1. **INTRODUCTION**

In today's digital world, communication between computers and devices over a network is essential. Every time we browse the internet, send a message, or access a website, data is exchanged between systems using different protocols and ports. Each service running on a device listens on a specific port, which acts like a virtual door to receive and send data.

This project, titled "Port Scanner using TCP Sockets", focuses on understanding how data communication works at the transport layer of the OSI model, specifically using the TCP (Transmission Control Protocol). The aim is to scan a range of ports on a target device to determine which ports are open and available for communication.

By using Python's socket programming, we simulate the real-world process of initiating a connection with another device. This scanner checks ports by attempting to establish a TCP connection. If the connection is successful, the port is open. Otherwise, it is closed or filtered. The scanner also identifies the service (like HTTP or FTP) running on the open port.

This project helps in learning about:

* Network communication basics
* How TCP establishes connections
* Security risks due to open ports
* How tools like port scanners assist in network monitoring

It also uses multithreading to make the scanning process faster and more efficient, making it a practical and educational tool for students learning data communication.

1. **OBJECTIVES**

The primary objective of this project is to design and develop a TCP Port Scanner using Python, with the goal of understanding how data communication takes place in computer networks at the transport layer. This project provides practical exposure to network programming and helps students understand how different services interact with devices over the internet or a local network.

The scanner is developed using Python's socket programming library, which allows the creation and management of network connections. The tool attempts to establish a TCP connection with various ports on a target device. Based on the response, the program determines whether a port is open, closed, or filtered. The open ports can indicate which services (such as HTTP, FTP, SSH, etc.) are running on the remote system.

This project not only introduces students to port scanning techniques but also provides insights into how TCP sockets work and how connection-oriented communication is established. It reflects real-world scenarios used by system administrators and security professionals for network monitoring and vulnerability assessments.

The specific objectives of this project are:

* To study the basics of data communication and the role of the TCP protocol in reliable transmission of data between systems.
* To create a functional port scanner that can detect open ports and report them to the user.
* To analyze how the TCP three-way handshake process works in establishing a connection.
* To implement multithreading in Python to perform faster and parallel scanning of multiple ports.
* To map open ports with commonly known services to better understand the services running on the scanned system.
* To gain knowledge about cybersecurity practices and how open ports can expose a system to potential risks if not secured.
* To develop an application that is simple to use, educational, and practical for demonstrating concepts in data communication.

1. **TOOLS AND TECHNOLOGIES USED**

This project makes use of several tools and technologies that are essential for developing and understanding how a TCP Port Scanner works. These tools not only assist in implementation but also provide a deeper understanding of the underlying concepts of data communication, socket programming, and system-level networking. Each component contributes to a different part of the project — from coding and logic building to real-time testing and analysis.

**1. Python Programming Language**

Python is a powerful, high-level, general-purpose programming language that is known for its simplicity and readability. It is especially popular in the fields of data science, automation, and networking. One of the biggest advantages of using Python in networking projects is the availability of built-in libraries for socket programming, multithreading, and data processing.

In this project, Python is the core programming language used to write the logic of the port scanner. Its easy-to-understand syntax helps in building reliable applications with fewer lines of code, which makes it beginner-friendly and suitable for academic projects.

**2. Python Socket Library**

The socket module in Python provides a low-level interface to the network. It allows the programmer to create sockets — virtual endpoints of a two-way communication link between two programs running on a network. Using sockets, we can send and receive messages or establish a connection between two devices.

In this port scanner, the socket library is used to:

* Create a socket object.
* Attempt a TCP connection to each port of a target IP address.
* Determine whether a port is open (based on success/failure of connection).
* Resolve the service running on the open port.

This helps demonstrate real-life data communication at the transport layer using TCP.

**3. Python Threading Library**

Threading is a method of running multiple tasks at the same time within a single program. In Python, the threading module enables concurrent execution of code. Without threading, the scanner would take a lot of time if scanning many ports one-by-one, especially on slower networks.

By using threading:

* Multiple ports are scanned simultaneously.
* Scanning becomes faster and more efficient.
* The program utilizes CPU resources better by handling multiple connections in parallel.

This mirrors the behavior of real-world scanning tools used by cybersecurity professionals and network administrators.

**4. IDLE (Integrated Development and Learning Environment)**

IDLE is the default IDE that comes with Python installations. It is a simple editor and debugger, perfect for students and beginners. IDLE provides features such as:

* Writing and running Python scripts (.py files).
* Viewing errors in real-time.
* Debugging with line-by-line execution.
* Interactive Python shell for testing small parts of code.

In this project, IDLE was used to write the port scanning program and run it during development and testing.

**5. Operating System (Windows)**

The development and testing of this project was done on the Windows operating system. Windows offers good support for Python and networking tools. The Command Prompt in Windows is used to run the Python scripts and see the output of the scanner in real-time.

Networking in Windows also allows us to test local and remote IPs, check firewall responses, and experiment with services that may be running on different ports.

**6. Command Line / Terminal**

Although Python can be run inside IDLE, the Command Line interface (Command Prompt or PowerShell in Windows) is used to run the program with custom input. The terminal helps:

* Input the target IP address.
* Show live scanning status.
* Print open ports and service names.
* Display errors if any.

This helps simulate real command-line tools like Nmap and Netcat that professionals use for scanning.

**7. Network Protocols and Concepts**

Even though not physical tools, the knowledge of certain protocols and models is vital in this project:

* **TCP (Transmission Control Protocol):** Reliable, connection-based protocol that establishes a proper link before sending data.
* **OSI Model:** Especially the **Transport Layer** (Layer 4), which is responsible for end-to-end communication and port management.
* **Ports and Services:** Standard ports (like 80 for HTTP, 443 for HTTPS, 22 for SSH) are used to identify specific services running on a device.
* **Three-Way Handshake:** TCP's connection establishment process involving SYN, SYN-ACK, and ACK messages.

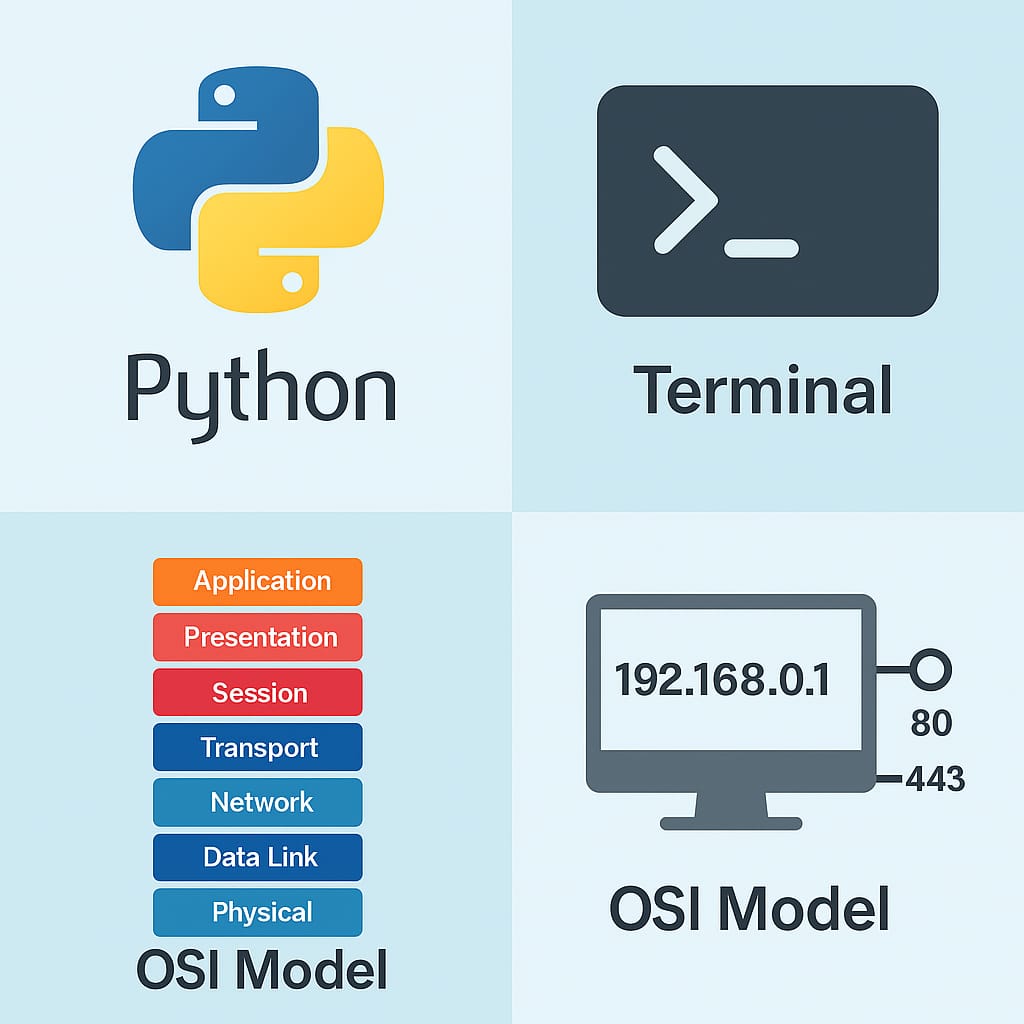


Fig 3.1 Tools Used Overview

1. **BASICS OF DATA COMMUNICATIONS**

In the digital age, data communication is the foundation of every online activity — from browsing websites to sending emails and scanning networks. Understanding how data flows between devices helps us build applications like Port Scanners, which rely on this exchange to identify open ports on remote systems.

**4.1 Components of Data Communication**

Any successful data communication system consists of five basic components

* **Sender**: This is the originator of the data. In our case, it is the user’s computer or device running the Port Scanner application. The sender initiates the process by sending data (in the form of connection requests) to the target system.
* **Receiver**: This is the destination device that receives the message. In our project, it refers to the system whose ports we are scanning. The receiver responds with an acknowledgment if a port is open, or nothing if it is closed or filtered.
* **Message**: This refers to the actual data being transmitted. For the port scanner, the message is the TCP packet sent to a specific port to check its status (open, closed, or filtered).
* **Transmission Medium**: This is the physical path or channel over which the message travels. Examples include Ethernet cables, optical fibers, or wireless signals. The quality and type of medium affect the speed and reliability of communication.
* **Protocol**: A protocol is a set of rules that both sender and receiver follow to ensure smooth data transmission. TCP/IP is the most widely used protocol in modern communication systems, and it plays a vital role in our project.

These components together ensure that data flows smoothly and accurately from the sender to the receiver.

**4.2 Types of Data Transmission Modes**

There are three modes in which data can be transmitted between devices

* **Simplex** mode is a one-way communication. Only the sender can send, and the receiver can only receive. This mode is rarely used in networking. An example of simplex communication is television broadcasting, where data is transmitted in only one direction — from the station to the user.
* **Half-Duplex** mode allows data transmission in both directions, but only one direction at a time. A good example is a walkie-talkie system, where one user speaks while the other waits to reply.
* **Full-Duplex** mode allows simultaneous data transmission and reception between devices. This is the mode used in modern communication systems, including our project. When the port scanner sends a TCP request, it also listens for a response — both actions occur at the same time.

Using full-duplex communication increases efficiency and is essential for interactive services such as online gaming, video calls, and network scanning.

**4.3 OSI (Open Systems Interconnection) Model**

The OSI model is a conceptual framework that standardizes the functions of a telecommunication or computing system into seven distinct layers. Each layer has a specific role, and they work together to handle data communication from end to end:

1. **Application Layer** – Provides services directly to the user (e.g., browsers, apps).
2. **Presentation Layer** – Translates, encrypts, and compresses data.
3. **Session Layer** – Manages sessions and controls dialogues between systems.
4. **Transport Layer** – Ensures reliable data delivery; uses TCP or UDP. This is the layer where our port scanner operates.
5. **Network Layer** – Handles logical addressing and routing (e.g., IP addresses).
6. **Data Link Layer** – Deals with physical addressing (MAC) and error detection.
7. **Physical Layer** – Transfers raw data as electrical or optical signals via cables or wireless.

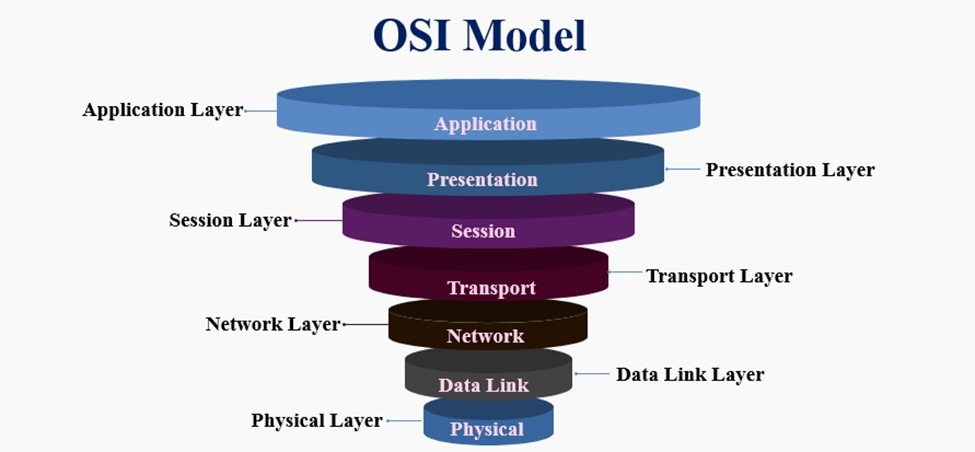


Fig 4.1 OSI Model

**4.4 TCP/IP Protocol Suite**

The TCP/IP model is a practical and simplified version of the OSI model. It is the foundation of the internet and most networks today. The four layers in this model are:

1. **Application Layer** – Provides services like HTTP, FTP, SMTP.
2. **Transport Layer** – Manages reliability and flow control (via TCP).
3. **Internet Layer** – Handles addressing and packet routing (via IP).
4. **Network Access Layer** – Deals with physical transmission and error handling.

In our project:

* The **Transport Layer** is responsible for checking if ports are open using TCP.
* The **Internet Layer** ensures that data reaches the correct IP address of the target device.

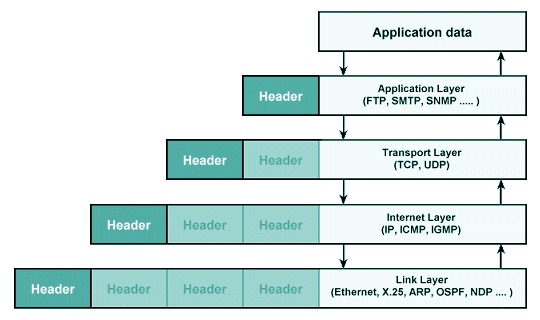


Fig 4.2 TCP/IP Protocol Stack

**4.5 Ports and Their Role in Communication**

A **port** is a numerical label assigned to specific services or applications running on a system. Think of a computer like a building and ports as doors — each door leads to a different room (service).

* Port **80** is used for HTTP (web browsing)
* Port **443** is for HTTPS (secure web browsing)
* Port **21** is for FTP (file transfer)
* Port **22** is for SSH (remote login)

Our port scanner sends a request to each door (port). If a service is running and listening, the door opens, and we receive a response. If no service is running, we get no reply or a rejection.

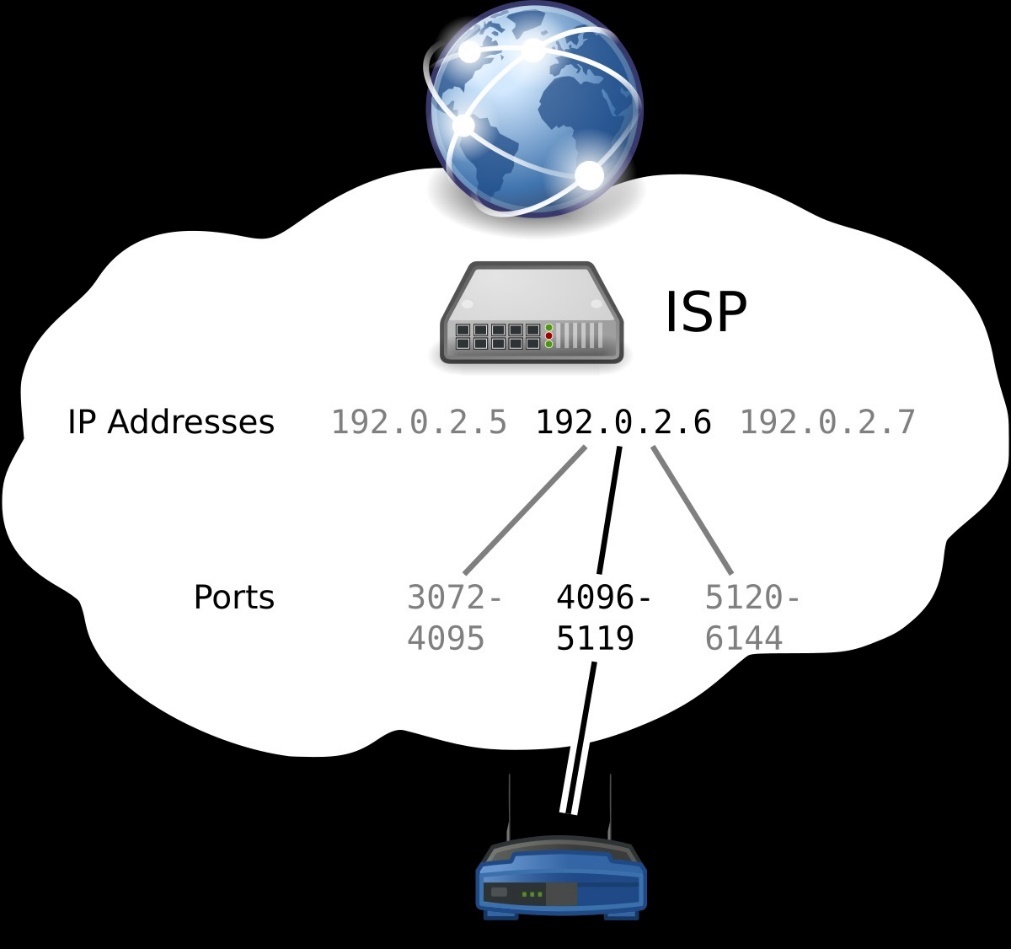


Fig 4.3 IP Address and Ports

1. **UNDERSTANDING PORTS AND SOCKETS**

Ports and sockets are two of the most essential components in computer networks, and they form the backbone of how data is communicated between different systems. Understanding how they work is critical, especially in the context of our project — *Port Scanner using TCP Sockets* — where communication is established and tested using these two elements.

A **port** in networking is a numerical identifier assigned to specific processes or services on a device. Think of a computer like a large building, and ports as doors. Each service running on the computer (like a web server, mail server, or remote login) has its own unique door or port number through which it communicates. Ports help a system understand where to send incoming data, and which application should receive it.

Each port is associated with a number from **0 to 65535**. These are divided into:

* **Well-Known Ports (0–1023)**: Reserved for essential internet services. For example, port 80 is used for HTTP (web pages), port 443 for HTTPS (secure web pages), and port 22 for SSH (remote access).
* **Registered Ports (1024–49151)**: Used by software companies for specific applications.
* **Dynamic or Private Ports (49152–65535)**: Temporary ports often assigned to client programs for short-term communications.

A **socket** is a combination of an **IP address** and a **port number**. It serves as an endpoint for sending and receiving data. A socket essentially says, "connect to this device at this specific service." For example, a socket like **(192.168.1.1, 80)** represents a connection to a computer at IP address 192.168.1.1 on port 80 — most likely a web server.

Sockets are a key part of the TCP/IP protocol, which governs how devices communicate over the internet. In our project, we use TCP sockets, which are connection-oriented, meaning the scanner attempts to establish a proper connection before transmitting data. If the connection is successful, it means that the target port is open and accepting connections — indicating that a service is active on that port.

When the scanner tries to connect to a port and gets a response, that is an example of real-time data communication. The scanner (client) sends a request to the target system (server) via a socket, and the server either accepts or denies the connection. Each of these attempts is an interaction of data being transmitted over the network.

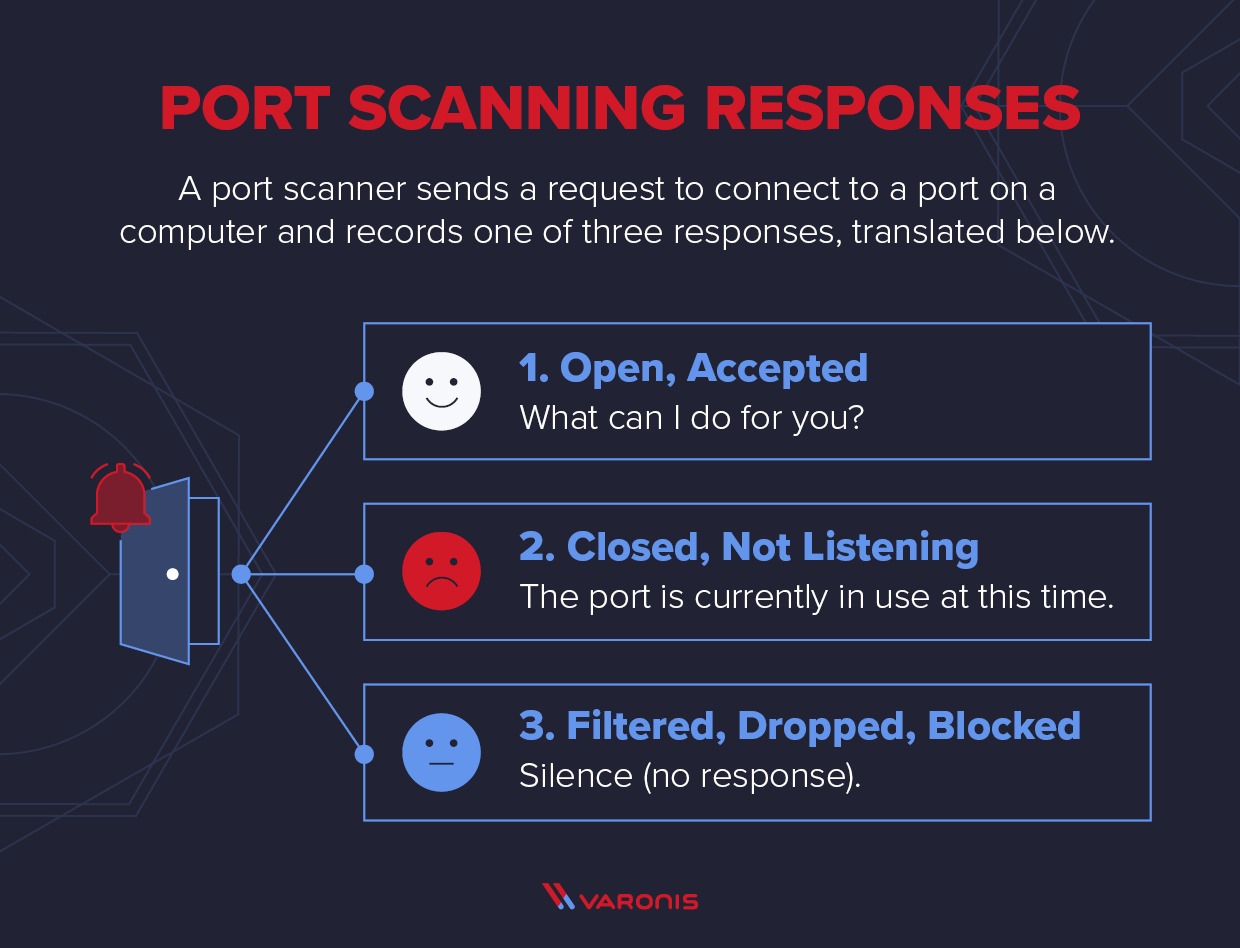
 From a cybersecurity point of view, open ports can be vulnerabilities if they are not secured properly. By scanning ports, network administrators can find out which services are exposed to the internet and whether they should be protected or disabled.

Fig 5.1 How a Port Scanner Works

1. **TCP PROTOCOL AND HANDSHAKE PROCESS**

The **Transmission Control Protocol (TCP)** is one of the primary protocols within the Internet protocol suite and plays a central role in enabling reliable, connection-oriented communication across networks. It ensures that data packets are transmitted in sequence, without duplication, and with mechanisms to detect and recover from errors. This makes TCP particularly suitable for applications where accurate and complete delivery of data is critical, including web browsing, file transfers, and secure communications.

In this project Port Scanner using TCP Sockets the TCP protocol is used to probe and determine the status of ports on a target system. The scanner sends connection requests to various TCP ports using sockets. The response, or lack thereof, helps to classify ports as **open**, **closed**, or **filtered**. The underlying process that enables this classification is known as the **TCP three-way handshake**.

**TCP Three-Way Handshake**

The TCP three-way handshake is a well-defined sequence of message exchanges that establishes a reliable connection between two devices over a network. It ensures synchronization between the communicating parties and sets the stage for controlled, orderly data transfer. The process consists of the following stages:

1. **SYN(Synchronize):**  
   The client initiates communication by sending a TCP packet with the SYN flag set to the server. This packet includes an initial sequence number that will be used for tracking transmitted data.
2. **SYN-ACK(Synchronize-Acknowledge):**  
   Upon receiving the SYN packet, the server responds with a packet that has both the SYN and ACK flags set. This indicates that the server acknowledges the receipt of the client's request and is willing to proceed with the connection. The server also includes its own sequence number.
3. **ACK(Acknowledge):**  
   Finally, the client sends an ACK packet, confirming the server's response. After this packet is received, a TCP connection is fully established, and data transmission can commence.

This handshake ensures mutual agreement between the client and server before any actual data is transferred, thereby reducing the chances of communication errors.

**Application in Port Scanning**

The handshake mechanism is essential to the operation of a TCP-based port scanner. In our implementation, the scanner (acting as the client) attempts to initiate a handshake with each port on the target system:

* If the port is **open**, the server responds with a SYN-ACK, indicating it is ready to establish a connection. The scanner recognizes this and marks the port as open.
* If the port is **closed**, the server typically responds with a RST (reset) packet, or it may not respond at all, depending on its configuration.
* If the port is **filtered**, perhaps by a firewall or security device, the request may be silently dropped, resulting in a timeout or lack of response.

This classification allows administrators and users to understand which services are accessible on the network, and where potential security risks or misconfigurations might exist.

Through this project, the theoretical concepts of data communication are translated into a working tool. By implementing a scanner that utilizes TCP sockets and mimics real-world client-server interactions, this project demonstrates practical application of fundamental network communication principles. It not only facilitates port-level diagnostics but also serves as a tool for learning and understanding protocol behavior at a deeper level.

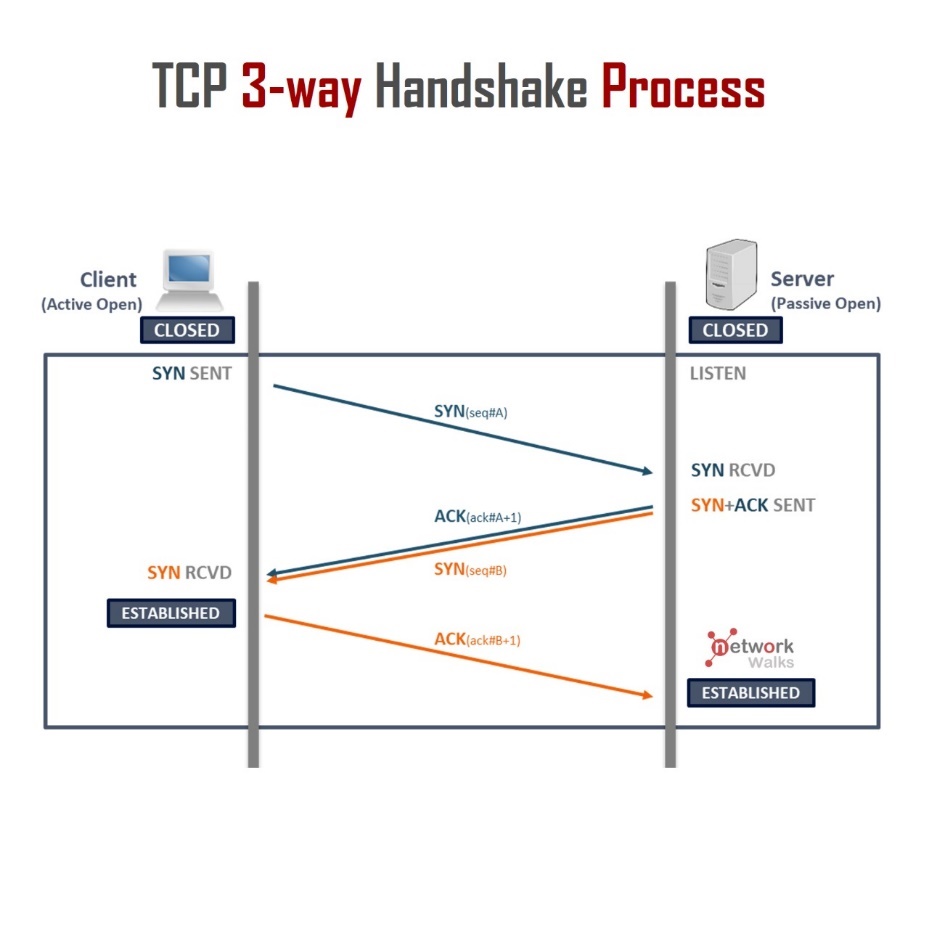


Fig 6.1 TCP Three-Way Handshake

1. **PROJECT DESIGN AND ARCHITECTURE**

**7.1. Introduction**

The TCP Socket Port Scanner is a network diagnostic tool designed to identify open ports on a target machine by attempting TCP connections over a specified range of ports. This helps in assessing network security by revealing services available on the host and potential vulnerabilities.

**7.2. System Architecture Overview**

The architecture of the TCP Port Scanner system is modular and layered to ensure clarity, maintainability, and scalability. It consists of the following key components:

* **User Interface (UI) Module**
* **Port Scanning Controller Module**
* **TCP Socket Connection Module**
* **Results Processing and Reporting Module**

Each module plays a specific role in the overall functionality of the system and interacts with others in a streamlined manner.

**7.3. Module Descriptions**

**a) User Interface (UI) Module**

The User Interface acts as the point of interaction between the user and the system. It allows the user to input parameters such as the target IP address and the range of ports to scan. The UI then initiates the scanning process and displays the results in a readable format.

* Input: Target IP, Start Port, End Port
* Output: Real-time display of open ports and summary report

**b) Port S canning Controller Module**

This module controls the scanning operation. It manages the iteration over the specified port range and coordinates connection attempts. It also handles timeout settings and manages any exceptions to ensure smooth execution.

* Input: IP and port range from UI
* Output: Port status (open/closed) for each port scanned

**c) TCP Socket Connection Module**

The core functional module responsible for attempting TCP connections to each port on the target machine. Using TCP socket programming, it tries to establish a connection. If successful, the port is marked open; otherwise, it is considered closed or filtered.

* Utilizes socket programming with timeout to avoid indefinite waits

**d) Results Processing and Reporting Module**

This module collects and processes the scanning results. It aggregates open ports, formats the output, and presents the findings to the user through the UI.

**7.4. Workflow**

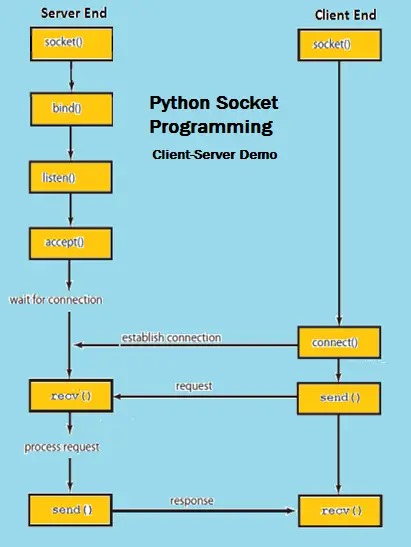
1. **Input Collection**: User provides the target IP address and port range.
2. **Scan Initialization**: The controller begins scanning each port sequentially or concurrently.
3. **Connection Attempts**: The TCP Socket module attempts to connect to each port.
4. **Result Evaluation**: Each connection attempt’s result is evaluated to determine port status.
5. **Reporting**: Results are displayed in real-time and compiled into a final report.
6. **Completion**: The system signals the completion of the scan.

Fig 7.1 Python Socket Program Flow

1. **PROGRAM CODE EXPLANATION**

import socket

import threading

def scan\_port(ip, port):

s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

s.settimeout(1)

try:

s.connect((ip, port))

try:

service = socket.getservbyport(port)

except:

service = "Unknown"

print(f"[OPEN] Port {port} is open ({service})")

except:

pass

finally:

s.close()

def start\_scan(ip, start\_port, end\_port):

print(f"Scanning {ip} from port {start\_port} to {end\_port}...\n")

for port in range(start\_port, end\_port + 1):

t = threading.Thread(target=scan\_port, args=(ip, port))

t.start()

# You can change these

target\_ip = "127.0.0.1" # This means your own computer

start\_port = 1

end\_port = 100

start\_scan(target\_ip, start\_port, end\_port)

The program is designed to perform a TCP port scan on a specified target IP address over a defined range of ports. It uses Python’s socket programming capabilities alongside multi-threading to efficiently and concurrently scan multiple ports. The following describes the structure and functionality of the code in detail.

**8.1 Libraries Used**

* **socket**: This module provides an interface for network communication via TCP sockets. It is used to attempt connections to target ports to check their status.
* **threading**: This module allows the program to create and manage multiple threads, enabling concurrent execution of port scans to reduce the total scanning time.

**8.2 Function: scan\_port(ip, port)**

This function attempts to establish a TCP connection to the given IP address and port number.

* It creates a TCP socket object configured with a timeout of one second to prevent the program from hanging on unresponsive ports.
* It then tries to connect to the specified port.
* If the connection is successful, the function tries to identify the common service running on the port using socket.getservbyport(port). If no standard service is found, it labels it as “Unknown.”
* Upon successful connection, it prints a message indicating that the port is open along with the identified service.
* If the connection attempt fails or times out, the function silently ignores the exception and moves on.
* Finally, it closes the socket connection to free system resources.

**8.3 Function: start\_scan(ip, start\_port, end\_port)**

This function manages the overall scanning process.

* It first prints a message indicating the IP address and port range being scanned.
* It iterates over the range of ports specified by the user.
* For each port, it creates a new thread that calls the scan\_port function.
* Each thread runs independently and concurrently, allowing multiple ports to be scanned simultaneously.
* This concurrent scanning approach significantly improves performance over a sequential scan.

**8.4 Program Execution Flow**

* The program’s parameters such as the target IP address (target\_ip) and port range (start\_port and end\_port) are defined at the start.
* The start\_scan function is called with these parameters to initiate the scan.
* As each thread completes a successful connection, it outputs the status of the port in real-time.
* The program continues until all ports in the specified range have been scanned.

1. **RESULTS AND OUTPUT**

**9.1 Test Environment**

* **Target IP Address:** 127.0.0.1 (localhost)
* **Port Range Scanned:** 1 to 100
* **System:** Personal computer running the Python script
* **Network Status:** Local machine with standard network services running

**9.2 Sample Output**

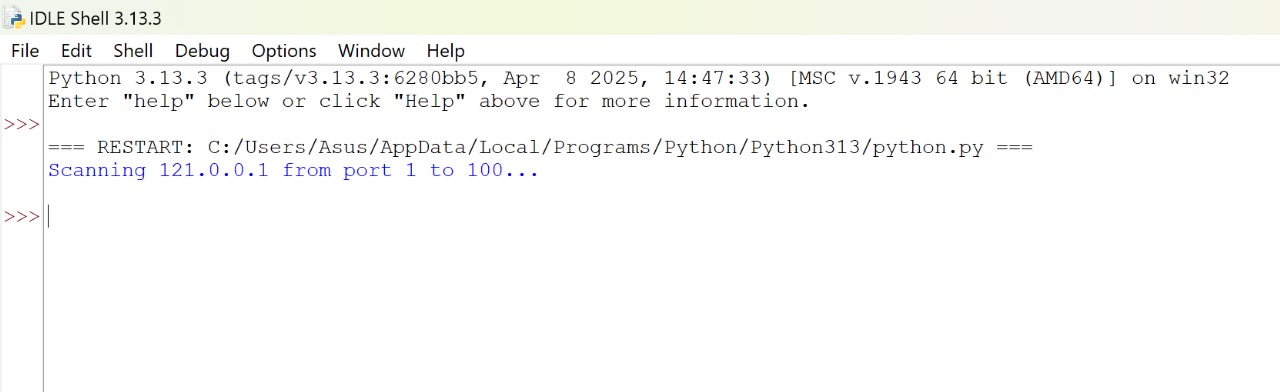
 Upon running the program, the scanner initiated concurrent connection attempts across the specified ports. The console displayed the following type of output for ports detected as open:

Fig 9.1 Sample Output 1

Ports for which the connection was refused or timed out were not displayed, ensuring the output was concise and focused only on open ports.

**9.3 Analysis of Results**

* **Open Ports Detected:** The program successfully identified open ports within the scanned range, along with their standard services where applicable.
* **Performance:** The use of multi-threading significantly reduced the total scan time, completing the scan in a few seconds instead of several minutes.
* **Accuracy:** The port status was correctly identified based on TCP connection success, verified by manual checks using other network tools such as netstat and nmap.
* **Service Identification:** The scanner correctly mapped well-known ports to their common services, providing useful information beyond simple port status.

**9.4 Observations**

* Ports that are closed or filtered did not produce output, keeping the results focused and uncluttered.
* The scanner’s timeout setting ensured that slow or non-responsive ports did not delay the overall scan.
* The program operated without crashing or hanging, demonstrating effective error handling and resource management.

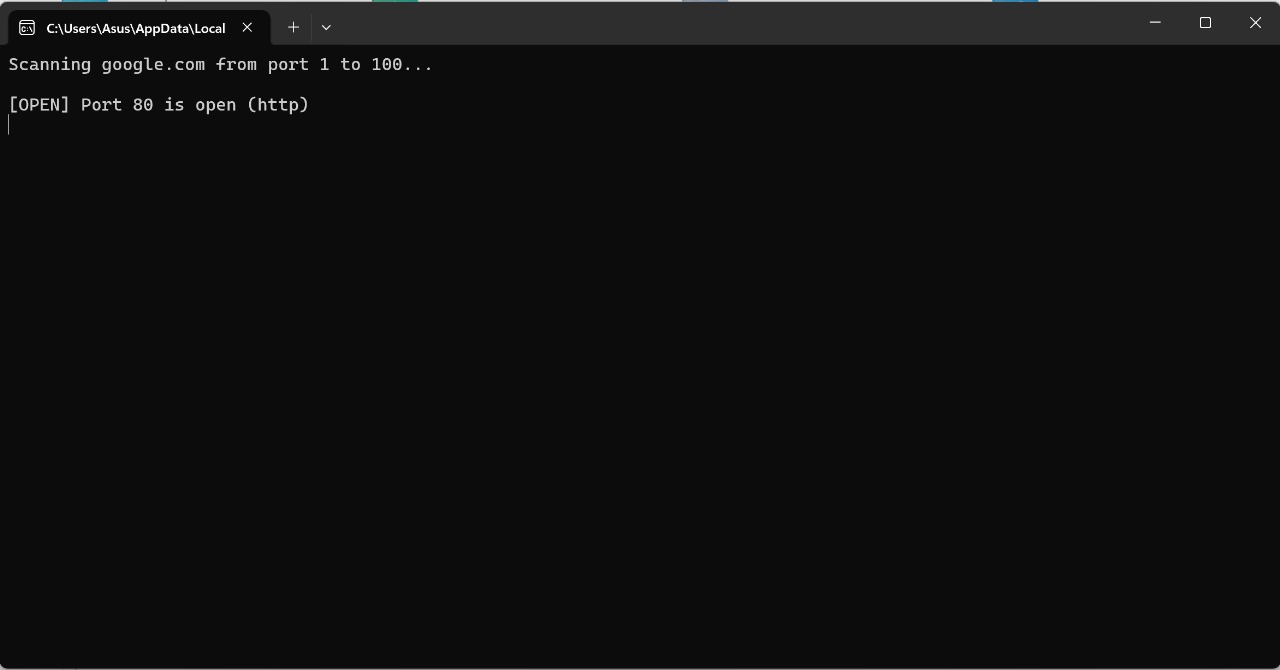


Fig 9.2 Sample Output 2

1. **CONCLUSION**

The TCP Socket Port Scanner project successfully demonstrates the fundamental concepts of network communication and socket programming. By implementing a multi-threaded port scanning tool, the project efficiently identifies open TCP ports on a target machine, providing valuable insights into the network services that are accessible.

The use of concurrent threads significantly improves scanning speed, making the tool practical for scanning large port ranges within a reasonable time frame. Additionally, the integration of service identification for open ports enhances the usefulness of the scan results by associating ports with common network services.

Throughout the development and testing phases, the program exhibited reliable performance with robust error handling and timeout management, ensuring smooth operation without interruptions or delays.

This project not only reinforces the practical application of TCP sockets and threading in Python but also lays a solid foundation for further enhancements, such as adding UDP scanning, banner grabbing, or a graphical user interface to improve usability.

In conclusion, the TCP Socket Port Scanner serves as an effective tool for network analysis and security auditing, demonstrating core principles of data communication and network programming.

**REFERENCES**

1. Python Software Foundation, Python Socket Programming HOWTO, <https://docs.python.org/3/howto/sockets.html>

2. <https://docs.python.org/3/howto/sockets.html>

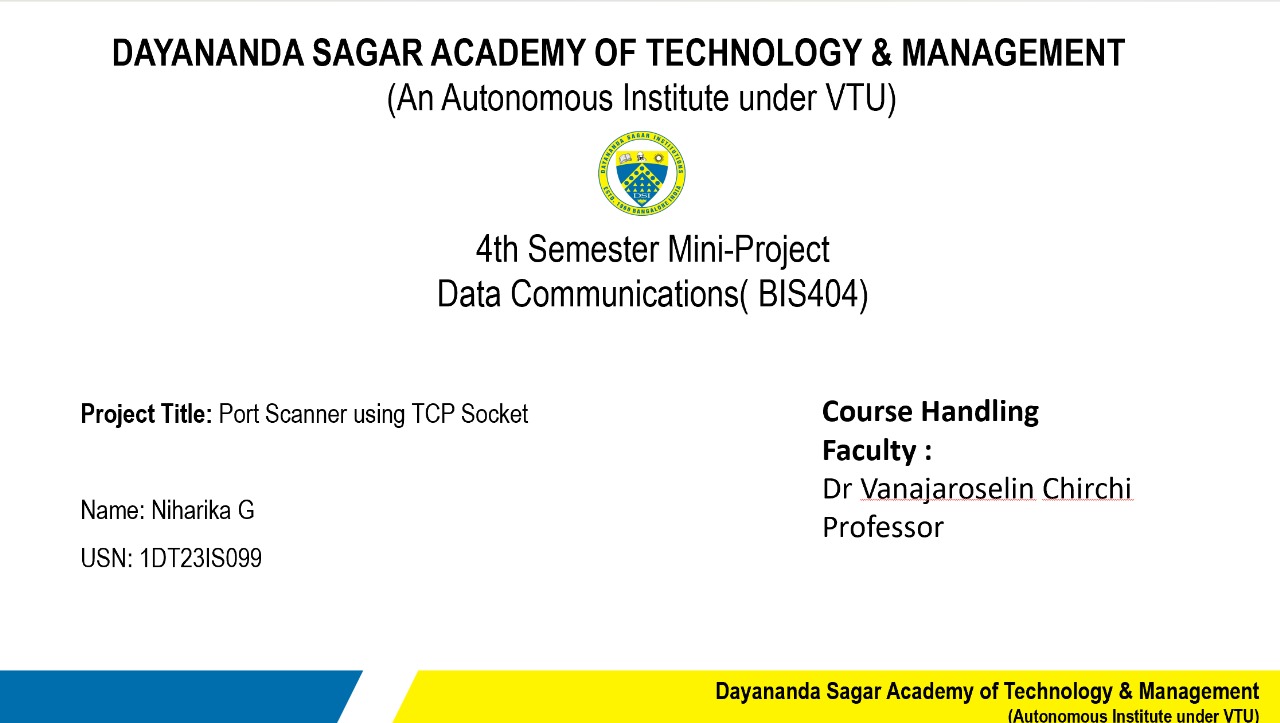
3. RFC 793, Transmission Control Protocol, J. Postel, 1981. <https://tools.ietf.org/html/rfc793> , Accessed May 2025.

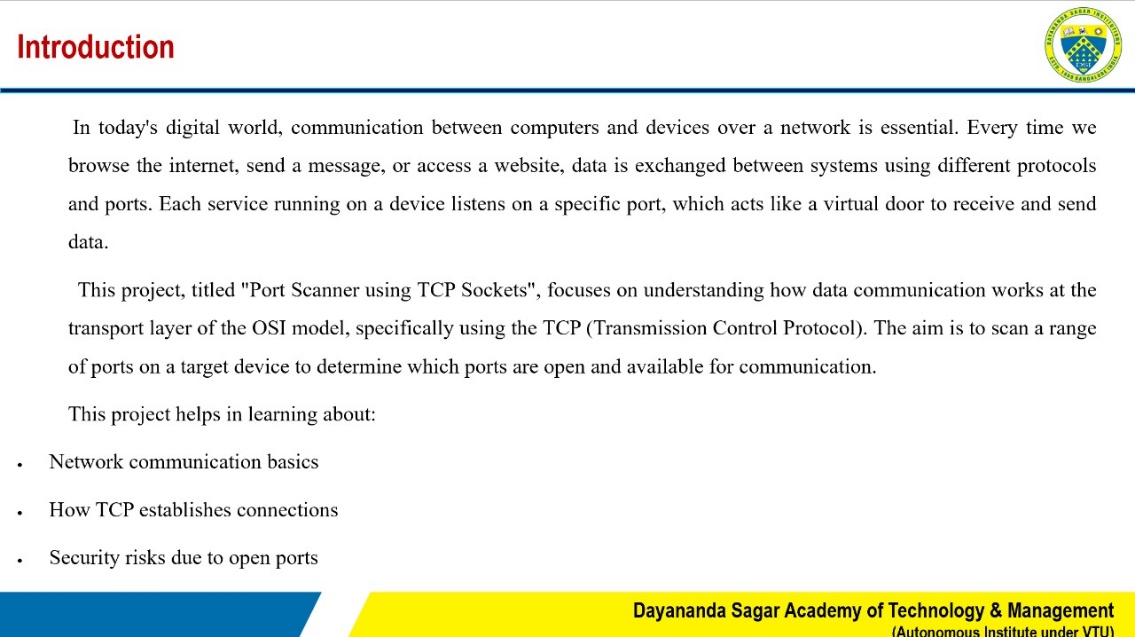
4. Python Official Documentation, threading — Thread-based parallelism, <https://docs.python.org/3/library/threading.html> , Accessed May 2025.

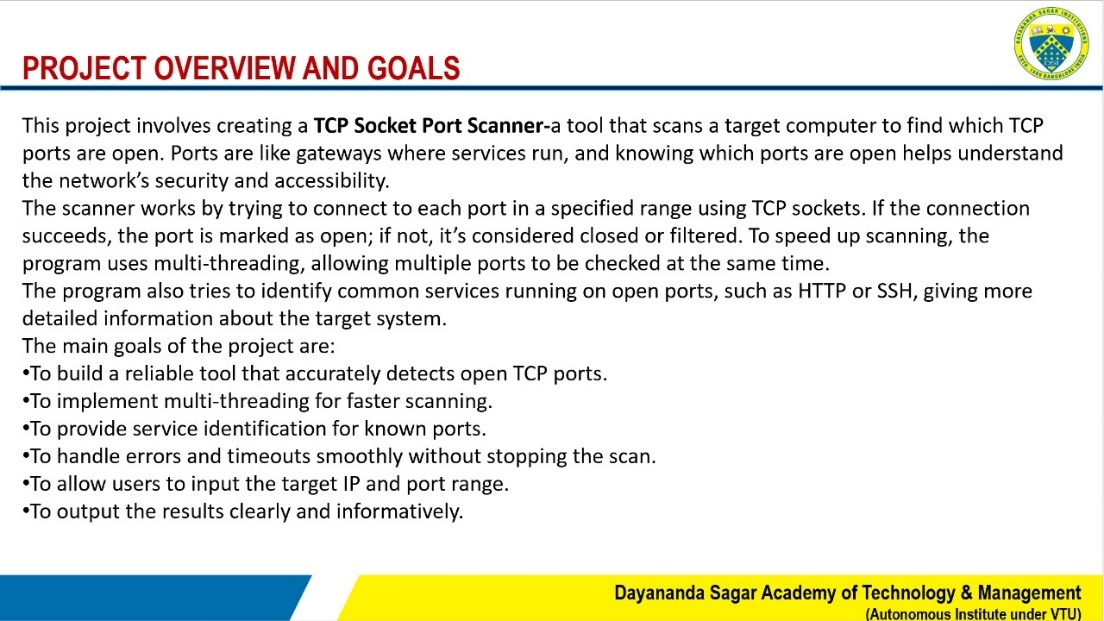
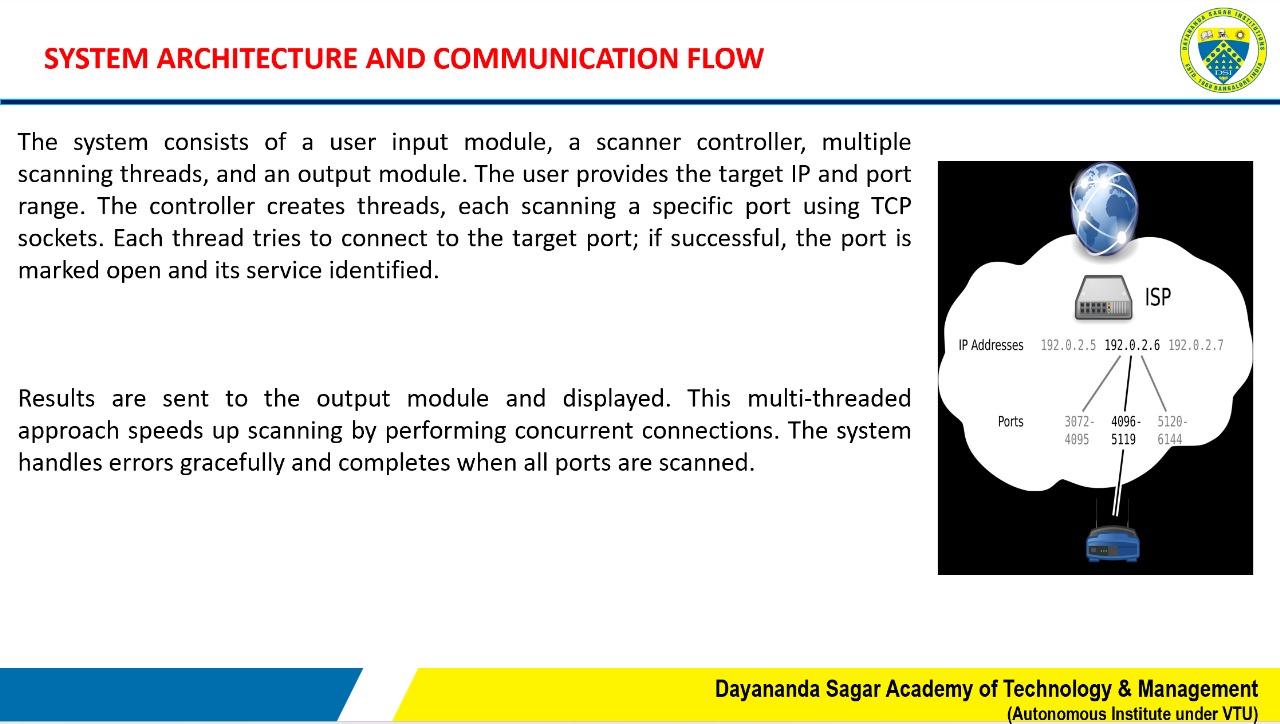
5. Kurose, J. F. and Ross, K. W., Computer Networking: A Top-Down Approach, 8th Edition, Pearson, 2021.

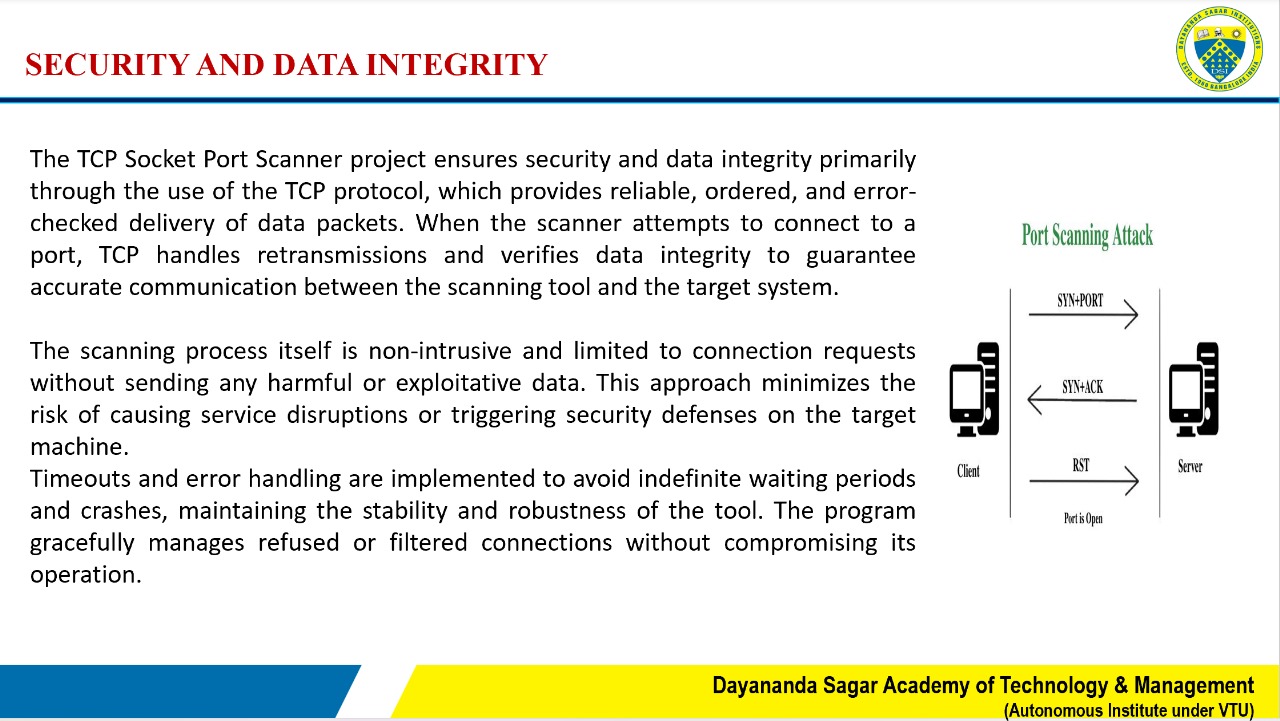
6. Nmap Project, Nmap Network Scanning, <https://nmap.org/book/> .

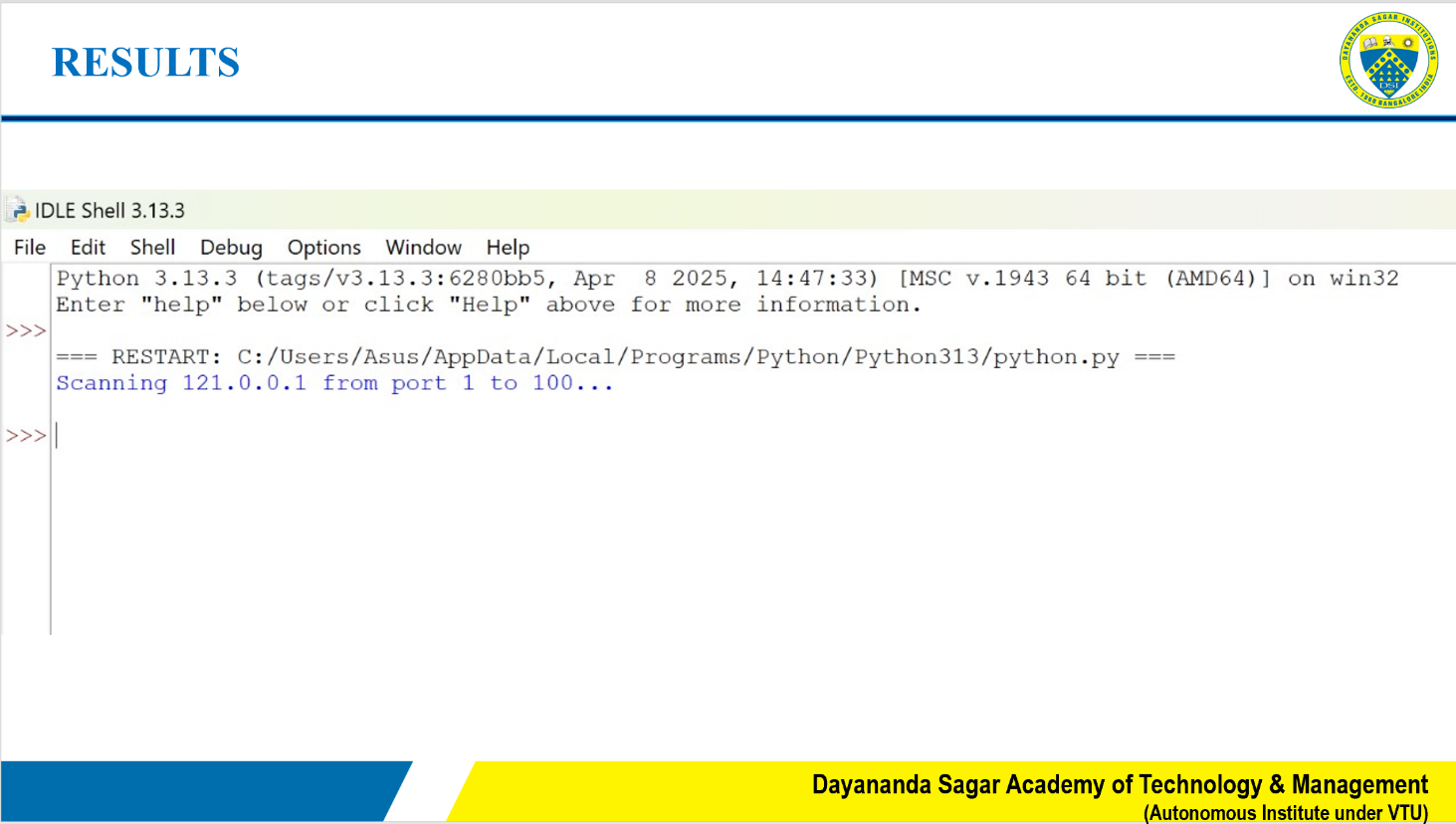
**PPTs**

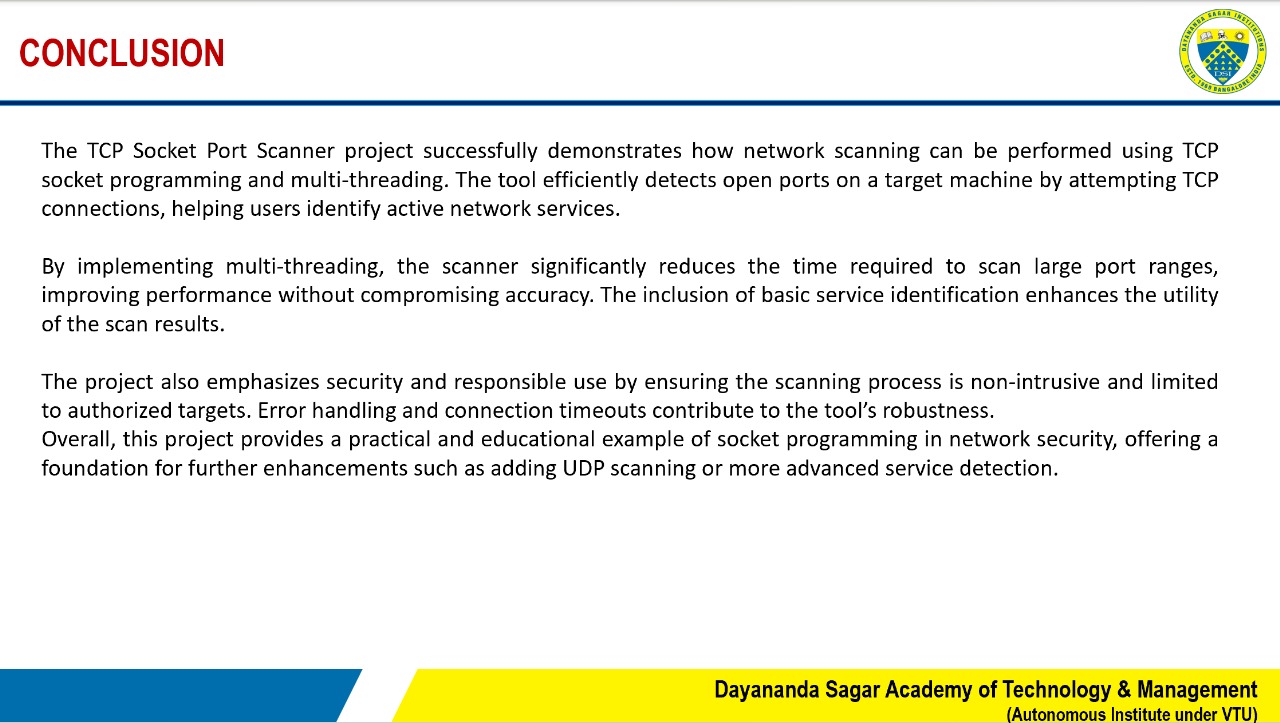


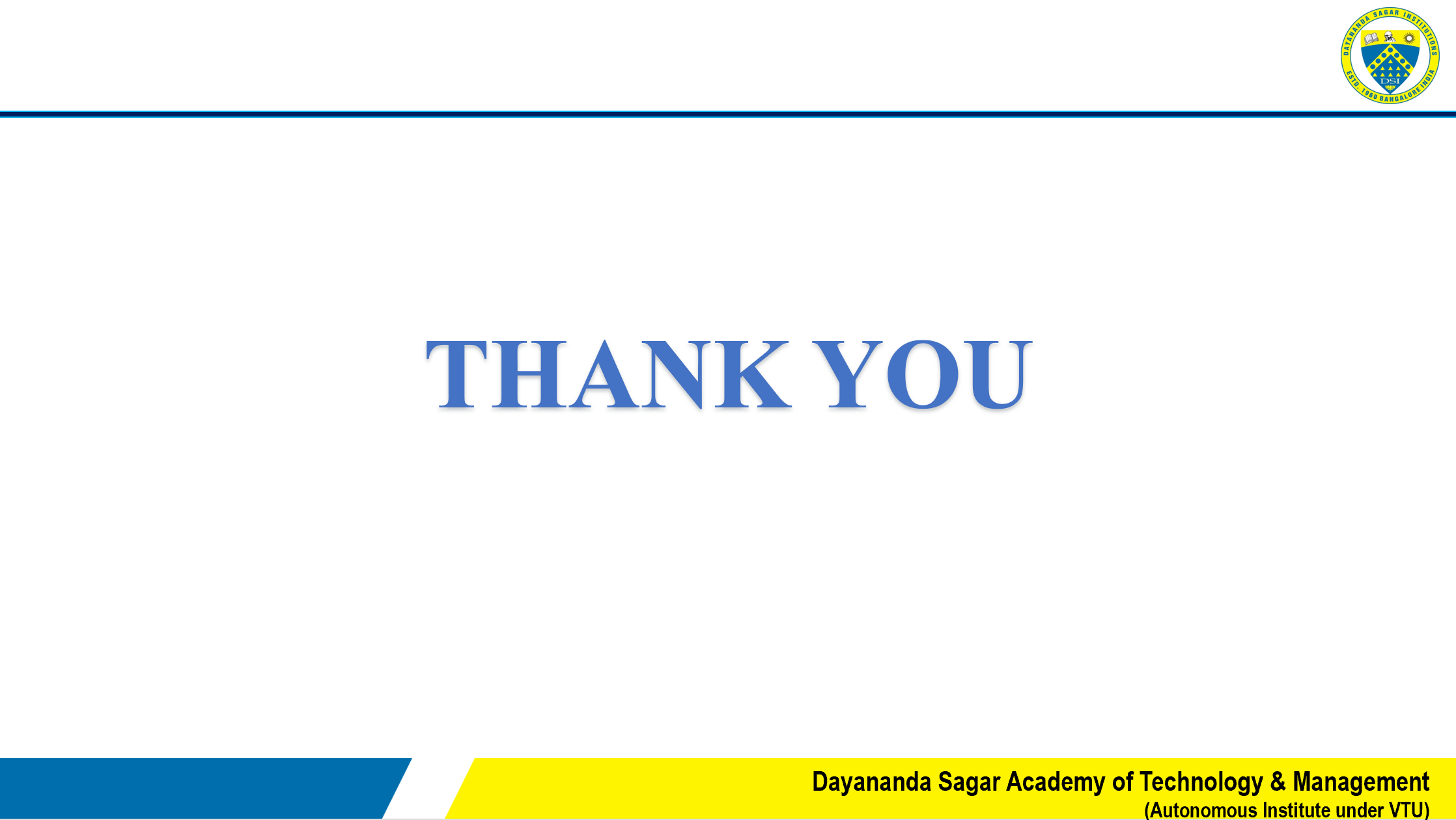






****



****

****