
R1#config t
Enter configuration commands, one per line.  End with CNTL/Z.
R1(config)#int s2/0
R1(config-if)#ip address 10.0.0.1 255.255.255.0
R1(config-if)#no shut
R1(config-if)#
*Mar  1 00:07:17.691: %LINK-3-UPDOWN: Interface Serial2/0, changed state to up
R1(config-if)#
*Mar  1 00:07:18.695: %LINEPROTO-5-UPDOWN: Line protocol on Interface Serial2/0, changed state to up
R1(config-if)#int s2/1
*Mar  1 00:07:42.139: %LINEPROTO-5-UPDOWN: Line protocol on Interface Serial2/0, changed state to down
R1(config-if)#int s2/2
R1(config-if)#ip address 30.0.0.1 255.255.255.0
R1(config-if)#no shut
R1(config-if)#
R1(config-if)#
*Mar  1 00:09:02.563: %LINK-3-UPDOWN: Interface Serial2/2, changed state to up
R1(config-if)#
*Mar  1 00:09:03.567: %LINEPROTO-5-UPDOWN: Line protocol on Interface Serial2/2, changed state to up
R1(config-if)#end
R1#
*Mar  1 00:09:15.195: %SYS-5-CONFIG_I: Configured from console by console
```

```
R1#config t
Enter configuration commands, one per line.  End with CNTL/Z.
R1(config)#int f0/0
R1(config-if)#ip address 192.168.10.1 255.255.255.0
R1(config-if)#no shutdown
R1(config-if)#
```

```
R1#config t
Enter configuration commands, one per line.  End with CNTL/Z.
R1(config)#ip route 20.0.0.0 255.255.255.0 10.0.0.2
R1(config)#end
R1#
*Mar  1 00:21:23.447: %SYS-5-CONFIG_I: Configured from console by console
```

```
R1#config t
Enter configuration commands, one per line.  End with CNTL/Z.
R1(config)#ip route 192.168.20.0 255.255.255.0 10.0.0.2
R1(config)#ip route 192.168.30.0 255.255.255.0 30.0.0.3
R1(config)#exit
R1#
*Mar  1 00:48:00.567: %SYS-5-CONFIG_I: Configured from console by console
```

```
R1#show ip int brief
Interface                  IP-Address      OK? Method Status                Protocol
FastEthernet0/0            192.168.10.1    YES manual up                    up
Serial0/0                  unassigned      YES unset  administratively down down
FastEthernet0/1            unassigned      YES unset  administratively down down
Serial0/1                  unassigned      YES unset  administratively down down
Serial0/2                  unassigned      YES unset  administratively down down
FastEthernet1/0            unassigned      YES unset  administratively down down
Serial2/0                  10.0.0.1        YES manual up                    up
Serial2/1                  unassigned      YES unset  administratively down down
Serial2/2                  30.0.0.1        YES manual up                    up
Serial2/3                  unassigned      YES unset  administratively down down
```

```
R1#show ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static route
       o - ODR, P - periodic downloaded static route

Gateway of last resort is not set

S    192.168.30.0/24 [1/0] via 30.0.0.3
     20.0.0.0/24 is subnetted, 1 subnets
S       20.0.0.0 [1/0] via 10.0.0.2
C    192.168.10.0/24 is directly connected, FastEthernet0/0
S    192.168.20.0/24 [1/0] via 10.0.0.2
     10.0.0.0/24 is subnetted, 1 subnets
C       10.0.0.0 is directly connected, Serial2/0
     30.0.0.0/24 is subnetted, 1 subnets
C       30.0.0.0 is directly connected, Serial2/2
```

```
R1#copy startup-config running-config
Destination filename [running-config]?

1636 bytes copied in 7.936 secs (206 bytes/sec)
```

- ROUTER2 (R2):

```
R2#en
R2#config t
Enter configuration commands, one per line.  End with CNTL/Z.
R2(config)#int s2/0
R2(config-if)#ip address 10.0.0.2 255.255.255.0
R2(config-if)#no shut
R2(config-if)#
*Mar  1 00:10:43.579: %LINK-3-UPDOWN: Interface Serial2/0, changed state to up
R2(config-if)#
*Mar  1 00:10:44.583: %LINEPROTO-5-UPDOWN: Line protocol on Interface Serial2/0, changed state to up
R2(config-if)#int s2/1
R2(config-if)#ip address 20.0.0.2 255.255.255.0
R2(config-if)#no shut
R2(config-if)#
*Mar  1 00:11:10.167: %LINK-3-UPDOWN: Interface Serial2/1, changed state to up
R2(config-if)#
*Mar  1 00:11:11.171: %LINEPROTO-5-UPDOWN: Line protocol on Interface Serial2/1, changed state to up
R2(config-if)#end
R2#
*Mar  1 00:11:16.779: %SYS-5-CONFIG_I: Configured from console by console
```

```
R2#config t
Enter configuration commands, one per line.  End with CNTL/Z.
R2(config)#int f0/0
R2(config-if)#ip address 192.168.20.1 255.255.255.0
R2(config-if)#no shutdown
R2(config-if)#
*Mar  1 00:42:40.399: %LINK-3-UPDOWN: Interface FastEthernet0/0, changed state to up
*Mar  1 00:42:41.399: %LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet0/0, changed state to up
R2(config-if)#exit
R2(config)#ip route 192.168.10.0 255.255.255.0 10.0.0.1
R2(config)#ip route 192.168.30.0 255.255.255.0 20.0.0.3
R2(config)#exit
```

```
R2#config t
Enter configuration commands, one per line.  End with CNTL/Z.
R2(config)#ip route 30.0.0.0 255.255.255.0 10.0.0.1
```

```
R2#show ip int brief
Interface                  IP-Address      OK? Method Status                Protocol
FastEthernet0/0            192.168.20.1    YES manual up                    up
Serial0/0                  unassigned      YES unset  administratively down down
FastEthernet0/1            unassigned      YES unset  administratively down down
Serial0/1                  unassigned      YES unset  administratively down down
Serial0/2                  unassigned      YES unset  administratively down down
FastEthernet1/0            unassigned      YES unset  administratively down down
Serial2/0                  10.0.0.2        YES manual up                    up
Serial2/1                  20.0.0.2        YES manual up                    up
Serial2/2                  unassigned      YES unset  administratively down down
Serial2/3                  unassigned      YES unset  administratively down down
```

```
R2#copy startup-config running-config
Destination filename [running-config]?

1632 bytes copied in 0.796 secs (2050 bytes/sec)
```

```
R2#show ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static route
       o - ODR, P - periodic downloaded static route

Gateway of last resort is not set

S    192.168.30.0/24 [1/0] via 20.0.0.3
     20.0.0.0/24 is subnetted, 1 subnets
C       20.0.0.0 is directly connected, Serial2/1
S    192.168.10.0/24 [1/0] via 10.0.0.1
C    192.168.20.0/24 is directly connected, FastEthernet0/0
     10.0.0.0/24 is subnetted, 1 subnets
C       10.0.0.0 is directly connected, Serial2/0
     30.0.0.0/24 is subnetted, 1 subnets
S       30.0.0.0 [1/0] via 10.0.0.1
```

- ROUTER3 (R3):

```
R3#en
R3#config t
Enter configuration commands, one per line.  End with CNTL/Z.
R3(config)#int s2/2
R3(config-if)#ip address 30.0.0.3 255.255.255.0
R3(config-if)#no shut
R3(config-if)#
*Mar  1 00:14:07.071: %LINK-3-UPDOWN: Interface Serial2/2, changed state to up
R3(config-if)#
*Mar  1 00:14:08.075: %LINEPROTO-5-UPDOWN: Line protocol on Interface Serial2/2, changed state to up
R3(config-if)#int s2/1
R3(config-if)#ip address 20.0.0.3 255.255.255.0
R3(config-if)#no shut
R3(config-if)#
*Mar  1 00:14:49.011: %LINK-3-UPDOWN: Interface Serial2/1, changed state to up
R3(config-if)#
*Mar  1 00:14:50.015: %LINEPROTO-5-UPDOWN: Line protocol on Interface Serial2/1, changed state to up
R3(config-if)#end
R3#
*Mar  1 00:14:56.111: %SYS-5-CONFIG_I: Configured from console by console
```

```
R3#config t
Enter configuration commands, one per line.  End with CNTL/Z.
R3(config)#int f0/0
R3(config-if)#ip address 192.168.30.1 255.255.255.0
R3(config-if)#no shutdown
R3(config-if)#exit
R3(config)#
*Mar  1 00:43:42.571: %LINK-3-UPDOWN: Interface FastEthernet0/0, changed state to up
*Mar  1 00:43:43.579: %LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet0/0, changed state to up
R3(config)#ip route 192.168.10.0 255.255.255.0 30.0.0.1
R3(config)#ip route 192.168.20.0 255.255.255.0 20.0.0.2
R3(config)#exit
```

```
R3#config t
Enter configuration commands, one per line.  End with CNTL/Z.
R3(config)#ip route 10.0.0.0 255.255.255.0 30.0.0.1
R3(config)#end
```

```
R3#show ip int brief
Interface                  IP-Address      OK? Method Status                Protocol
FastEthernet0/0            192.168.30.1    YES manual up                    up
Serial0/0                  unassigned      YES unset  administratively down down
FastEthernet0/1            unassigned      YES unset  administratively down down
Serial0/1                  unassigned      YES unset  administratively down down
Serial0/2                  unassigned      YES unset  administratively down down
FastEthernet1/0            unassigned      YES unset  administratively down down
Serial2/0                  unassigned      YES unset  administratively down down
Serial2/1                  20.0.0.3        YES manual up                    up
Serial2/2                  30.0.0.3        YES manual up                    up
Serial2/3                  unassigned      YES unset  administratively down down
```

```
R3#show ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static route
       o - ODR, P - periodic downloaded static route

Gateway of last resort is not set

C    192.168.30.0/24 is directly connected, FastEthernet0/0
     20.0.0.0/24 is subnetted, 1 subnets
C       20.0.0.0 is directly connected, Serial2/1
S    192.168.10.0/24 [1/0] via 30.0.0.1
S    192.168.20.0/24 [1/0] via 20.0.0.2
     10.0.0.0/24 is subnetted, 1 subnets
S       10.0.0.0 [1/0] via 30.0.0.1
     30.0.0.0/24 is subnetted, 1 subnets
C       30.0.0.0 is directly connected, Serial2/2
```

```
R3#copy startup-config running-config
Destination filename [running-config]?

1632 bytes copied in 8.408 secs (194 bytes/sec)
```

➢ ANALYSIS USING PYTHON:

1. Import the necessary libraries: You'll need a few Python libraries such as pandas for data handling, matplotlib for graphing data, and Networkx for graphing data as nodes if they communicated.
    - import pandas as pd
    - import matplotlib.pyplot as plt
    - import networkx as nx

```
[52] import pandas as pd

[54] import matplotlib.pyplot as plt

[55] import networkx as nx
```

2. Load your data: Next, load your network traffic data based on its file path into a pandas Data Frame.

    df = pd.read_csv('filepath')

    Examine the data: Look at your DataFrame to understand the structure of your data. The columns might include 'No.', 'Time', 'Source', 'Destination', 'Protocol', 'Length', 'Info'.

    print(df.head())

```
wireshark_url = "/content/final_data.csv"
wireshark_data = pd.read_csv(wireshark_url)
wireshark_data.head(10)
```

	No.	Time	Source	Destination	Protocol	Length	Info
0	1	0.000000	NaN	NaN	SLARP	24	Line keepalive, outgoing sequence 87, returned...
1	2	4.017878	NaN	NaN	SLARP	24	Line keepalive, outgoing sequence 89, returned...
2	3	10.634956	NaN	NaN	SLARP	24	Line keepalive, outgoing sequence 88, returned...
3	4	14.704566	NaN	NaN	SLARP	24	Line keepalive, outgoing sequence 90, returned...
4	5	19.551944	NaN	NaN	CDP	336	Device ID: R2 Port ID: Serial2/1
5	6	21.497641	NaN	NaN	SLARP	24	Line keepalive, outgoing sequence 89, returned...
6	7	26.670169	NaN	NaN	SLARP	24	Line keepalive, outgoing sequence 91, returned...
7	8	34.130708	NaN	NaN	SLARP	24	Line keepalive, outgoing sequence 90, returned...
8	9	38.476818	NaN	NaN	CDP	336	Device ID: R3 Port ID: Serial2/1
9	10	38.614501	NaN	NaN	SLARP	24	Line keepalive, outgoing sequence 92, returned...

3. Analyse the data: Use pandas functions like groupby(), count(), and sort_values() to understand devices that initiated conversations, accepted communications, and types of communications

```
sources = wireshark_data.groupby("Source").Source.count()
sources.sort_values()
```

```
Source
10.0.0.2         5
192.168.10.2     5
30.0.0.3         5
20.0.0.2        10
20.0.0.3        10
192.168.20.2    20
192.168.30.2    20
Name: Source, dtype: int64
```

```
[58] destinations = wireshark_data.groupby("Destination").Source.count()
     destinations.sort_values()
```

```
Destination
20.0.0.2        10
20.0.0.3        15
192.168.20.2    25
192.168.30.2    25
Name: Source, dtype: int64
```

```
[59] protocols = wireshark_data.groupby("Protocol").Source.count()
     protocols.sort_values()
```

```
Protocol
CDP      0
SLARP    0
ICMP    75
Name: Source, dtype: int64
```

```
[60] network = nx.from_pandas_edgelist(wireshark_data, source = 'Source', target = 'Destination', edge_attr = None, create_using = None, edge_key = None)
     network.nodes()
```

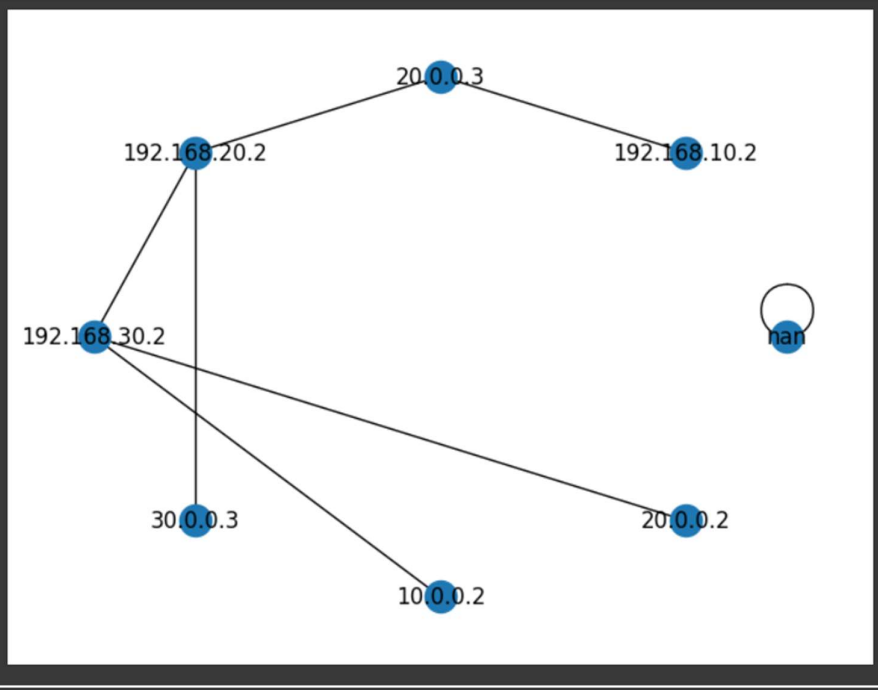
```
NodeView((nan, '192.168.10.2', '20.0.0.3', '192.168.20.2', '192.168.30.2', '30.0.0.3', '10.0.0.2', '20.0.0.2'))
```

```
[61] network.edges()
```

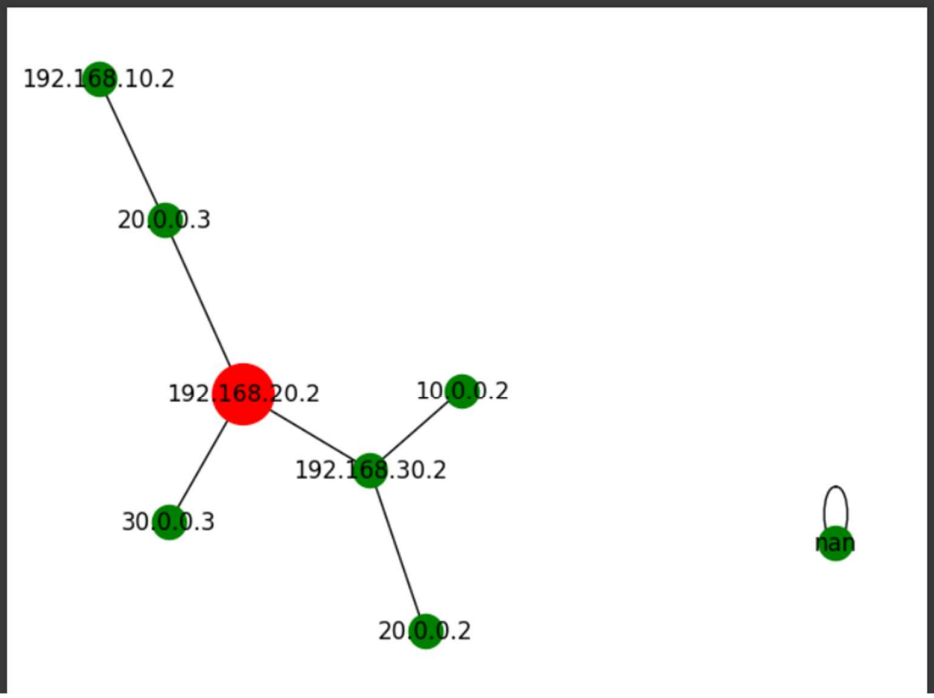
```
EdgeView([(nan, nan), ('192.168.10.2', '20.0.0.3'), ('20.0.0.3', '192.168.20.2'), ('192.168.20.2', '192.168.30.2'), ('192.168.20.2', '30.0.0.3'), ('192.168.30.2', '10.0.0.2'),
('192.168.30.2', '20.0.0.2')])
```

4. Apply Graph Theory: Using Networkx in Python, you can build node graphs that represent a type of communication. Since your data is in a pandas DataFrame, you can use Networkx directly on it and select your nodes as “source” and "target"

```
nx.draw_circular(network, with_labels = True)
```



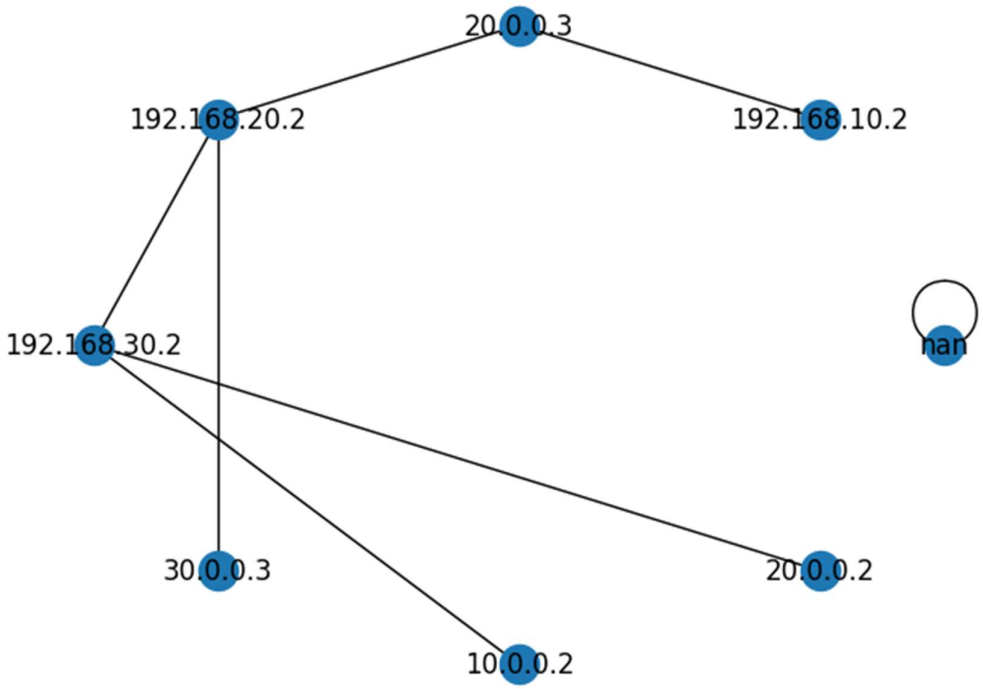

```
suspect = "192.168.20.2"
pos = nx.spring_layout(network)
nx.draw(network, pos, node_color = "green", node_size = 300, with_labels = True)
options = {"node_size": 1000, "node_color" : "r"}
nx.draw_networkx_nodes(network, pos, nodelist = [suspect], **options)
plt.show()
```

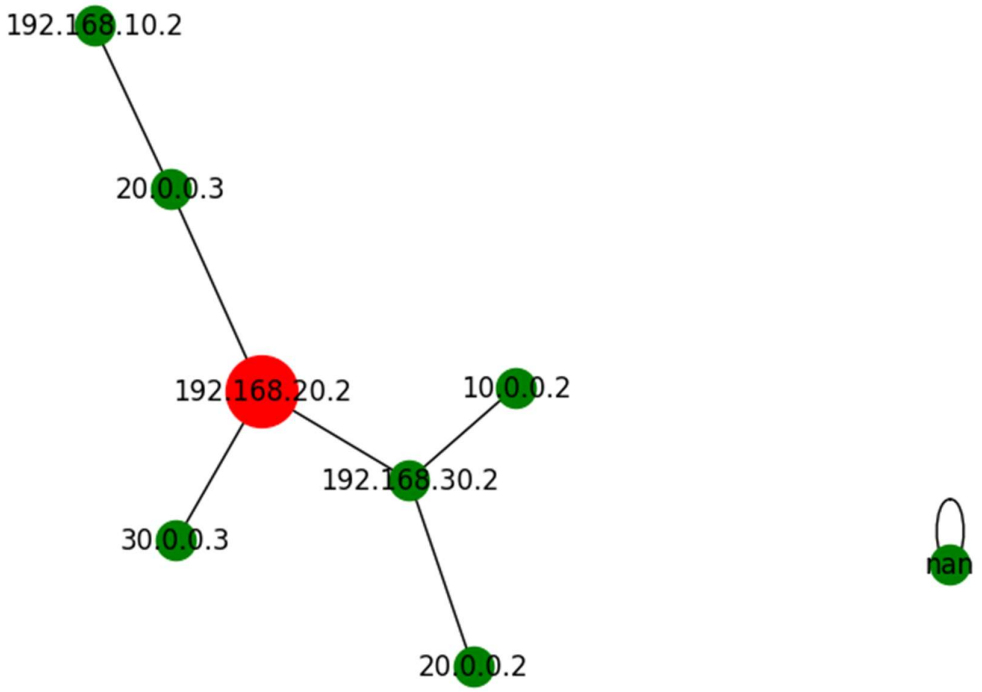


```
wireshark_data.loc[wireshark_data['Source'] == "192.168.20.2"]
```

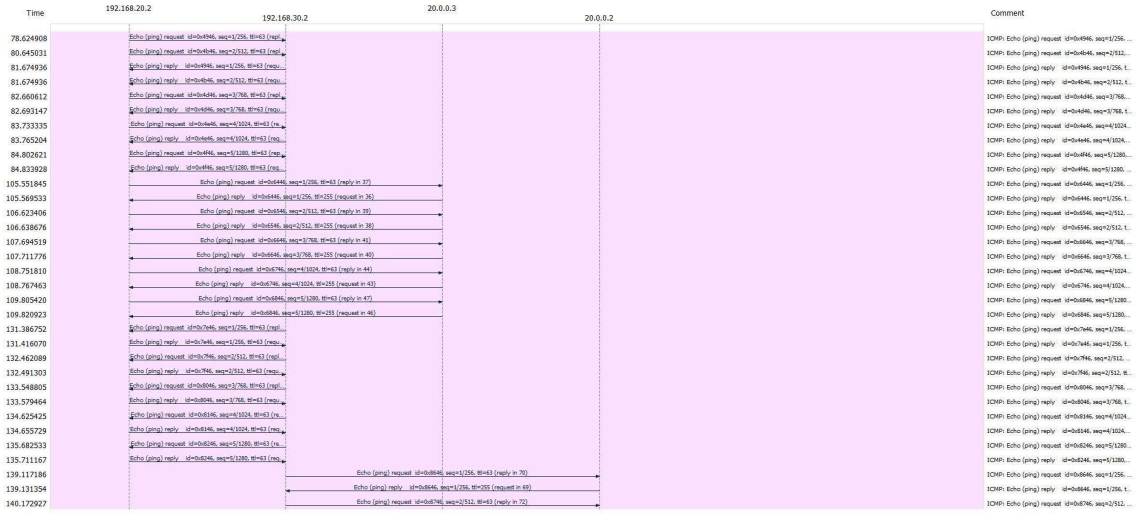
	No.	Time	Source	Destination	Protocol	Length	Info
29	30	116.329792	192.168.20.2	192.168.30.2	ICMP	88	Echo (ping) request id=0x543e, seq=1/256, ttl...
31	32	117.413908	192.168.20.2	192.168.30.2	ICMP	88	Echo (ping) request id=0x553e, seq=2/512, ttl...
33	34	118.498884	192.168.20.2	192.168.30.2	ICMP	88	Echo (ping) request id=0x563e, seq=3/768, ttl...
35	36	119.587313	192.168.20.2	192.168.30.2	ICMP	88	Echo (ping) request id=0x573e, seq=4/1024, tt...
38	39	120.677733	192.168.20.2	192.168.30.2	ICMP	88	Echo (ping) request id=0x583e, seq=5/1280, tt...
48	49	143.547831	192.168.20.2	20.0.0.3	ICMP	88	Echo (ping) request id=0x6f3e, seq=1/256, ttl...
50	51	144.617442	192.168.20.2	20.0.0.3	ICMP	88	Echo (ping) request id=0x703e, seq=2/512, ttl...
53	54	145.689359	192.168.20.2	20.0.0.3	ICMP	88	Echo (ping) request id=0x713e, seq=3/768, ttl...
55	56	146.748177	192.168.20.2	20.0.0.3	ICMP	88	Echo (ping) request id=0x723e, seq=4/1024, tt...
57	58	147.788954	192.168.20.2	20.0.0.3	ICMP	88	Echo (ping) request id=0x733e, seq=5/1280, tt...
65	66	172.670425	192.168.20.2	192.168.30.2	ICMP	88	Echo (ping) reply id=0x8c3e, seq=1/256, ttl...
68	69	173.757182	192.168.20.2	192.168.30.2	ICMP	88	Echo (ping) reply id=0x8d3e, seq=2/512, ttl...
70	71	174.839678	192.168.20.2	192.168.30.2	ICMP	88	Echo (ping) reply id=0x8e3e, seq=3/768, ttl...
72	73	175.927362	192.168.20.2	192.168.30.2	ICMP	88	Echo (ping) reply id=0x8f3e, seq=4/1024, tt...
74	75	177.010503	192.168.20.2	192.168.30.2	ICMP	88	Echo (ping) reply id=0x913e, seq=5/1280, tt...
117	118	256.946029	192.168.20.2	20.0.0.3	ICMP	104	Echo (ping) reply id=0x0003, seq=0/0, ttl=6...
119	120	256.991683	192.168.20.2	20.0.0.3	ICMP	104	Echo (ping) reply id=0x0003, seq=1/256, ttl...

RESULT

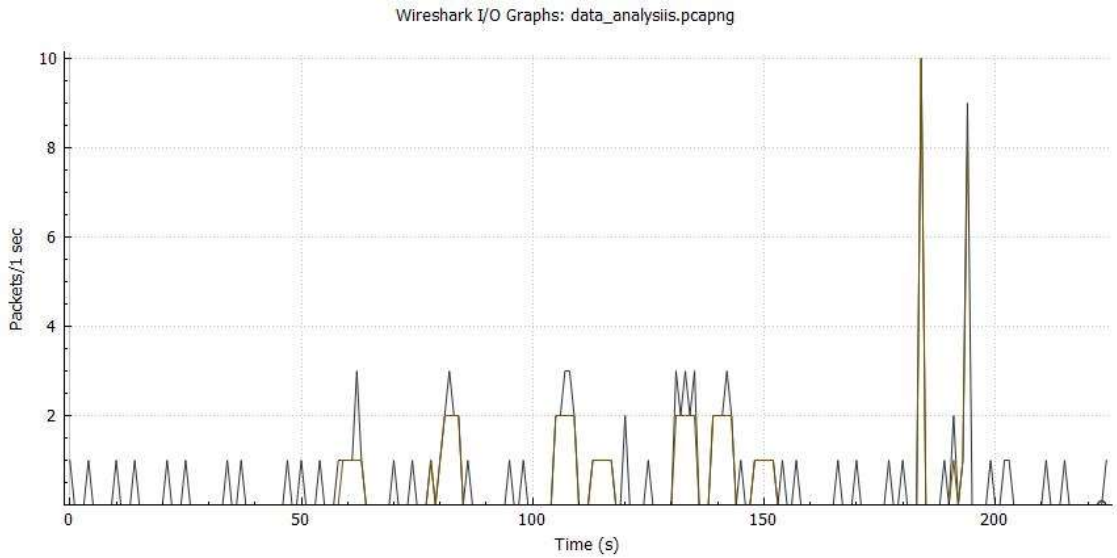



192.168.10.2
20.0.0.3
192.168.20.2
10.0.0.2
192.168.30.2
30.0.0.3
nan
20.0.0.2

- FLOW GRAPH:


Time
192.168.20.2
192.168.30.2
20.0.0.3
20.0.0.2
Comment

- I/O GRAPH:



CONCLUSION

In summary, this network traffic analysis project provided valuable insights into data flow within the network. We identified security vulnerabilities, optimized resource allocation, and gained key performance metrics for future improvements. Ongoing monitoring and updates to analysis tools will be essential for maintaining network efficiency and security.

REFERENCES

- "Practical Packet Analysis: Using Wireshark to Solve Real-World Network Problems" by Chris Sanders
- https://docs.python.org/
- https://pandas.pydata.org/pandas-docs/stable/
- https://docs.gns3.com/
- https://www.wireshark.org/docs/