P2Pal - Project Documentation

Laya Fakher A20582305

P2Pal is a peer-to-peer chat application that uses UDP-based communication for messaging and peer discovery. It uses Gossip Protocol and Anti-Entropy mechanisms to make sure that messages are reliable across multiple peers.

Project File Structure and Explanation

main.cpp

This is the entry point of the application. It initializes the QApplication and launches the MainWindow GUI.

```
#include "mainwindow.h"

#include "networking.h"

#include <QApplication>

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

  QApplication app(argc, argv);

  MainWindow window;

  window.show();

  return app.exec();

}
```

QApplication app(argc, argv); : initializes the Qt application

MainWindow window; : creates the main chat window

window.show(); : displays the GUI

app.exec(); : starts the event loop for user interaction

mainwindow.h

This header file defines the MainWindow class, which is responsible for handling the user interface and user interactions.

```
#ifndef MAINWINDOW_H

#define MAINWINDOW_H
```

```
#include <QMainWindow>
#include <QTextEdit>
#include <QLineEdit>
#include <QPushButton>
#include <QUdpSocket>
#include <QTimer>
#include <QSet>
#include "networking.h"
#include "vectorclock.h"

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow {
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private slots:
    void sendMessage();
    void processPendingDatagrams();
    void discoverPeers();
```

```
  void runGossipProtocol();

  void addPeer();

  void runAntiEntropy();


private:

  QTimer *discoveryTimer;

  QTimer *gossipTimer;

  QTimer *antiEntropyTimer;

  Ui::MainWindow *ui;

  QTextEdit *chatLog;

  QLineEdit *inputField;

  QPushButton *addPeerButton;

  QUdpSocket *udpSocket;

  QSet<QHostAddress> peers;

  int sequenceNumber;

  QString localIdentifier;

  Networking *network;

  VectorClock vectorClock;

};

#endif // MAINWINDOW_H
```

QTextEdit chatLog : stores and displays messages

QLineEdit inputField : accepts user input

QPushButton addPeerButton : allows adding a peer manually

QUdpSocket udpSocket : handles UDP-based communication

QTimer discoveryTimer : periodically searches for peers

QTimer gossipTimer : guarantees message propagation using gossip

QTimer antiEntropyTimer : periodically runs anti-entropy to synchronize messages

Networking network : manages networking functions

VectorClock vectorClock : tracks received messages

mainwindow.cpp

This file implements the user interface logic and integrates networking.

Constructor (MainWindow::MainWindow)

```
MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent), ui(new Ui::MainWindow), sequenceNumber(1), network(new Networking(this)) {
    discoveryTimer = new QTimer(this);
    gossipTimer = new QTimer(this);
    antiEntropyTimer = new QTimer(this);
    ui->setupUi(this);
    localIdentifier = QHostInfo::localHostName();
```

Initializes the timers and GUI elements.

Sets the local identifier using the hostname.

Sending a Message (sendMessage)

```
void MainWindow::sendMessage() {
    QString message = inputField->text().trimmed();
    if (message.isEmpty()) return;

    QVariantMap messageMap;
    messageMap["Type"] = "CHAT";
    messageMap["ChatText"] = message;
    messageMap["Origin"] = localIdentifier;
    messageMap["SequenceNumber"] = network->getNextSequenceNumber();

    QByteArray datagram =
QJsonDocument(QJsonObject::fromVariantMap(messageMap)).toJson();
    network->sendDatagram(datagram, messageMap["SequenceNumber"].toInt());

    chatLog->append("Me: " + message);
    inputField->clear();
}
```

Converts the message into a JSON format.

Sends it using the Networking class.

Handling Peer Discovery

```
void MainWindow::discoverPeers() {
    network->broadcastDiscovery();
}
```

Calls the Networking class to find peers via UDP.

networking.h

This header file defines the Networking class, which handles all UDP communications.

```
#ifndef NETWORKING_H
#define NETWORKING_H
#include <QObject>
#include <QUdpSocket>
#include <QTextEdit>
#include <QSet>
```

```
#include <QHostAddress>

#include "vectorclock.h"


class Networking : public QObject {

  Q_OBJECT


public:

  explicit Networking(QObject *parent = nullptr);

  void sendDatagram(const QByteArray &datagram, int sequenceNumber);

  void processIncomingDatagrams(QTextEdit *chatLog);

  void broadcastDiscovery();

  void runGossip();

  void sendAcknowledgment(const QHostAddress &receiver, int sequenceNumber);

  void removeAcknowledgedMessage(int sequenceNumber);

  int getNextSequenceNumber();

  void runAntiEntropy();


private slots:

  void handleIncomingDatagrams();


private:

  QMap<QString, QSet<int>> peerMessages;

  QUdpSocket *udpSocket;

  QSet<QHostAddress> peers;

  VectorClock vectorClock;

  int sequenceNumber = 1;
  QMap<int, QByteArray> messageBuffer;
};

#endif // NETWORKING_H
```

QUdpSocket udpSocket : manages UDP packets.

QMap peerMessages : stores received message tracking.

QSet peers : stores connected peers.

QMap messageBuffer : buffers unsent messages.

networking.cpp

Handles the **network communication logic**.

Broadcasting Peer Discovery

```
void Networking::broadcastDiscovery() {
  QVariantMap discoveryMap;
  discoveryMap["Type"] = "DISCOVERY";
  QByteArray discoveryMessage =
QJsonDocument(QJsonObject::fromVariantMap(discoveryMap)).toJson();
  udpSocket->writeDatagram(discoveryMessage, QHostAddress::Broadcast, 45454);
}
```

Sends a UDP discovery packet to find other peers.

Receiving Messages

```
void Networking::processIncomingDatagrams(QTextEdit *chatLog) {
  while (udpSocket->hasPendingDatagrams()) {
    QByteArray datagram;
    datagram.resize(udpSocket->pendingDatagramSize());
    QHostAddress sender;
    quint16 senderPort;
    udpSocket->readDatagram(datagram.data(), datagram.size(), &sender, &senderPort);

    QJsonDocument doc = QJsonDocument::fromJson(datagram);
    QVariantMap messageMap = doc.object().toVariantMap();
    QString type = messageMap["Type"].toString();

    if (type == "CHAT") {
      QString origin = messageMap["Origin"].toString();
      int seqNum = messageMap["SequenceNumber"].toInt();
      if (!peerMessages[origin].contains(seqNum)) {
        peerMessages[origin].insert(seqNum);
        chatLog->append(origin + ": " + messageMap["ChatText"].toString());
      }
      sendAcknowledgment(sender, seqNum);
    }
  }
}
```

Handles incoming messages and prevents duplicates.

vectorclock.h and vectorclock.cpp

Implements message tracking using vector clocks.

Tracking Received Messages

```
void VectorClock::updateClock(const QString &origin, int sequenceNumber) {
  if (!clock.contains(origin) || clock[origin] < sequenceNumber) {
    clock[origin] = sequenceNumber;
    }
}
```

Updates the last received message from each peer.

Checking for Missing Messages

```
bool VectorClock::isNewMessage(const QString &origin, int sequenceNumber) {
    return !clock.contains(origin) || clock[origin] < sequenceNumber;
}
```

Helps detect missing or duplicate messages.

Build Instructions

Build the Project in Windows (Using MinGW)

```
mkdir build

cd build

cmake -G "MinGW Makefiles" ..

mingw32-make
```

Run the Application in Windows

```
.\P2Pal.exe
```

Basic Test Cases

Test Case 1: GUI Loads Successfully

Steps:

1. Run P2Pal.
2. Make sure the chat window opens without errors.
3. Verify that the input field and chat log are visible.

Expected Result:
The application window appears correctly, and the UI elements are functional.

Test Case 2: Peer Discovery

Steps:

1. Run two instances of P2Pal on the same machine.
2. Check the log messages for Peer discovery request received and Discovery response received.
3. Verify that peers are added automatically.

Expected Result:
Both instances detect each other and can communicate

Test Case 3: Sending and Receiving Messages

Steps:

1. Start two instances of P2Pal.
2. In one instanc type a message and press Enter.
3. Observe the message appearing in both chat logs.

Expected Result:
The message is received by both peers

Test Case 4: Message Propagation (Gossip Protocol)

Steps:

1. Run three instances of P2Pal.
2. Send a message from Instance 1.
3. Close Instance 1.
4. Make sure Instance 2 retransmits the message to Instance 3.

Expected Result:
The message reaches all active peers, even if the sender disconnects.

Test Case 5: Anti-Entropy Synchronization

Steps:

1. Start three instances.
2. Block the network connection of Instance 2.
3. Send messages from Instance 1 and Instance 3.
4. Re-enable Instance 2's network.
5. Check if missing messages are received via Anti-Entropy.

Expected Result:
Instance 2 should receive previously missed messages after resynchronization.

Automation Scripts

Windows Batch Script (run_multiple_instances.bat)

```
@echo off

start P2Pal.exe

start P2Pal.exe

start P2Pal.exe
```

Double-click run_multiple_instances.bat to run three instances.

Windows PowerShell Script (test_p2pal.ps1)

```
Write-Host "Running P2Pal Tests..."

Start-Process -FilePath .\P2Pal.exe

Start-Process -FilePath .\P2Pal.exe

Start-Sleep -Seconds 5

Write-Host "Sending Test Message..."

$udpClient = New-Object System.Net.Sockets.UdpClient

$udpClient.Connect("127.0.0.1", 45454)

$bytes = [System.Text.Encoding]::UTF8.GetBytes("Test Message")
```

```
$udpClient.Send($bytes, $bytes.Length)

Start-Sleep -Seconds 5

if (Get-Content P2Pal.log | Select-String "Test Message") {

  Write-Host "Test Passed"

} else {

  Write-Host "Test Failed"

}
```

Run it with:

```
.\test_p2pal.ps1
```