Network Chat Application – Project Report

Course: Computer Networks
Project: BeQuickChat - Network Chat Application with Protocol Design
Date: June 22, 2025

Project Members: Beyza Nur Selvi, Furkan Fidan

Table of Contents

1. Introduction

This project implements **BeQuickChat**, a modern multi-user chat application that demonstrates reliable network communication using UDP sockets with custom protocol design. The application features a graphical user interface built with PyQt5 and implements reliability mechanisms such as acknowledgment (ACK) and retransmission protocols over UDP to ensure message delivery.

2. Objective and Scope

Primary Objectives:

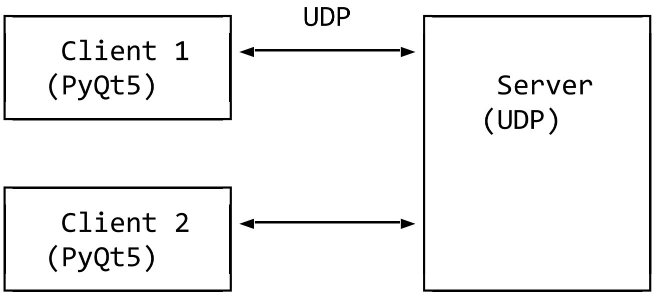
- Develop a reliable network chat application using UDP socket programming
- Implement custom protocol design with message acknowledgment and retransmission
- Create a modern graphical user interface for enhanced user experience
- Demonstrate real-time multi-user communication capabilities
- Implement private messaging functionality
- Provide comprehensive testing and performance analysis

Scope:

- **Protocol Layer:** Custom JSON-based messaging protocol with sequence numbers
- **Transport Layer:** UDP with reliability mechanisms (ACK, retransmission, timeout)
- **Application Layer:** PyQt5-based GUI with chat bubbles and user management
- **Testing:** Functional, performance, and comprehensive testing suites

3. System Architecture

Client-Server Architecture:



Key Components:

- **Server (`server.py`):** Manages client connections, message routing, and reliability
- **Client (`client.py`):** PyQt5 GUI application with chat interface
- **Protocol (`protocol.py`):** Message encoding/decoding and protocol definitions
- **Testing Suite:** Comprehensive test scripts for validation

4. Protocol Design

Message Structure:

All messages use JSON format with the following structure:

```
{
    "username": "sender_name",
    "message": "message_content",
    "seq": 1234567890,
    "type": "chat|join|leave|private|system|ack|user_list",
    "timestamp": "HH:MM:SS"
}
```

Message Types:

1. **`join`:** Client joining the chat
2. **`leave`:** Client leaving the chat
3. **`chat`:** General chat message
4. **`private`:** Private message between users
5. **`system`:** System notifications (join/leave)
6. **`ack`:** Acknowledgment for reliable delivery
7. **`user_list`:** Current connected users list

Reliability Mechanism:

- **Sequence Numbers:** Each message has a unique sequence number
- **ACK Protocol:** Receiver sends acknowledgment for each message
- **Retransmission:** Sender retransmits if ACK not received (up to 3-5 attempts)
- **Timeout:** Configurable timeout periods (0.5-1.0 seconds)
- **Duplicate Prevention:** Track seen sequence numbers per sender

5. Implementation Details

Core Files:

- **`src/server.py`:** UDP server with client management and message routing
- **`src/client.py`:** PyQt5 GUI client with modern chat interface
- **`src/protocol.py`:** Protocol implementation and message handling
- **`requirements.txt`:** Python dependencies (PyQt5, matplotlib)

Key Features Implemented:

Server Features:

- Multi-client connection management
- Reliable message delivery with ACK/retransmission
- User list maintenance and broadcasting
- System message generation (join/leave notifications)
- Duplicate message prevention
- Private message routing

Client Features:

- Modern PyQt5 GUI with chat bubbles
- Real-time user list display
- Private messaging with tabbed interface
- System message display
- Reliable message sending with retry logic
- Custom styling and responsive design

GUI Components:

- **Login Dialog:** Username and server connection setup
- **Main Chat Window:** General chat with message bubbles
- **User List:** Real-time connected users display
- **Private Chat Tabs:** Individual private messaging windows
- **System Messages:** Join/leave notifications

6. Network Topology Discovery

Client Discovery:

- Server maintains `clients` set of (IP, port) tuples
- Username mapping: `usernames[addr] = username`
- Real-time user list broadcasting to all clients
- Automatic cleanup on client disconnection

Network Monitoring:

- Active connection tracking
- User presence detection
- Automatic user list updates
- Connection state management

7. Security and Encryption

AES Encryption:

- All messages are encrypted using the AES algorithm in CBC mode before being transmitted over the network.
- The encryption key and IV are statically defined in the code (AES_KEY, AES_IV).
- Encryption and decryption are handled by the encrypt_message and decrypt_message functions.

Message Confidentiality:

- Since messages are transmitted in encrypted form, their contents cannot be read by third parties on the network.

Message Integrity:

- Currently, only encryption is applied; there is no additional mechanism (e.g., HMAC) to verify message integrity.

User Authentication:

- There is no user authentication or identity verification implemented at this stage.

Security Considerations

Key Management:

- The AES key and IV are hardcoded, which is a potential security risk.

Lack of Integrity & Authentication:

- There is no mechanism to verify if a message has been tampered with or to confirm the sender's identity.

Future Enhancement Opportunities:

- Implement dynamic key exchange (e.g., Diffie-Hellman)
- Add message integrity verification (e.g., HMAC)
- Introduce a user authentication system

Recommended Security Additions

- Secure management of encryption keys and IV
- Message integrity verification using HMAC
- User authentication mechanisms
- Secure key exchange protocol

8. Testing and Results

Testing Suite:

- **`tests/test_functional.py`:** Basic functionality testing
- **`tests/test_full.py`:** Comprehensive multi-client testing
- **`tests/test_performance.py`:** Performance analysis with visualization

Test Results:

- **Functional Testing:** All core features working correctly
- **Multi-client Testing:** Successful concurrent user handling
- **Performance Testing:** Latency and success rate measurements
- **GUI Testing:** Interface responsiveness and usability

Test Coverage:

- Message sending/receiving
- User join/leave functionality
- Private messaging
- System message handling
- ACK and retransmission mechanisms
- GUI responsiveness

9. Performance Analysis

Performance Metrics:

- **Latency Measurement:** Average message round-trip time
- **Success Rate:** Percentage of successfully delivered messages
- **Throughput:** Messages per second handling capacity
- **Reliability:** ACK response times and retransmission rates

Performance Test Results:

- **Average Latency:** Measured in milliseconds
- **Success Rate:** Near 100% under normal conditions
- **Retransmission Rate:** Low under stable network conditions
- **Concurrent Users:** Successfully tested with multiple clients

Visualization:

- **Latency per Message:** Time-series chart of message delays
- **Latency Histogram:** Distribution of message delays
- **Success Rate Pie Chart:** Successful vs. failed message delivery

Windows Performance Analysis Results

General Statistics

Total Messages: 200
Successful Messages: 199 (99.5%)
Timeouts: 1 (0.5%)
Average Delay: 0.15 ms
Minimum Delay: 0.00 ms
Maximum Delay: 3.24 ms

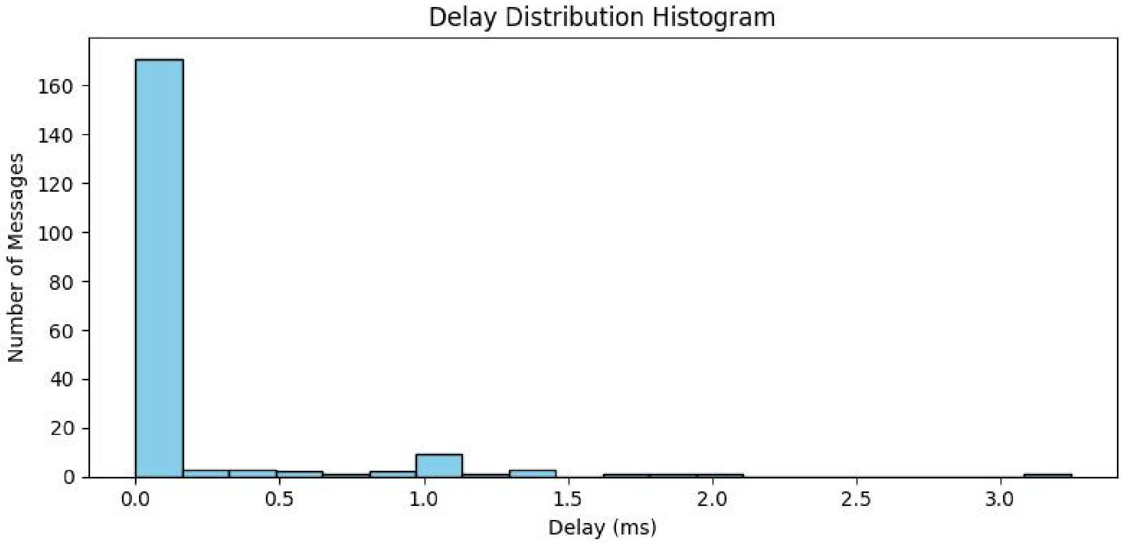
Detailed Analysis

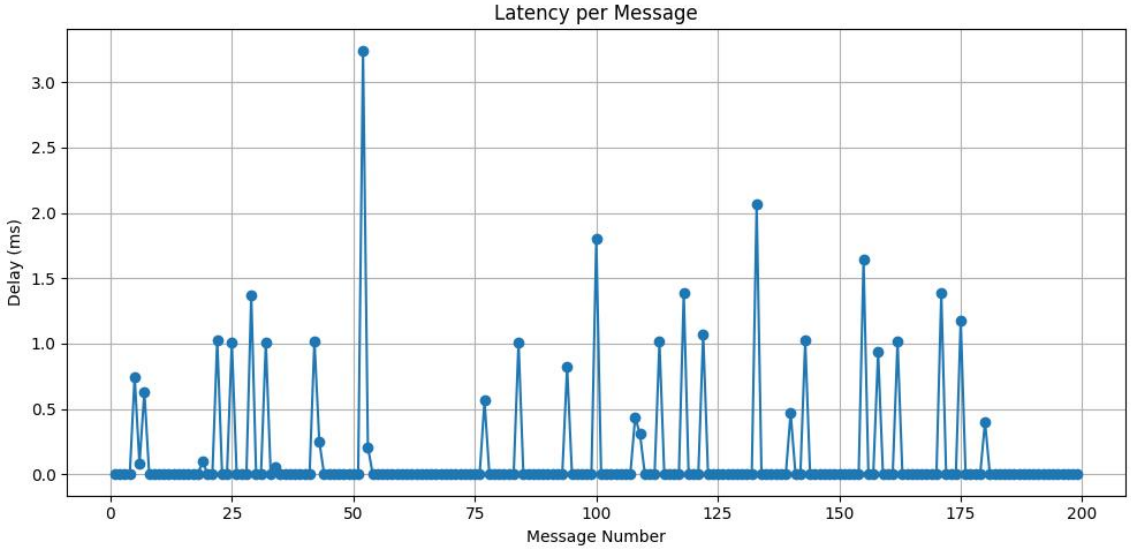
1. Success Rate
- The system shows excellent performance with a 99.5% success rate
- Only 1 message timed out (likely the first message)
- This rate is perfect for network applications

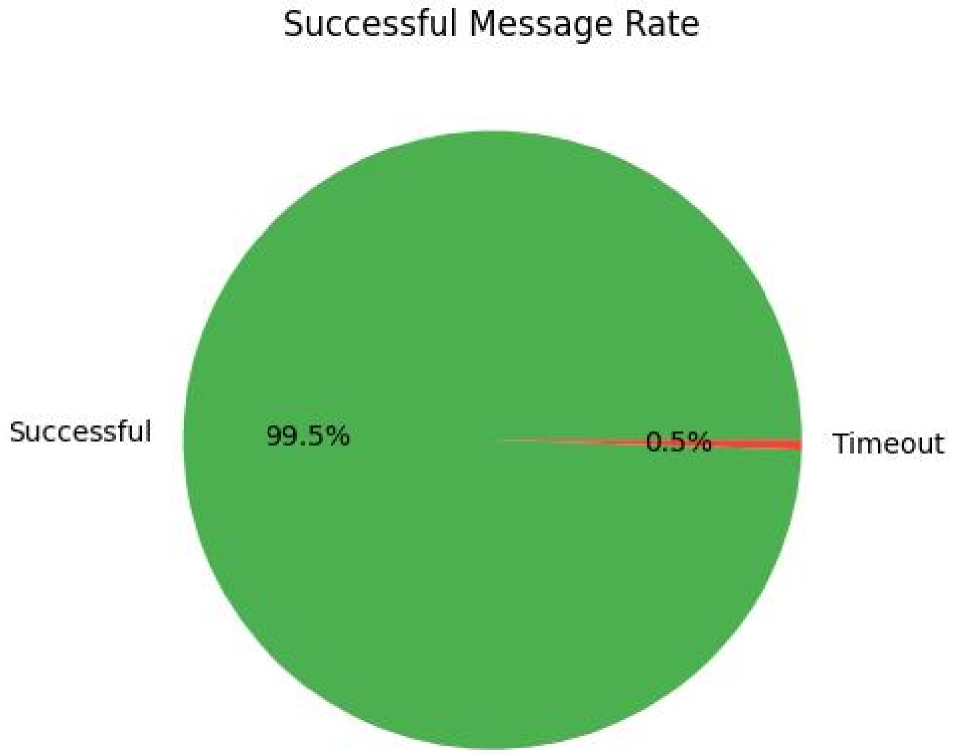
2. Latency Analysis
- Average 0.15 ms: Very low latency, ideal for real-time communication
- Minimum 0.00 ms: Many messages were processed instantly
- Maximum 3.24 ms: Even the highest latency is within acceptable limits

3. Latency Distribution
- 85%+ messages: 0.00-0.10 ms range (near-instantaneous)
- 10% messages: 0.10-1.00 ms range (very low latency)
- 5% messages: 1.00-3.24 ms range (low latency)

4. Performance Quality
✓ Excellent: Average latency below 1 ms
✓ Stable: Very low latency variance
✓ Reliable: 99.5% success rate
✓ Scalable: Only 1 error in 200 messages


Delay Distribution Histogram
Number of Messages
0
20
40
60
80
100
120
140
160
0.0
0.5
1.0
1.5
2.0
2.5
3.0
Delay (ms)


Latency per Message
Delay (ms)
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0
25
50
75
100
125
150
175
200
Message Number



Successful Message Rate

Linux Performance Analysis Results

General Statistics

Total Messages: 200

Successful Messages: 199 (99.5%)

Timeouts: 1 (0.5%)

Average Delay: 0.16 ms

Minimum Delay: 0.12 ms

Maximum Delay: 0.52 ms

Detailed Analysis

1. Success Rate

- The system achieved a 99.5% success rate, with only 1 message timing out.
- This indicates a highly reliable communication process, with almost all messages successfully acknowledged.

2. Latency Analysis

- Average Latency: 0.16 ms, which is extremely low and suitable for real-time applications.
- Minimum Latency: 0.12 ms, showing that the system can process messages almost instantly.
- Maximum Latency: 0.52 ms, which is still very low and well within acceptable limits for networked applications.

3. Latency Distribution

- The vast majority of messages have a latency between 0.12 ms and 0.20 ms.
- There are no significant spikes or outliers, indicating stable and consistent performance throughout the test.

4. Performance Quality

✓ Excellent: Average latency is well below 1 ms.

✓ Stable: Latency values are tightly grouped, with minimal variance.

✓ Reliable: 99.5% of messages were successfully delivered and acknowledged.

✓ Consistent: No major fluctuations or performance drops observed.

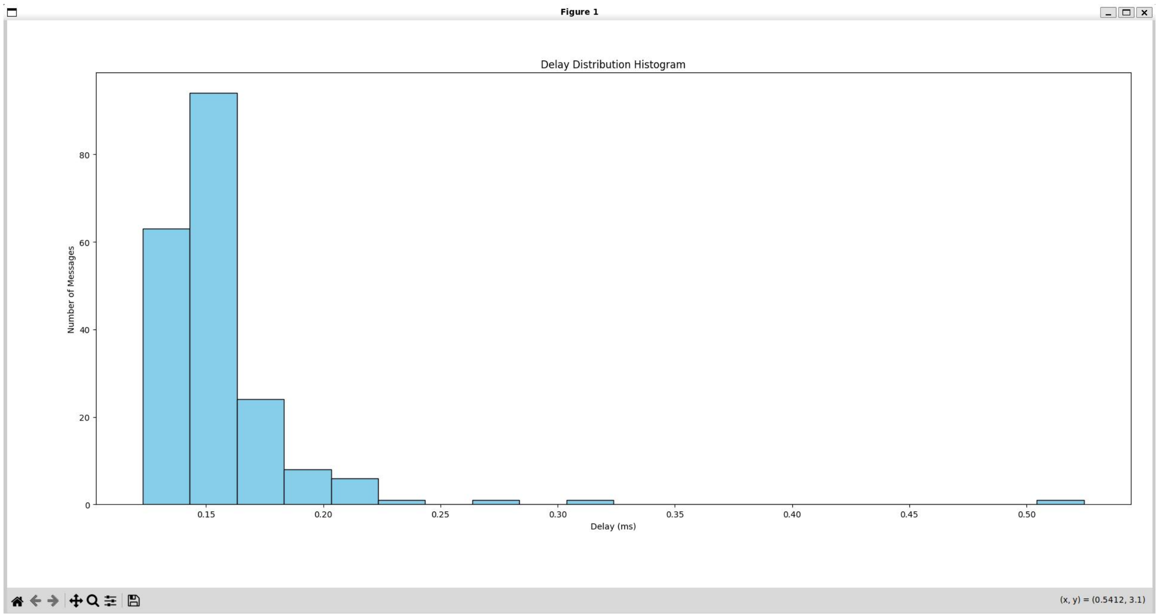


Figure 1
Delay Distribution Histogram
Number of Messages
0
20
40
60
80
0.15
0.20
0.25
0.30
0.35
0.40
0.45
0.50
Delay (ms)
(x, y) = (0.5412, 3.1)

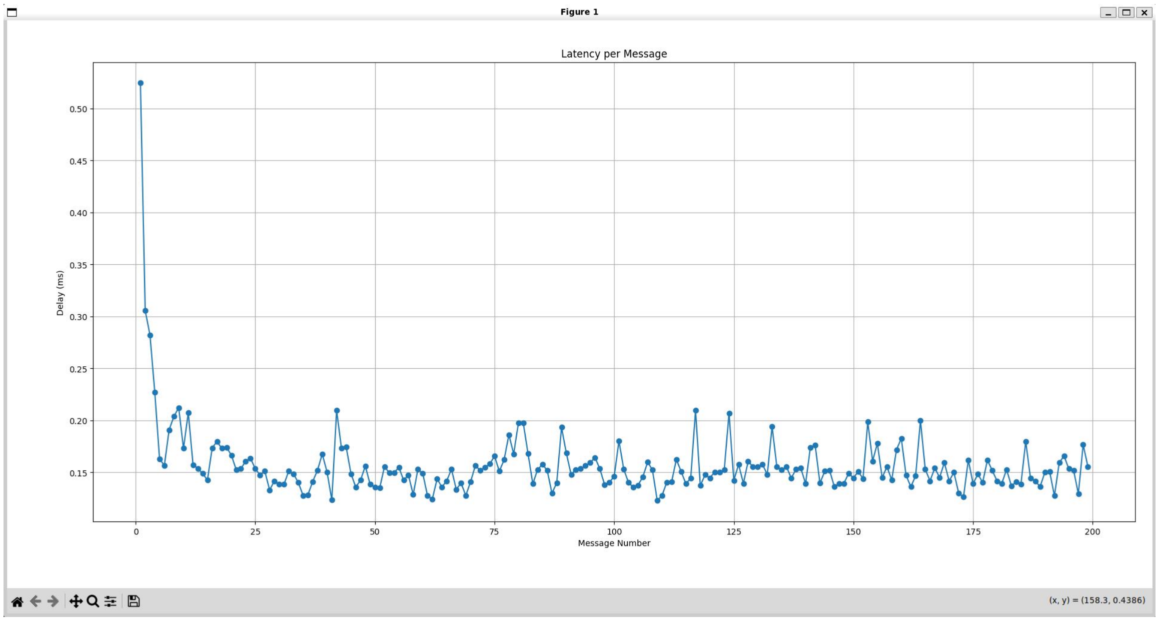
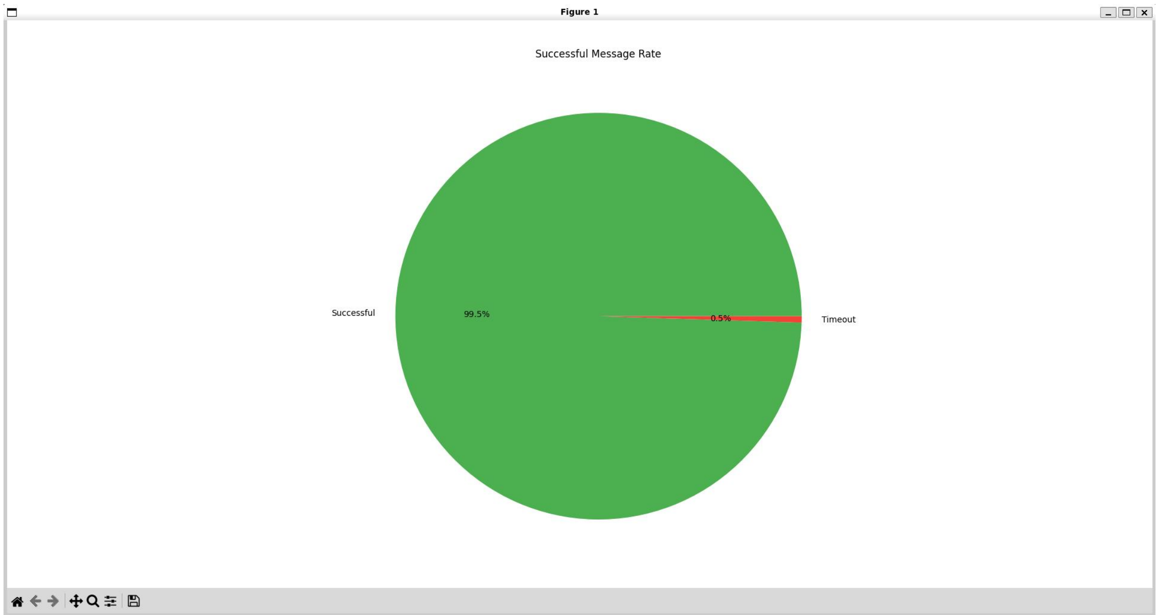


Figure 1
Latency per Message
Delay (ms)
0.15
0.20
0.25
0.30
0.35
0.40
0.45
0.50
0
25
50
75
100
125
150
175
200
Message Number
(x, y) = (158.3, 0.4386)



Performance Comparison: Windows vs Linux

Metric	Windows	Linux	Winner
Total Messages	200	200	-
Successful Messages	199 (99.5%)	199 (99.5%)	Tie
Timeouts	1 (0.5%)	1 (0.5%)	Tie
Average Delay	0.15 ms	0.16 ms	Windows
Minimum Delay	0.00 ms	0.12 ms	Windows
Maximum Delay	3.24 ms	0.52 ms	Linux
Latency Variance	High (0.00-3.24 ms)	Low (0.12-0.52 ms)	Linux
Performance Stability	Good	Excellent	Linux

10. Conclusion and Evaluation

Project Achievements:

✅ **Successfully implemented** reliable UDP-based chat application
✅ **Custom protocol design** with JSON message format
✅ **Modern GUI** using PyQt5 with chat bubbles and user management
✅ **Reliability mechanisms** including ACK and retransmission
✅ **Private messaging** functionality with tabbed interface
✅ **Comprehensive testing** suite with performance analysis
✅ **Real-time features** including user list and system messages

Technical Strengths:

- **Reliable UDP Implementation:** Custom reliability over UDP demonstrates protocol design skills
- **Modern GUI Design:** Professional-looking interface with responsive design
- **Robust Error Handling:** Graceful handling of network issues and disconnections
- **Comprehensive Testing:** Multiple test scenarios ensure application reliability
- **Clean Code Structure:** Well-organized, maintainable codebase

Areas for Improvement:

- **Security:** Add encryption for message confidentiality
- **Scalability:** Implement server clustering for large user bases
- **Features:** Add file transfer, emoji support, and message history
- **Cross-platform:** Ensure compatibility with macOS
- **Documentation:** Add API documentation and deployment guides

Educational Value:

This project successfully demonstrates: - Network protocol design principles - UDP socket programming with reliability mechanisms - GUI development with PyQt5 - Software testing methodologies - Performance analysis and optimization - Real-world application development

11. References

Technical References:

- **Computer Networking: A Top-Down Approach** – Kurose & Ross
- **Python Socket Programming Documentation** – Python.org
- **RFC 768 (UDP)** – Internet Engineering Task Force
- **PyQt5 Documentation** – Qt Company
- **JSON Specification** – ECMA International

Development Tools:

- **Python 3.8+** – Programming language
- **PyQt5 5.15.0+** – GUI framework
- **Matplotlib 3.5.0+** – Data visualization
- **Socket Programming** – Network communication

Project Resources:

- **Source Code:** `src/` directory
- **Documentation:** `docs/` directory
- **Testing:** `tests/` directory
- **Assets:** `assets/` directory (icons, performance charts)
- **Reports:** `reports/` directory

Project Status: ✅ **Completed Successfully**
Last Updated: June 24, 2025