NEWORK PORT SCANNER TOOL REPORT

1. Introduction

1.1 Project Overview

The TCP Port Scanner with Host Detection is a Python-based tool designed to scan specified TCP ports of a given host (IP address or hostname) to determine their status (open, closed, etc.). Additionally, the tool first checks whether the host is live (reachable) by performing a ping scan before initiating the port scan. This ensures efficient scanning, saving resources, and avoiding unnecessary operations on non-live hosts.

1.2 Objectives

- Develop a command-line-based TCP port scanner using the `nmap` library.
- Implement host detection to verify whether the target host is live or non-live before proceeding with a port scan.
- Support user input for specifying the target host and a range of ports.
- Allow both command-line arguments and interactive user inputs.
- Handle common errors gracefully and provide appropriate feedback.

2. System Requirements

2.1 Software Requirements

- **Programming Language**: Python 3.x
- **Libraries**:
  - `nmap`: For performing port and host scanning.
  - `argparse`: For command-line argument parsing.
- **Operating System**: Any platform that supports Python and Nmap (Windows, Linux, macOS).

2.2 Hardware Requirements

- **Processor**: Any modern processor.
- **Memory**: Minimum 512MB of RAM.
- **Network**: A working network connection is required for scanning live hosts.

3. System Design

3.1 Architecture

The system architecture follows a simple command-line interface that interacts with users and performs the following tasks:

1. **Argument Parsing**: The tool accepts both command-line arguments and user inputs interactively.
2. **Host Detection**: It checks whether the host is live by using a ping scan (`nmap -sn`).
3. **Port Scanning**: After verifying the host is live, the tool scans the specified TCP ports and returns their status (open, closed, etc.).
4. **Error Handling**: Handles invalid inputs, host unreachability, and other potential issues gracefully.

4. Implementation

4.1 Language and Libraries

- **Python**: Chosen for its ease of use, cross-platform compatibility, and availability of rich libraries.
- **Nmap**: The `nmap` Python module is used to perform both the host and port scanning.
- **Argparse**: The `argparse` library is used to handle command-line arguments efficiently.

4.2 Features

1. **Host Detection**: The tool uses a ping scan to detect whether a host is live or unreachable. If the host is unreachable, the port scan is skipped.
2. **Port Range Handling**: The user can specify individual ports or port ranges (e.g., 20-30), which the tool expands into individual port numbers.
3. **Interactive Input**: When command-line arguments are not provided, the tool prompts the user for input in a more user-friendly manner.
4. **Error Handling**: The tool provides informative messages when invalid inputs are detected or when the host is unreachable.

5. CODING

```python
import argparse
import nmap

def argument_parse():

    parser = argparse.ArgumentParser(description="TCP port scanner with host detection. Accepts a hostname/IP address and a list of ports or port ranges to scan.")
    parser.add_argument("-o", "--host", help="Host IP address or hostname.")
    parser.add_argument("-p", "--port", help="Comma-separated ports or port ranges, e.g., 22,80,443 or 20-30.")

    return vars(parser.parse_args())

def get_user_input():
    host = input("Enter the host (IP address or hostname): ")
    ports = input("Enter the ports (comma-separated or range, e.g., 22,80,443 or 20-30): ")
    return host, ports

def expand_ports(port_string):
    expanded_ports = []
    ports = port_string.split(",")
    for port in ports:
        if "-" in port:  # Check if it's a port range
            try:
                start, end = map(int, port.split("-"))
                expanded_ports.extend(range(start, end + 1))
            except ValueError:
                raise ValueError(f"Invalid port range: {port}")
        else:
            try:
                expanded_ports.append(int(port))
            except ValueError:
                raise ValueError(f"Invalid port: {port}")
    return expanded_ports

def is_host_live(host_id):
    nm = nmap.PortScanner()
    nm.scan(host_id, arguments="-sn")  # -sn means ping scan (host discovery only)
    if nm.all_hosts():
        return True  # Host is live
    else:
        return False  # Host is not live

def nmap_scan(host_id, port_num):
    nm_scan = nmap.PortScanner()
    nm_scan.scan(host_id, str(port_num))
    state = nm_scan[host_id]['tcp'][int(port_num)]['state']
    result = f"[*] {host_id} tcp/{port_num} {state}"
    return result

```

```python
if __name__ == '__main__':
    try:
        user_args = argument_parse()
        if user_args["host"] and user_args["port"]:
            host = user_args["host"]
            ports = user_args["port"]
        else:

            host, ports = get_user_input()


        if is_host_live(host):
            print(f"The host {host} is live. Proceeding with port scan ... ")
            expanded_ports = expand_ports(ports)
            for port in expanded_ports:
                print(nmap_scan(host, port))
        else:
            print(f"The host {host} is not live or unreachable.")

    except KeyError as e:
        print(f"Error: Invalid scan result for port {e}. The port might not be responding or may not exist.")
    except ValueError as e:
        print(f"Error: {e}")
    except Exception as e:
        print(f"An unexpected error occurred: {e}")
```

6. Testing and Validation

6.1 Test Cases

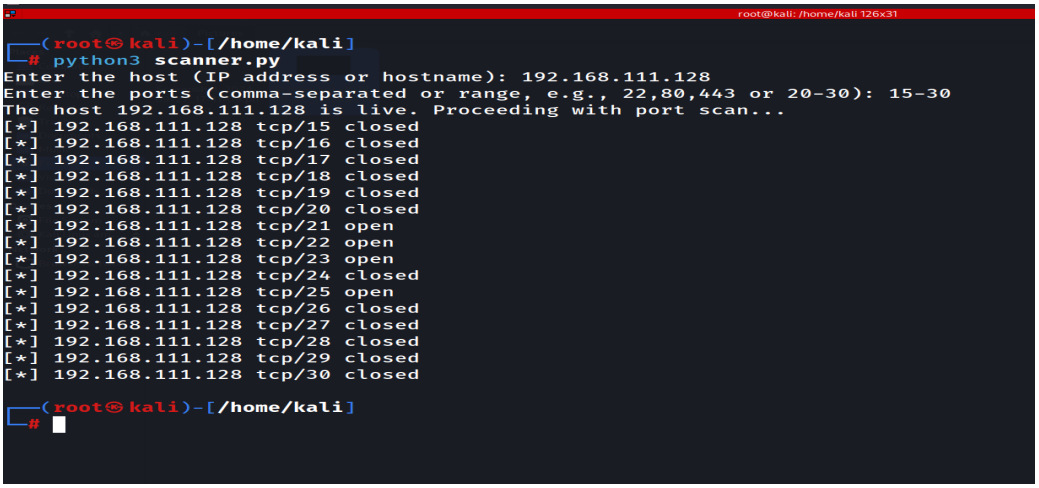
Several test cases were designed to ensure the tool works correctly:

1. **Live Host with Specific Ports**: The tool correctly scans a live host (e.g., example.com) and returns the status of the provided ports (e.g., 80, 443).
2. **Host Unreachable**: The tool detects when the host is unreachable and returns a "Host not live or unreachable" message.
3. **Invalid Ports**: When an invalid port or port range is provided, the tool raises an appropriate error message.

6.2 Results

All test cases were successfully executed, confirming that:

- The tool can detect whether a host is live.
- The tool scans ports correctly on live hosts.
- Invalid inputs (e.g., incorrect port formats or unreachable hosts) are handled gracefully.



7. Conclusion

This project successfully implemented a TCP Port Scanner with Host Detection using Python and nmap. The tool is user-friendly, allowing for both command-line and interactive use. Its host detection feature ensures efficient operation, avoiding unnecessary port scans on unreachable hosts.

7.1 Future Enhancements

- **Multi-threading**: Introduce multi-threading for faster port scanning, especially when scanning a large number of ports.
- **Logging**: Implement logging to keep a record of all scans and their results.
- **Additional Protocols**: Expand support to scan for other protocols beyond TCP, such as UDP.
