Network Packet Sniffer Application (Netsnif)

Project Report

Team Details

Group No. – 14

Team Members:

1. Name – Aastik Bansal
   Enrollment No. – 23114002
   Mobile No. – 7889775996
   Email ID. – aastik_b@cs.iitr.ac.in
2. Name – Harshit Kumar Meena
   Enrollment No. – 23114037
   Mobile No. – 7240385472
   Email ID. – harshit_km@cs.iitr.ac.in
3. Name – Pradyumn Kejriwal
   Enrollment No. – 23114081
   Mobile No. – 7486981629
   Email ID. – pradyumn_k@cs.iitr.ac.in
4. Name – Ridham Dave
   Enrollment No. – 23114087
   Mobile No. – 9313494392
   Email ID. – ridham_d@cs.iitr.ac.in
5. Name – Suprajeet Suman
   Enrollment No. – 23114097
   Mobile No. – 7855087787
   Email ID. – suprajeet_s@cs.iitr.ac.in
6. Name – Utkarsh Raj
   Enrollment No. – 23114102
   Mobile No. – 7683886642
   Email ID. – utkarsh_r@cs.iitr.ac.in

Contributions

1. Harshit and Utkarsh

   Implemented the PacketSniffer engine for real-time packet capture using Npcap. Developed filtering logic for protocols and IP and performed thorough testing to ensure stability and correctness.

2. Aastik and Pradyumn

   Built CSV and PCAP export features and implemented live protocol distribution and traffic graphs using Qt Charts. Developed the hex dump viewer to visualize raw packet data in both hex and ASCII formats.

3. Suprajeet and Ridham

   Designed the application interface using Qt Designer and styled it with custom stylesheets. Integrated all UI components (device selection, filters, charts, hex view) with backend logic using Qt signals and slots.

Table of Contents

Introduction

Background

Computer Networks

Computer networks enable the interconnection of computing devices for the purpose of sharing resources and exchanging data. These networks may range from small **Local Area Networks (LANs)** within a single building to expansive **Wide Area Networks (WANs)** like the Internet. In a network, data is transmitted in the form of small units called **packets**, and the way these packets are created, addressed, routed, and received is governed by a set of rules known as **network protocols**.

1.1 Network Models and Layered Architecture

In order to standardize how devices communicate over networks, models like the **OSI model** and the **TCP/IP model** were developed. These layered architectures break down the complex process of data communication into manageable, hierarchical functions, allowing interoperability across different hardware and software platforms.

The OSI Model (Open Systems Interconnection)

The OSI model is a **seven-layer conceptual framework** introduced by the International Organization for Standardization (ISO) that defines how data is transmitted and processed over a network. Each layer has a distinct role:

1. **Physical Layer**
   Transmits raw bits over physical media (e.g., Ethernet cables, radio frequencies).
   - Relevance to our project: This is where actual packet signals travel—though not directly accessed by our application.
2. **Data Link Layer**
   Manages node-to-node communication and MAC addressing. Responsible for framing and error detection at the hardware level.
   - Our packet sniffer can capture Ethernet headers and MAC addresses, which exist here.
3. **Network Layer**
   Handles logical addressing and routing (IP addresses).
   - You'll see IP headers, source and destination IPs in the packets our tool analyzes.
4. **Transport Layer**
   Ensures reliable data transfer (e.g., TCP) or fast, connectionless transfer (e.g., UDP).
   - Our application may extract port numbers, sequence numbers, and flags here.
5. **Session Layer**
   Manages sessions or connections between systems.
   - Less directly relevant—many session functions are folded into the Transport or Application Layer in real-world implementations.
6. **Presentation Layer**
   Deals with data encoding, compression, and encryption.
   - Our application might detect encrypted traffic but doesn't decode it unless additional processing is done.

7. **Application Layer**
   Interfaces directly with software applications (e.g., HTTP, FTP, DNS).
   - Packet inspection here can reveal application-specific headers or protocol use.

The TCP/IP Model

The **TCP/IP model** (also known as the Internet Protocol Suite) is a more practical and widely implemented model compared to the OSI. It has four layers:

1. **Link Layer**
   Combines OSI's Physical and Data Link layers.
   - In our project, Npcap provides access to this layer, especially for Ethernet frames.
2. **Internet Layer**
   Equivalent to the OSI's Network Layer. Handles IP addressing and routing.
   - IP headers are captured and decoded in our application.
3. **Transport Layer**
   Mirrors the OSI Transport Layer. Responsible for TCP and UDP communication.
   - This is where the tool detects port numbers and protocol flags.
4. **Application Layer**
   Includes the functionality of OSI's Session, Presentation, and Application layers.
   - High-level traffic like HTTP GET requests or DNS lookups may be interpreted here if payloads are parsed.

Comparing the Two Models

OSI Model	TCP/IP Model	Relevance to Our Project
Physical	Link	Raw packet data transmission
Data Link	Link	MAC addresses, frame headers
Network	Internet	IP addresses, routing
Transport	Transport	TCP/UDP protocols, ports
Session	Application	Logical connections
Presentation	Application	Data formatting, encoding (e.g., SSL)
Application	Application	High-level protocols (e.g., HTTP)

1.2 Packet Structure and Traffic

Each packet that traverses a network contains:

- **Header** – Information such as source/destination IP, port numbers, and protocol type.
- **Payload** – The actual data being transmitted.
- **Trailer** – Error-checking information (e.g., CRC).

Monitoring this traffic allows network administrators to inspect anomalies, detect intrusions, monitor performance, or troubleshoot connectivity issues. This is the fundamental purpose of **packet sniffers**—tools that capture and analyze network traffic in real-time.

1.3 Packet Sniffing and Network Monitoring

Packet sniffing is the process of intercepting and logging traffic that passes over a digital network. A packet sniffer works by setting a **network interface card (NIC)** into **promiscuous mode**, allowing it to capture all packets regardless of destination.

Packet sniffers are essential tools in:

- **Network diagnostics** – Detecting latency, dropped packets, and connectivity issues.
- **Cybersecurity** – Identifying unauthorized access, malware communications, or data leaks.
- **Traffic analysis** – Understanding bandwidth usage or application-specific traffic trends.

Packet sniffers can be either:

- **Passive** – Only observe and log traffic (e.g., Wireshark),
- **Active** – Inject packets or interfere with traffic (used in testing or attacks).

The functionality of a sniffer depends heavily on low-level access to packet data, which is not provided by regular OS APIs. This is where libraries like **Npcap** come into play.

Motivation

The development of Netsnif was motivated by several key factors:

1. Need for a modern, user-friendly packet analysis tool
2. Requirement for real-time network traffic monitoring
3. Integration of advanced visualization features
4. Need for detailed packet inspection capabilities
5. Support for various export formats

Application Domain

The application finds its utility in various domains:

- Network Security Analysis
- Network Performance Monitoring
- Protocol Analysis and Debugging
- Network Troubleshooting
- Educational Purposes
- Security Research and Forensics

Proposed System

System Architecture

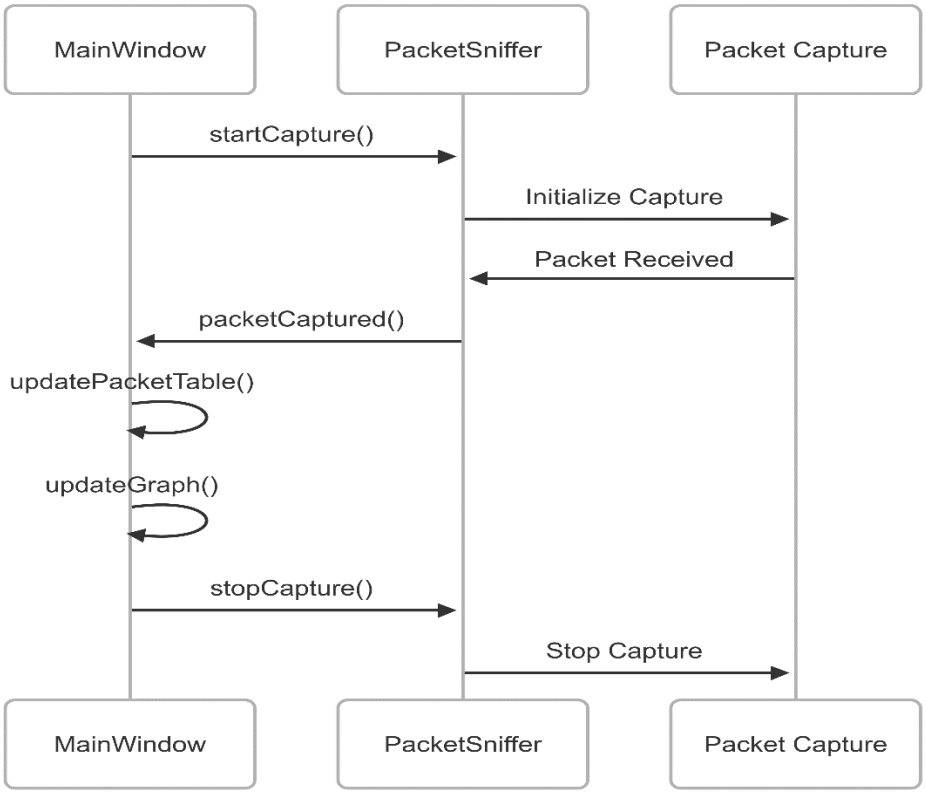
The Netsnif application is built on a modular architecture that consists of three main components:

1. **MainWindow (User Interface)**
   - Qt-based graphical user interface
   - Real-time packet visualization
   - Protocol filtering system
   - Statistical analysis charts
   - Hex dump viewer
   - Export functionality
2. **PacketSniffer (Core Engine)**
   - Packet capture thread
   - Network interface management
   - Packet processing and analysis
   - Protocol identification
   - Data storage management
3. **Libpcap/Npcap Integration**
   - Low-level packet capture
   - Network interface access
   - Raw packet handling
   - Platform-independent capture

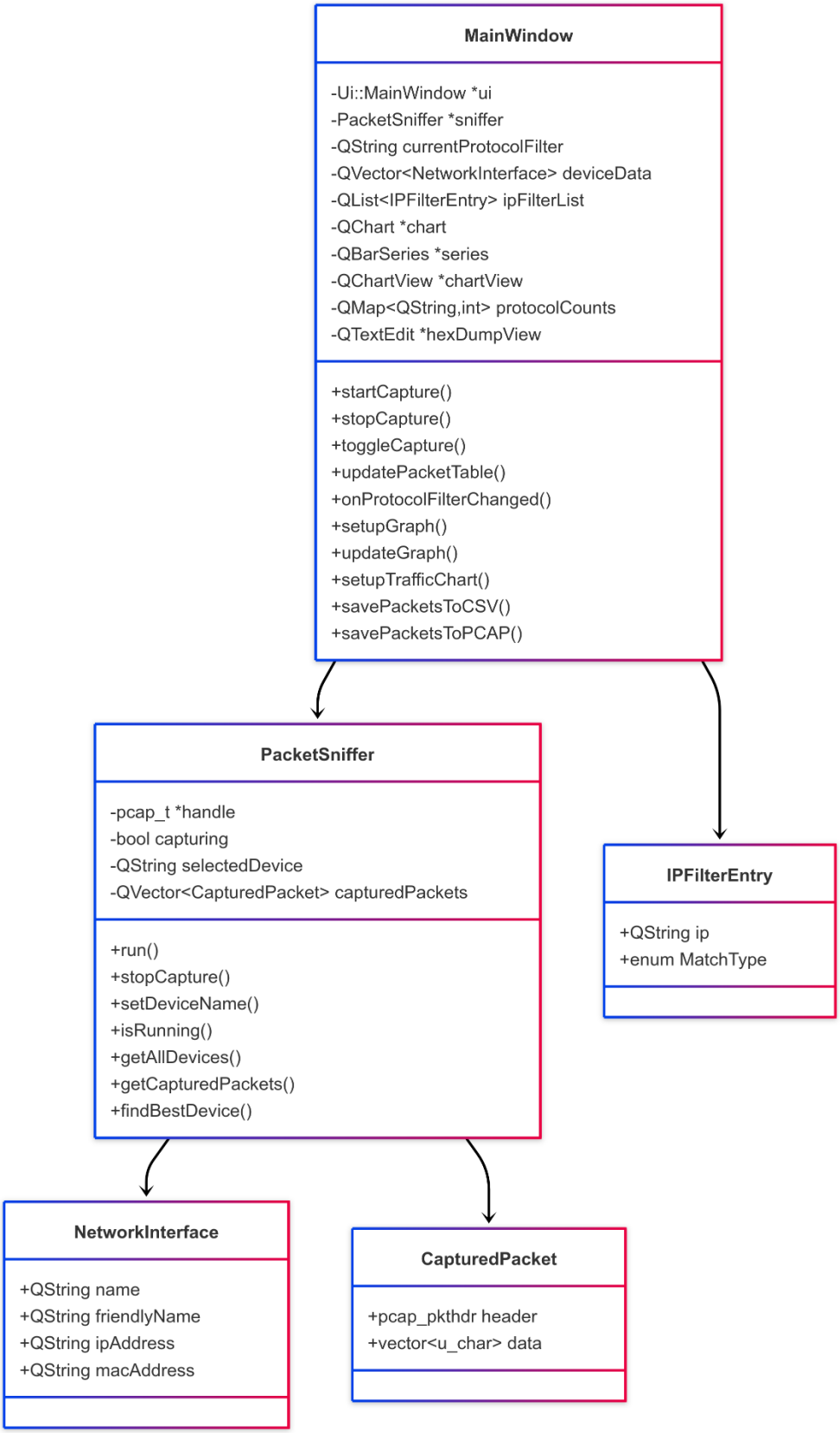
System Components

1. **MainWindow Class**
   - UI management and event handling
   - Packet display and filtering
   - Statistical visualization
   - Export functionality
   - Hex dump viewing
   - Network interface selection
2. **PacketSniffer Class**
   - Packet capture thread
   - Network interface management
   - Packet processing
   - Protocol analysis
   - Data storage
3. **Data Structures**
   - NetworkInterface: Interface information
   - CapturedPacket: Packet data and metadata
   - IPFilterEntry: Filter configuration

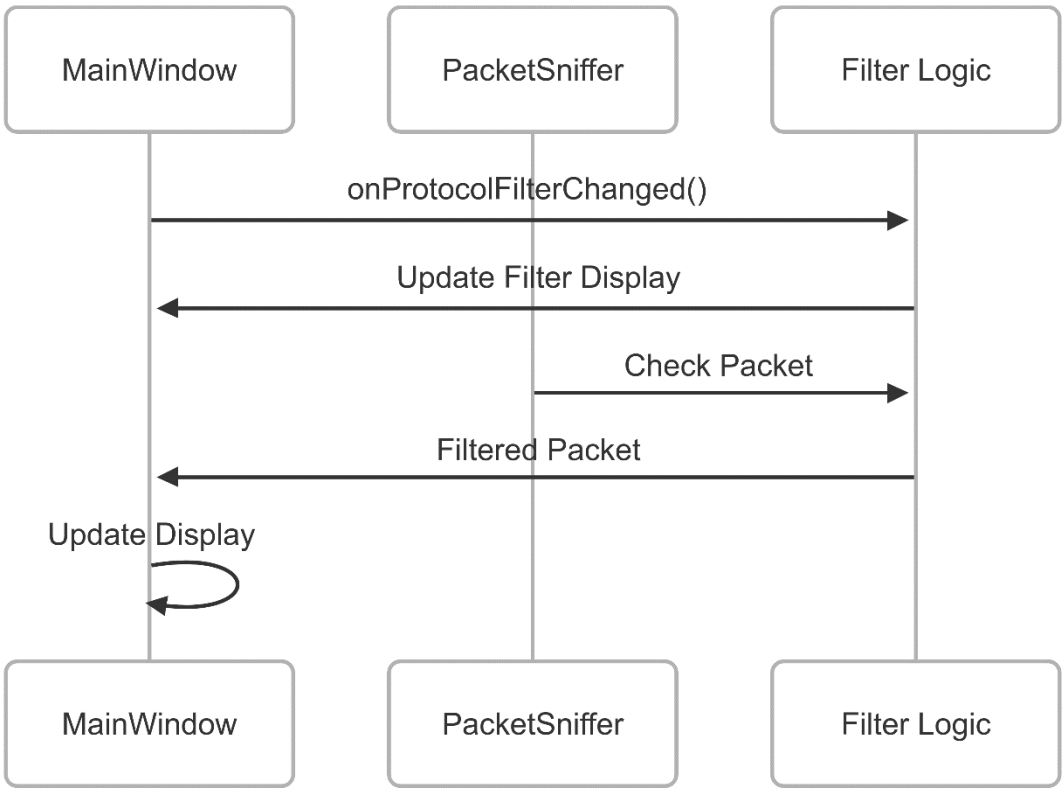
Sequence Diagram for Packet Capture



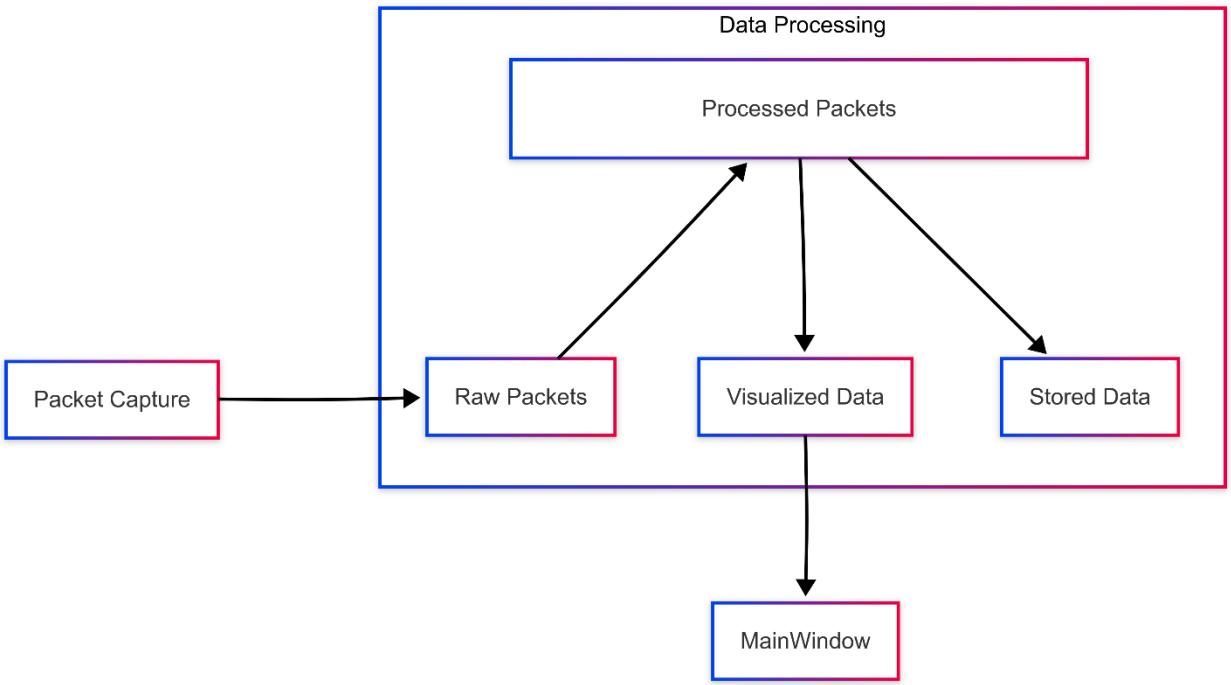
Class Diagram



Sequence Diagram for Filter Application



Data Flow Diagram



Novel Features

1. **Real-time Analysis**
   - Live packet capture and display
   - Protocol-based filtering
   - IP-based filtering
   - Real-time statistics
2. **Visualization**
   - Protocol distribution charts
   - Traffic volume graphs
   - Hex dump viewer
   - Packet details display
3. **Data Management**
   - CSV export
   - PCAP export
   - Capture constraints
   - Automatic device selection

Algorithms

1. **Packet Capture Algorithm**

```
function CapturePackets():
    Initialize network interface
    while capturing_enabled:
        packet = ReadFromInterface()
        if packet_meets_filter_criteria:
            ProcessPacket(packet)
            UpdateStatistics(packet)
            DisplayPacket(packet)
        end if
    end while
end function
```

2. **Filter Application Algorithm**

```
function ApplyFilters(packet):
    if protocol_filter_enabled:
        if packet.protocol != selected_protocol:
            return false
        end if
    end if
    if ip_filter_enabled:
        if not matches_ip_filter(packet.source_ip, packet.dest_ip):
            return false
        end if
    end if

    return true
end function
```

3. Hex Dump Generation Algorithm

```
function MainWindow::createHexDump(packet, length):
    hexDump = ""
    asciiDump = ""

    for i = 0 to length:
        // Format hex byte
        hexByte = formatHexByte(packet[i])
        hexDump += hexByte + " "

        // Format ASCII character
        asciiChar = formatAsciiChar(packet[i])
        asciiDump += asciiChar

        // New line every 16 bytes
        if (i + 1) % 16 == 0:
            hexDump += "  " + asciiDump + "\n"
            asciiDump = ""
        end if
    end for

    // Pad last line if needed
    if length % 16 != 0:
        remaining = 16 - (length % 16)
        hexDump += string(remaining * 3, ' ')
        hexDump += "  " + asciiDump + "\n"
    end if
    return hexDump
end function
```

4. Network Interface Selection Algorithm

```
function PacketSniffer::findBestDevice():
    devices = getAllDevices()
    if devices.empty():
        return ""
    end if
    for device in devices:
        run all devices for 2 seconds
        count number of packets
    end for
    if maximum count > 0
         choose device with maximum packet captured
         return bsetdevice.name
    end if
// Try to find a device with an IP address
    for device in devices:
        if device.ipAddress != "":
            return device.name
        end if
    end for
end function
```

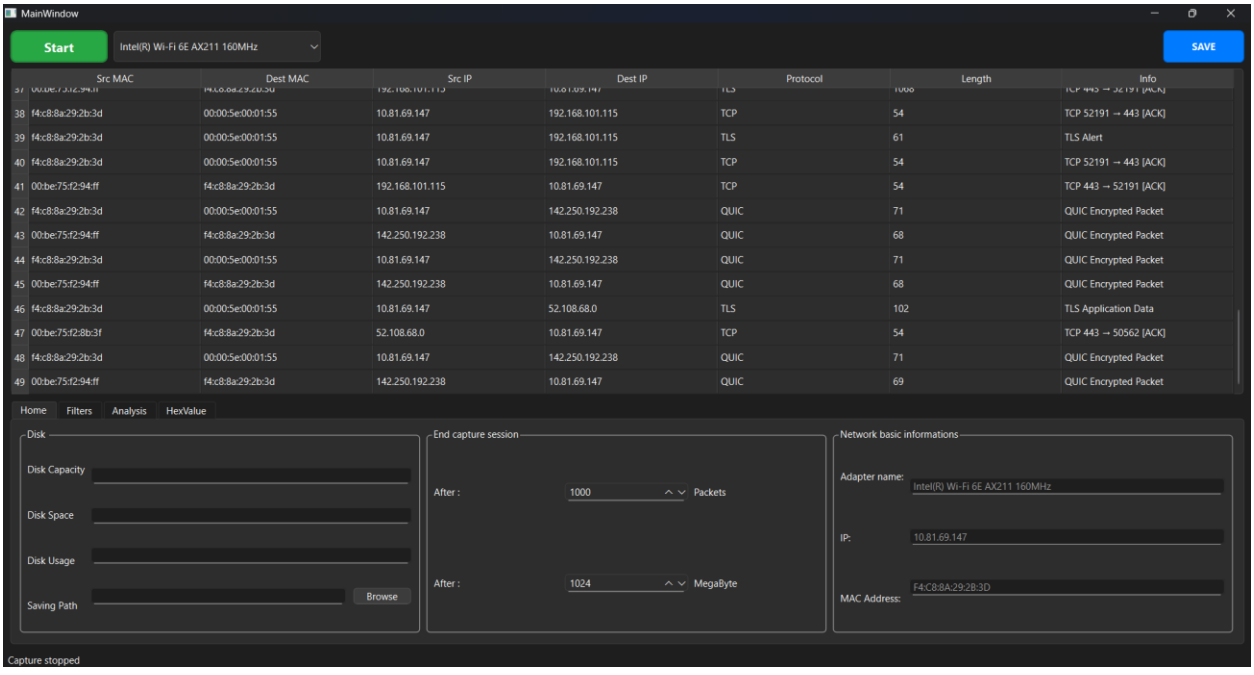
Implementation

Tools and Technologies Used

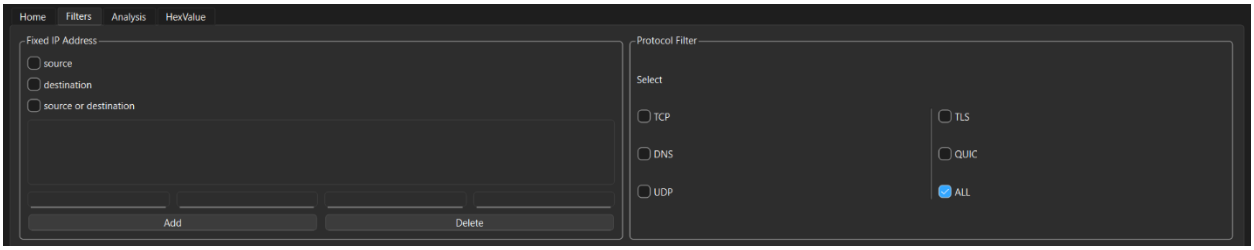
1. **Development Framework**
   - Qt 6 Framework
   - C++ 17
   - Libpcap/Npcap
   - Qt Charts
2. **Build System**
   - CMake
   - MinGW-w64 compiler
   - Qt Creator IDE
3. **Libraries**
   - Qt6Core
   - Qt6Gui
   - Qt6Widgets
   - Qt6Network
   - Qt6Charts

System Screenshots

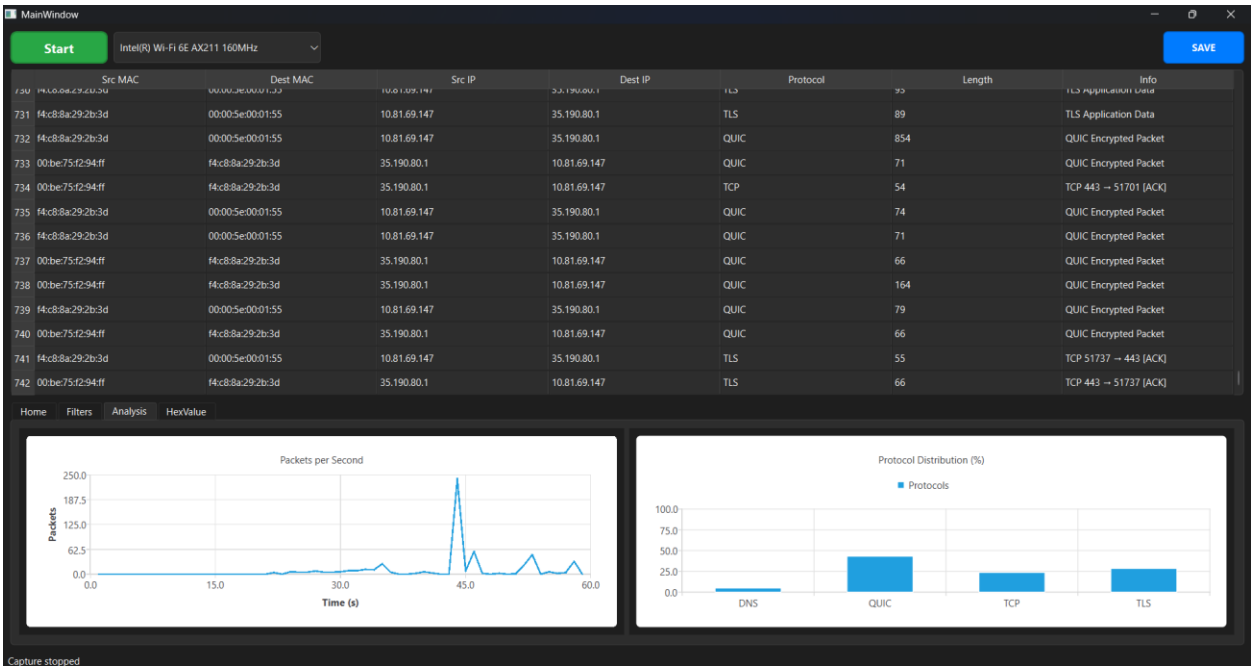
1. Main application interface



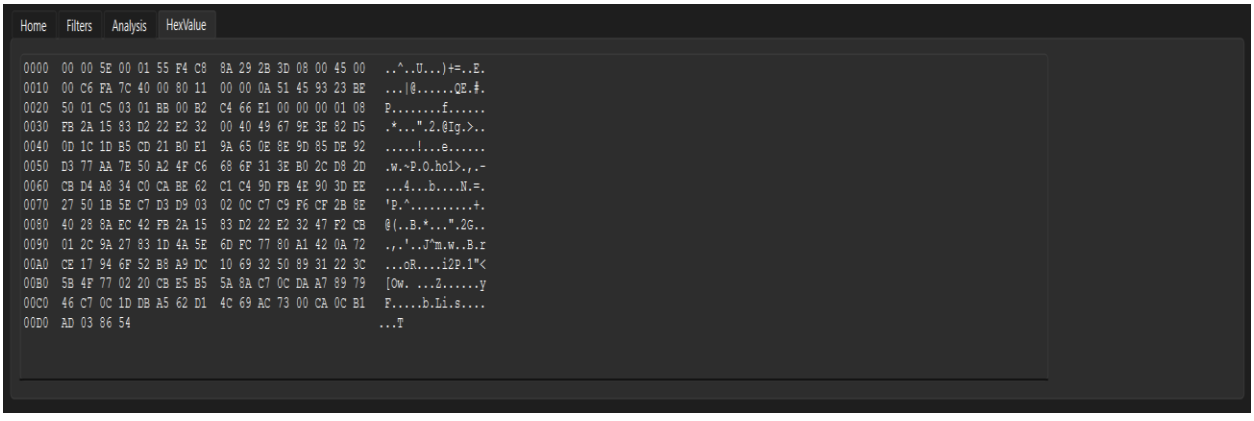
2. Filter configuration



3. Statistical charts



4. Hex dump view



Comparative Analysis

Feature	Netsnif	Wireshark	TCPDump	NetworkMiner
GUI Interface	✓	✓	✗	✓
Real-time Capture	✓	✓	✓	✓
Protocol Analysis	✓	✓	Limited	✓
Statistical Analysis	✓	✓	✗	Limited
Custom Filters	✓	✓	Limited	Limited
Cross-platform	✓	✓	✓	✗
Performance Impact	Low	Medium	Low	Medium
Export Capabilities	✓	✓	Limited	✓
Hex Dump View	✓	✓	✗	Limited

Conclusion

Netsnif represents a modern approach to network packet analysis, offering a balanced combination of powerful features and user-friendly interface. The application successfully achieves its goals of providing:

1. Real-time packet capture and analysis
2. Intuitive user interface
3. Advanced filtering capabilities
4. Comprehensive protocol support
5. Statistical analysis and visualization
6. Flexible export options

The implementation demonstrates strong adherence to software engineering principles, including:

- Modular design
- Separation of concerns
- Efficient resource management
- Cross-platform compatibility
- Extensible architecture

Future enhancements could include:

- Advanced protocol decoding
- Custom protocol support
- Enhanced export formats
- Network topology visualization
- Plugin system for extensibility

Bibliography

1. Stevens, W. R., Fenner, B., & Rudoff, A. M. (2004). UNIX Network Programming: The Sockets Networking API (Vol. 1). Addison-Wesley Professional.
2. Kozierok, C. M. (2005). The TCP/IP Guide: A Comprehensive, Illustrated Internet Protocols Reference. No Starch Press.
3. Qt Documentation. (2023). Qt Network Programming. Retrieved from https://doc.qt.io/qt-6/qtnetwork-index.html
4. Tanenbaum, A. S., & Wetherall, D. J. (2011). Computer Networks (5th ed.). Pearson.
5. Sanders, C. (2017). Practical Packet Analysis: Using Wireshark to Solve Real-World Network Problems (3rd ed.). No Starch Press.
6. Hunt, C. (2002). TCP/IP Network Administration (3rd ed.). O'Reilly Media.
7. Qt Company. (2023). Qt Charts Documentation. Retrieved from https://doc.qt.io/qt-6/qtcharts-index.html
8. Fall, K. R., & Stevens, W. R. (2011). TCP/IP Illustrated, Volume 1: The Protocols (2nd ed.). Addison-Wesley Professional.