Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет телекоммуникаций и информатики»

(СибГУТИ)

Кафедра прикладной математики и кибернетики

Курсовая работа

по дисциплине «Операционные системы»

Тема «Разработка анализатора трафика (сниффера)»

Выполнил:

Студент гр. ИП-213

Терновский Д.Р.

Работу проверил:

Ст. преподаватель кафедры ПМиК

Нужнов А.В.

Новосибирск 2024 г.

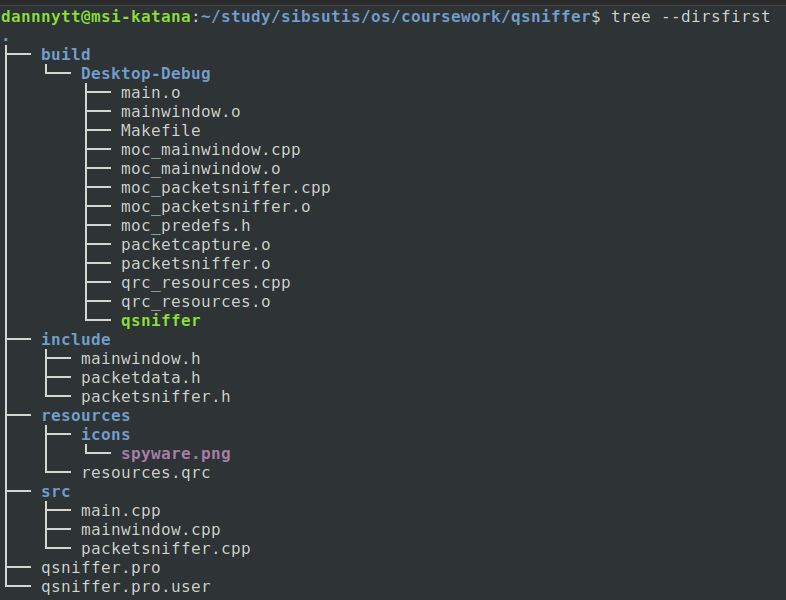
**Введение**

Цель данной работы — разработать анализатор сетевого трафика (сниффер), который позволит перехватывать, обрабатывать и визуализировать сетевые пакеты.

Для реализации проекта используется C++ как основной язык программирования. Графический интерфейс реализован на базе Qt, что обеспечивает кроссплатформенность приложения и интуитивно понятный пользовательский опыт. Захват сетевых пакетов осуществляется с помощью библиотеки libpcap, которая предоставляет доступ к данным на уровне сетевого интерфейса и позволяет управлять процессом перехвата пакетов.

В рамках работы были реализованы такие функции как, захват и фильтрация пакетов, отображение информации в удобном формате.

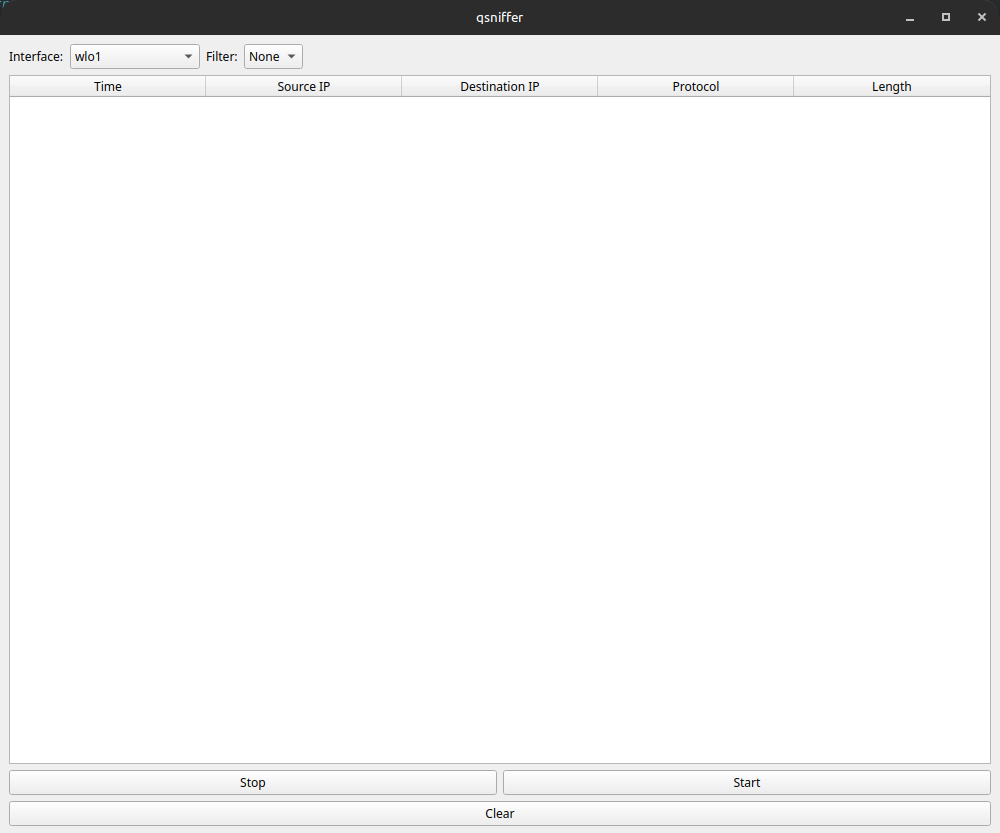
**Описание программы**

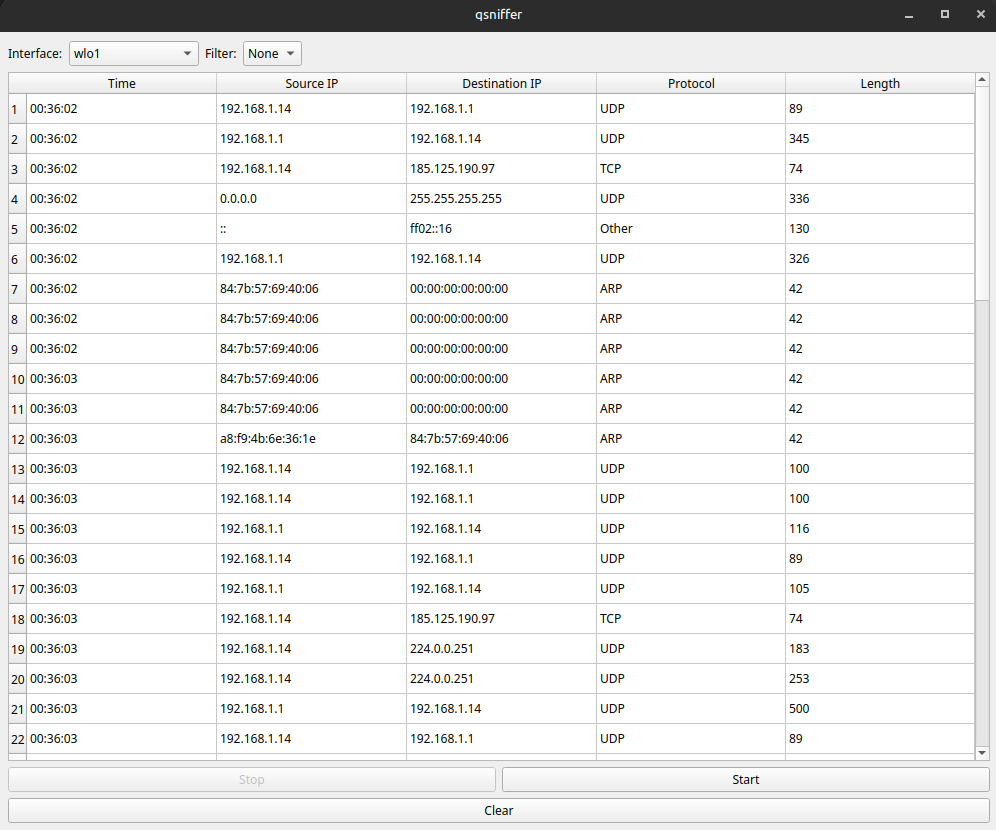
Рис.1 — Структура проекта.

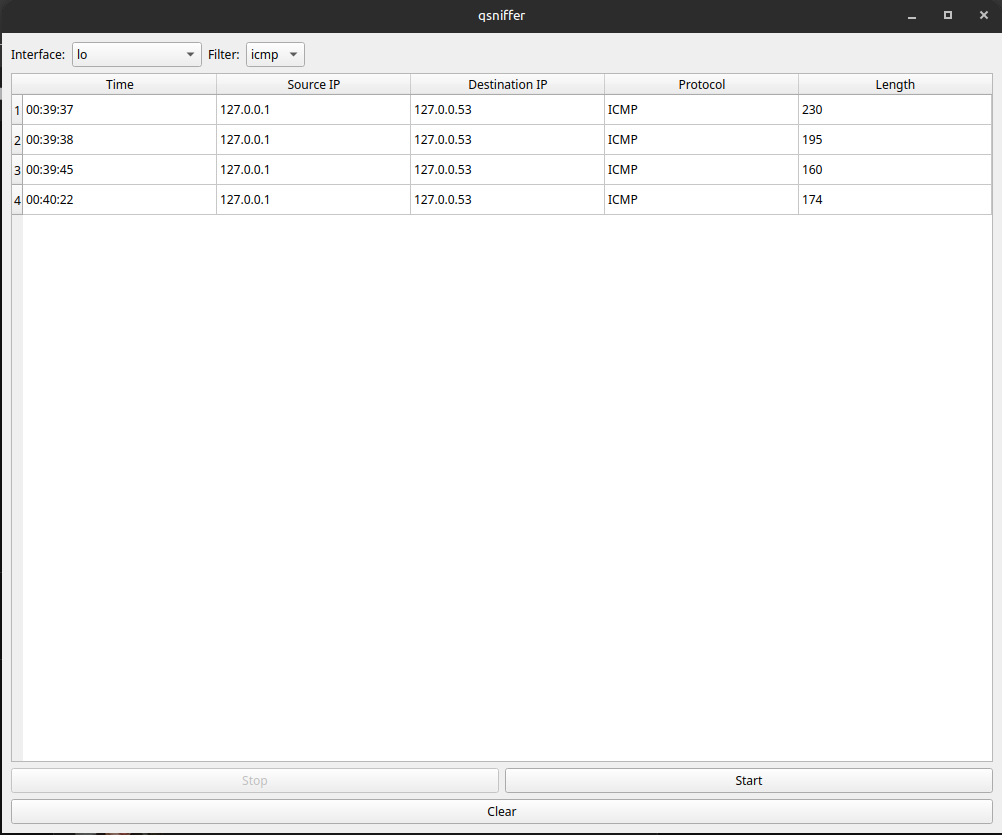
Для улучшения читаемости и «чистоты» проекта, он был разделен на следующие модули:

* build/ - директория, которая содержит скомпилированные файлы, объектные файлы и исполняемый файл.
* include/ - директория, хранящая заголовочные файлы, которые определяют классы, их интерфейсы и структуры.
  + - mainwindow.h — графический интерфейс главного окна.
    - packetdata.h — структура для хранения данных пакета.
    - packetsniffer.h - основной функционал для захвата сетевых пакетов.
* resources/ - директория, которая хранит ресурсы для программы, в данном случае в ней хранится иконка программы.
* src/ - директория, содержащая исходные файлы с реализациями классов.
  + - main.cpp — точка входа в программу.
    - mainwindow.cpp — реализация главного окна.
    - packetsniffer.h — реализация логики захвата и обработки пакетов.
* resources/

**Демонстрация работы программы**

Рис.2 — Главное окно программы.

Рис.3 — Захват пакетов (интрефейс wlo1, фильтр отсутсвует).

Рис.4 — Захват пакет с фильтрацией (интерфейс lo, фильтр icmp).

**Заключение**

Целью данной работы было разработать анализатор сетевого трафика (сниффер), который позволит эффективно перехватывать, фильтровать и визуализировать сетевые пакеты. Для реализации проекта использовался язык программирования **C++**, который обеспечил производительность и гибкость в разработке, а также **Qt** для создания графического интерфейса, что сделало приложение кросс-платформенным и интуитивно понятным для пользователя.

Захват сетевых пакетов был реализован с помощью библиотеки **libpcap**, предоставляющей доступ на уровне сетевого интерфейса и управление процессом захвата пакетов. В ходе работы были разработаны основные функции сниффера, такие как захват пакетов, их фильтрация и удобное отображение информации.

**Листинг программы**

**main.cpp**

#include "mainwindow.h"

#include <QMetaType>

#include "packetdata.h"

#include <QApplication>

#include <QIcon>

int main(int argc, char \*argv[])

{

QApplication app(argc, argv);

qRegisterMetaType<PacketData>("PacketData");

app.setWindowIcon(QIcon(":/icons/spyware.png"));

MainWindow w;

w.show();

return app.exec();

}

**mainwindow.h**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include "packetsniffer.h"

#include "packetdata.h"

#include <QMainWindow>

#include <QWidget>

#include <QTableWidget>

#include <QPushButton>

#include <QComboBox>

#include <QVBoxLayout>

#include <QHBoxLayout>

#include <QGridLayout>

#include <QStringList>

#include <QString>

#include <QThread>

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

MainWindow(QWidget \*parent = nullptr);

~MainWindow();

private:

QStringList getAllNetIfaces();

private slots:

void startSniffing();

void stopSniffing();

void clearTable();

void addPacketToTable(const PacketData &data);

private:

QWidget \*centralWidget;

QVBoxLayout \*mainLayout;

QTableWidget \*snifferTable;

QGridLayout \*buttonsLayout;

QPushButton \*startButton;

QPushButton \*stopButton;

QPushButton \*clearButton;

QHBoxLayout \*hLayout;

QComboBox \*netIfaces;

QComboBox \*protocolFilter;

PacketSniffer \*sniffer;

QThread \*snifferThread;

};

#endif // MAINWINDOW\_H

**mainwindow.cpp**

#include "mainwindow.h"

#include <QLabel>

#include <QHeaderView>

#include <QListView>

#include <QMessageBox>

#include <pcap.h>

MainWindow::MainWindow(QWidget \*parent)

: QMainWindow(parent),

sniffer(nullptr),

snifferThread(nullptr)

{

/\*Параметры окна\*/

this->resize(1000, 800);

this->setMinimumSize(800, 600);

/\*Настройка центрального виджета\*/

centralWidget = new QWidget(this);

this->setCentralWidget(centralWidget);

mainLayout = new QVBoxLayout(centralWidget);

/\*Создание и настройка combo box для выбора интерфейса\*/

QLabel \*ifaceLabel = new QLabel("Interface:", this);

hLayout = new QHBoxLayout;

hLayout->addWidget(ifaceLabel);

netIfaces = new QComboBox(this);

netIfaces->addItems(getAllNetIfaces());

hLayout->addWidget(netIfaces);

/\*Создание и настройка combo box для выбора фильтра\*/

QLabel \*filterLabel = new QLabel("Filter:", this);

hLayout->addWidget(filterLabel);

protocolFilter = new QComboBox(this);

protocolFilter->addItems({

"None",

"tcp",

"udp",

"icmp",

"arp"

});

hLayout->addWidget(protocolFilter);

mainLayout->addLayout(hLayout);

hLayout->addStretch();

/\*Настройка таблицы для отображения пакетов\*/

snifferTable = new QTableWidget(this);

snifferTable->setColumnCount(5);

snifferTable->setHorizontalHeaderLabels({

"Time",

"Source IP",

"Destination IP",

"Protocol",

"Length"

});

snifferTable->horizontalHeader()->setSectionResizeMode(QHeaderView::Stretch);

snifferTable->setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Expanding);

mainLayout->addWidget(snifferTable);

/\*Настройка кнопок\*/

buttonsLayout = new QGridLayout;

startButton = new QPushButton("Start", this);

stopButton = new QPushButton("Stop", this);

clearButton = new QPushButton("Clear", this);

buttonsLayout->addWidget(stopButton, 0, 0);

buttonsLayout->addWidget(startButton, 0, 1);

buttonsLayout->addWidget(clearButton, 1, 0, 1, 2);

mainLayout->addLayout(buttonsLayout);

connect(startButton, &QPushButton::clicked, this, &MainWindow::startSniffing);

connect(stopButton, &QPushButton::clicked, this, &MainWindow::stopSniffing);

connect(clearButton, &QPushButton::clicked, this, &MainWindow::clearTable);

}

MainWindow::~MainWindow() {}

QStringList MainWindow::getAllNetIfaces() {

QStringList ifaceList;

char errbuf[PCAP\_ERRBUF\_SIZE];

pcap\_if\_t \*interfaces;

if (pcap\_findalldevs(&interfaces, errbuf) == -1) {

QMessageBox::warning(this, "Warning", QString("Error getting devices: %1").arg(errbuf));

return ifaceList;

}

for (pcap\_if\_t \*dev = interfaces; dev; dev = dev->next) {

ifaceList.append(dev->name);

}

pcap\_freealldevs(interfaces);

return ifaceList;

}

void MainWindow::startSniffing() {

startButton->setEnabled(false);

stopButton->setEnabled(true);

clearButton->setEnabled(false);

sniffer = new PacketSniffer;

sniffer->setInterface(netIfaces->currentText());

sniffer->setFilter(protocolFilter->currentText() == "None" ? "" : protocolFilter->currentText());

snifferThread = new QThread(this);

snifferThread->start();

sniffer->moveToThread(snifferThread);

connect(snifferThread, &QThread::started, sniffer, &PacketSniffer::startCapture);

connect(sniffer, &PacketSniffer::packetCaptured, this, &MainWindow::addPacketToTable);

connect(sniffer, &PacketSniffer::destroyed, snifferThread, &QThread::quit);

connect(snifferThread, &QThread::finished, snifferThread, &QThread::deleteLater);

}

void MainWindow::stopSniffing() {

stopButton->setEnabled(false);

startButton->setEnabled(true);

clearButton->setEnabled(true);

if (sniffer) {

sniffer->stopCapture();

}

if (snifferThread) {

snifferThread->quit();

snifferThread->wait();

delete snifferThread;

snifferThread = nullptr;

}

delete sniffer;

sniffer = nullptr;

}

void MainWindow::clearTable() {

snifferTable->setRowCount(0);

clearButton->setEnabled(false);

}

void MainWindow::addPacketToTable(const PacketData &packet) {

int row = snifferTable->rowCount();

snifferTable->insertRow(row);

snifferTable->setItem(row, 0, new QTableWidgetItem(packet.time));

snifferTable->setItem(row, 1, new QTableWidgetItem(packet.source));

snifferTable->setItem(row, 2, new QTableWidgetItem(packet.dest));

snifferTable->setItem(row, 3, new QTableWidgetItem(packet.protocol));

snifferTable->setItem(row, 4, new QTableWidgetItem(packet.length));

clearButton->setEnabled(true);

}

**packetdata.h**

#ifndef PACKETDATA\_H

#define PACKETDATA\_H

#include <QString>

struct PacketData {

QString time;

QString source;

QString dest;

QString protocol;

QString length;

};

#endif // PACKETDATA\_H

**packetsniffer.h**

#ifndef PACKETSNIFFER\_H

#define PACKETSNIFFER\_H

#include "packetdata.h"

#include <pcap.h>

#include <QObject>

#include <QString>

#include <QDateTime>

#include <QMessageBox>

class PacketSniffer : public QObject {

Q\_OBJECT

public:

explicit PacketSniffer(QObject \*parent = nullptr);

~PacketSniffer();

void setInterface(const QString &iface);

void setFilter(const QString &filter);

public slots:

void startCapture();

void stopCapture();

void onShowWarningMessage(const QString &message);

signals:

void packetCaptured(const PacketData &packet);

void showWarningMessage(const QString &message);

private:

bool applyFilter();

static void processPacket(u\_char \*userData, const struct pcap\_pkthdr \*header, const u\_char \*packet);

void processIPv4(const u\_char \*packet, PacketData &packetData);

void processIPv6(const u\_char \*packet, PacketData &packetData);

void processARP(const u\_char \*packet, PacketData &packetData);

QString macToString(const unsigned char \*mac);

QString protocolToString(int protocol);

private:

bool capturing;

QString ifaceName;

QString filter;

pcap\_t \*pcapHandle;

};

#endif // PACKETCAPTURE\_H

**packetsniffer.cpp**

#include "packetsniffer.h"

#include <netinet/ether.h>

#include <netinet/ip.h>

#include <netinet/ip6.h>

#include <netinet/udp.h>

#include <netinet/tcp.h>

#include <netinet/ip\_icmp.h>

PacketSniffer::PacketSniffer(QObject \*parent)

: QObject(parent), capturing(false), pcapHandle(nullptr)

{

connect(this, &PacketSniffer::showWarningMessage, this, &PacketSniffer::onShowWarningMessage);

}

PacketSniffer::~PacketSniffer() {}

void PacketSniffer::setInterface(const QString &iface) {

ifaceName = iface;

}

void PacketSniffer::setFilter(const QString &filter) {

this->filter = filter;

}

void PacketSniffer::onShowWarningMessage(const QString &message) {

QMessageBox::warning(nullptr, "Warning", message);

}

void PacketSniffer::startCapture() {

char errbuf[PCAP\_ERRBUF\_SIZE];

pcapHandle = pcap\_open\_live(ifaceName.toUtf8().constData(), BUFSIZ, 1, 1000, errbuf);

if (!pcapHandle) {

emit showWarningMessage(QString("Failed to open device: %1").arg(errbuf));

return;

}

if (!filter.isEmpty() && !applyFilter()) {

pcap\_close(pcapHandle);

return;

}

capturing = true;

pcap\_loop(pcapHandle, 0, processPacket, reinterpret\_cast<u\_char \*>(this));

pcap\_close(pcapHandle);

pcapHandle = nullptr;

}

bool PacketSniffer::applyFilter() {

struct bpf\_program bpfProgram;

if (pcap\_compile(pcapHandle, &bpfProgram, filter.toUtf8().constData(), 0, PCAP\_NETMASK\_UNKNOWN) == -1) {

emit showWarningMessage(QString("Failed to compile filter: %1").arg(pcap\_geterr(pcapHandle)));

return false;

}

if (pcap\_setfilter(pcapHandle, &bpfProgram) == -1) {

emit showWarningMessage(QString("Failed to compile filter: %1").arg(pcap\_geterr(pcapHandle)));

pcap\_freecode(&bpfProgram);

return false;

}

pcap\_freecode(&bpfProgram);

return true;

}

void PacketSniffer::stopCapture() {

capturing = false;

if (pcapHandle) {

pcap\_breakloop(pcapHandle);

}

}

void PacketSniffer::processPacket(u\_char \*userData, const struct pcap\_pkthdr \*header, const u\_char \*packet) {

auto \*sniffer = reinterpret\_cast<PacketSniffer \*>(userData);

PacketData packetData;

packetData.time = QDateTime::currentDateTime().toString("hh:mm:ss");

const auto \*ethHeader = reinterpret\_cast<const struct ether\_header \*>(packet);

uint16\_t ethertype = ntohs(ethHeader->ether\_type);

if (ethertype == ETHERTYPE\_IP) {

sniffer->processIPv4(packet, packetData);

} else if (ethertype == ETHERTYPE\_IPV6) {

sniffer->processIPv6(packet, packetData);

} else if (ethertype == ETHERTYPE\_ARP) {

sniffer->processARP(packet, packetData);

}

packetData.length = QString::number(header->len);

emit sniffer->packetCaptured(packetData);

}

void PacketSniffer::processIPv4(const u\_char \*packet, PacketData &packetData) {

const auto \*ipHeader = reinterpret\_cast<const struct ip \*>(packet + sizeof(struct ether\_header));

char srcIp[INET\_ADDRSTRLEN], dstIp[INET\_ADDRSTRLEN];

inet\_ntop(AF\_INET, &ipHeader->ip\_src, srcIp, INET\_ADDRSTRLEN);

inet\_ntop(AF\_INET, &ipHeader->ip\_dst, dstIp, INET\_ADDRSTRLEN);

packetData.source = QString(srcIp);

packetData.dest = QString(dstIp);

packetData.protocol = protocolToString(ipHeader->ip\_p);

}

void PacketSniffer::processIPv6(const u\_char \*packet, PacketData &packetData) {

const auto \*ipv6Header = reinterpret\_cast<const struct ip6\_hdr \*>(packet + sizeof(struct ether\_header));

char srcIp[INET6\_ADDRSTRLEN], dstIp[INET6\_ADDRSTRLEN];

inet\_ntop(AF\_INET6, &ipv6Header->ip6\_src, srcIp, INET6\_ADDRSTRLEN);

inet\_ntop(AF\_INET6, &ipv6Header->ip6\_dst, dstIp, INET6\_ADDRSTRLEN);

packetData.source = QString(srcIp);

packetData.dest = QString(dstIp);

packetData.protocol = protocolToString(ipv6Header->ip6\_nxt);

}

void PacketSniffer::processARP(const u\_char \*packet, PacketData &packetData) {

const auto \*arpHeader = reinterpret\_cast<const struct ether\_arp \*>(packet + sizeof(struct ether\_header));

packetData.source = macToString(arpHeader->arp\_sha);

packetData.dest = macToString(arpHeader->arp\_tha);

packetData.protocol = "ARP";

}

QString PacketSniffer::macToString(const unsigned char \*mac) {

return QString("%1:%2:%3:%4:%5:%6").arg(mac[0], 2, 16, QChar('0'))

.arg(mac[1], 2, 16, QChar('0'))

.arg(mac[2], 2, 16, QChar('0'))

.arg(mac[3], 2, 16, QChar('0'))

.arg(mac[4], 2, 16, QChar('0'))

.arg(mac[5], 2, 16, QChar('0'));

}

QString PacketSniffer::protocolToString(int protocol) {

switch (protocol) {

case IPPROTO\_TCP: return "TCP";

case IPPROTO\_UDP: return "UDP";

case IPPROTO\_ICMP: return "ICMP";

case IPPROTO\_ICMPV6: return "ICMPv6";

default: return "Other";

}

}

**qsniffer.pro**

QT += core gui

greaterThan(QT\_MAJOR\_VERSION, 4): QT += widgets

CONFIG += c++17

LIBS += -lpcap

INCLUDEPATH += $$PWD/include

HEADERS += \

include/mainwindow.h \

include/packetdata.h \

include/packetsniffer.h

SOURCES += \

src/main.cpp \

src/mainwindow.cpp \

src/packetsniffer.cpp

# Default rules for deployment.

qnx: target.path = /tmp/$${TARGET}/bin

else: unix:!android: target.path = /opt/$${TARGET}/bin

!isEmpty(target.path): INSTALLS += target

RESOURCES += \

resources/resources.qrc