Network Packet Sniffer

Introduction

A network packet sniffer is a tool used to capture, analyze, and log network packets to understand the data flow, monitor network performance, and troubleshoot issues. This project implements a network packet sniffer in C++ for macOS, capturing data packets on the network interface and analyzing various protocols, including Ethernet, IP, TCP, UDP, and ICMP. The sniffer inspects packet headers and payloads, logging essential information such as source and destination addresses, protocol details, and any errors encountered during packet transmission. By simulating packet transfers and recording network data, this project serves as an educational resource for understanding network protocols and packet analysis fundamentals.

The core functionality is built around the `ProcessPacket` function, which parses incoming packets, identifies the protocol type, and processes it accordingly. The project uses socket programming and low-level packet capturing techniques to obtain packets in real-time. Captured packets are then logged to a file for further analysis, providing insights into the network's behavior and performance.

Working Methods

1. Socket and Packet Capture

- The program opens a socket using `pcap_open_live`, allowing real-time capture of packets from a specified network interface. Each packet is processed as it arrives, with the callback function `ProcessPacket` handling packet analysis.
- A wrapper function `Recvfrom` is used to capture packets in the socket, ensuring error handling in case of data retrieval issues. The `close` function terminates the socket once packet capture is complete, releasing network resources.

2. Packet Processing and Protocol Handling

- The main processing logic is handled within `ProcessPacket`, which uses the IP header to identify the protocol of each packet (TCP, UDP, ICMP, or others).
- For each protocol, a dedicated function (e.g., `print_tcp_packet`, `print_udp_packet`, or `print_icmp_packet`) extracts and logs detailed packet information, including headers and payloads.
- Additional protocol-specific functions include `print_ethernet_header` and `print_ip_header`, which log Ethernet and IP header details respectively. These functions display key data such as source and destination MAC addresses, IP addresses, and protocol IDs, providing a structured view of each captured packet.

3. Logging and Data Storage

- All packet details are logged in a text file (`log.txt`). This includes source and destination addresses, protocol types, and packet sizes.

- `logPacketInfo` writes structured information for each packet, including error handling if the log file is inaccessible. This logging mechanism offers a persistent record of network activity, supporting performance monitoring and troubleshooting.

4. Error Handling and Data Formatting

- Error handling is integrated into each function, with messages logged for issues encountered during socket operations (`Socket Error`, `Recvfrom Error`, etc.). This ensures stable operation even when network interruptions occur.
- The `intToIp` function converts integer IP addresses into a readable dot-decimal format, making log entries easier to interpret.

Software Tools and Header Files

1. Key Software Tools

- **PCAP Library**: The sniffer uses the `libpcap` library, a widely used tool for packet capture in Unix-based systems, to capture packets at the network level. It allows the program to receive all network packets without filtering, essential for protocol analysis.
- **C++ Standard Library**: Various standard library components, such as file I/O (`std::ofstream`) for logging, string manipulation (`std::string`), and console output, are used to manage data and log output efficiently.

2. Custom Header Files

- **main.hpp**: This header file includes all function declarations and external libraries required by the main program. It organizes the primary packet capture and analysis code, simplifying program structure and readability.
- **Logger.h**: Defines logging functions to store network activities in a persistent log file (`log.txt`). Functions in this file, such as `logPacketInfo`, provide structured logging for captured packet data, enhancing analysis and debugging capabilities.
- **PacketStructures.h**: Contains structure definitions for different network protocols, including Ethernet, IP, TCP, UDP, and ICMP. These structures make it easy to parse packet headers and identify protocol-specific fields for logging.
- **Algorithms.h** (if implemented): Can include any pathfinding or shortest path algorithms used for packet routing within the simulated network graph. Algorithms like Dijkstra's may be relevant if the sniffer is expanded to include network simulation features.

3. Key Libraries and Dependencies

- **Standard Libraries**: `iostream` for basic input-output operations, `fstream` for file operations, `iomanip` for data formatting, and `arpa/inet.h` for IP address manipulation.
- **Network Libraries**: `netinet/ip.h` for IP header definitions, `netinet/tcp.h` for TCP header structures, and `netinet/udp.h` for UDP headers. These libraries provide the structures and functions needed to parse and analyze network packet headers accurately.