SIMP: A Simple UDP-Based Chat Protocol Implementation

Ali Aliyev & Xander Smakman

2025

Table of Contents

[Introduction 2](#_Toc186405400)

[Project Structure 3](#_Toc186405401)

[External Dependencies 4](#_Toc186405402)

[Module Descriptions 5](#_Toc186405403)

[Module Dependencies and Interaction 7](#_Toc186405404)

[Summary and Limitations 8](#_Toc186405405)

[Execution Guide 9](#_Toc186405406)

# Introduction

This document describes the implementation of the Simple IMC Messaging Protocol (SIMP), a lightweight, UDP-based protocol designed for simple chat applications. The protocol provides basic mechanisms for establishing connections between 2 users, exchanging messages, and terminating chat sessions. It uses a three-way handshake and a stop-and-wait strategy for message delivery. This project implements the SIMP protocol using a client-server model, consisting of a daemon application and a client application. The purpose of this documentation is to provide an overview of the code and its architecture, as well as how to run the application.

# Project Structure

The project is divided into two main Python files:

## simp\_daemon.py

This file implements the server-side daemon that listens for incoming chat requests and relays messages between clients. It manages active chat sessions and ensures that message delivery works correctly.

## simp\_client.py

This file implements the client application, which is used by end-users to initiate chat sessions and send messages. It manages the client-side logic for establishing connections, sending chat messages, and handling retransmission.

# External Dependencies

This project uses the following Python standard library modules:

* **socket:** Used for creating and managing UDP socket connections.
* **threading:** Used to create and manage threads for handling concurrent network operations (e.g., receiving messages and handling requests).
* **time:** Used to implement timers and retransmission logic.

No other external libraries or dependencies are required for this project.

# Module Descriptions

This section describes the main modules (files) within the project:

* **simp\_daemon.py**
  + **SIMPDaemon Class:** The main class responsible for the daemon's functionality. It does the following:
    - Creates and binds a UDP socket to port 7777.
    - Maintains a dictionary of active chats, pairing usernames with their socket addresses and sequence numbers.
    - Handles incoming control and chat datagrams.
    - Implements the three-way handshake for establishing connections with clients.
    - Relays chat messages between connected clients.
    - Uses a dictionary to manage sequence numbers for each client to avoid duplicate messages.
  + **Helper functions:**
    - **create\_header(datagram\_type, operation, sequence, username, length):** Creates the header for a SIMP datagram as a byte string.
    - **parse\_header(header):** Parses the header part of a SIMP datagram and returns its components.
    - **Constants:** Constants for the port and message types
* **simp\_client.py**
  + **SIMPClient Class:** The main class for client functionality. It performs these main actions:
    - Creates and binds a UDP socket to the port specified by the user.
    - Provides functionality for starting a chat, sending messages, and quitting.
    - Implements the stop-and-wait protocol for reliable message sending, including retransmissions.
    - Receives and handles both control and chat messages
    - Uses a flag (waiting\_for\_ack) to keep track of outstanding chat messages.
    - **Constants:** Constants for the port and message types
  + **Helper Functions:**
    - **create\_header(datagram\_type, operation, sequence, username, length):** Creates the header for a SIMP datagram as a byte string.
    - **parse\_header(header):** Parses the header part of a SIMP datagram and returns its components.

# Module Dependencies and Interaction

The simp\_daemon.py and simp\_client.py files operate independently, but they communicate using the SIMP protocol:

1. **Connection Establishment:**
   * The client sends a SYN control datagram to the daemon.
   * The daemon responds with a SYN+ACK control datagram to the client
   * The client sends an ACK control datagram.
2. **Chat Message Exchange:**
   * The client sends a chat datagram to the daemon.
   * The daemon relays the message to the other participant.
   * The daemon sends an ACK control datagram to the original client
   * The message sent by the client is sent directly to its partner by the daemon.
3. **Connection Termination:**
   * Either the client sends a FIN control datagram to the daemon
   * The daemon sends an ACK control datagram to the chat partner.

All communication is done using UDP datagrams. The header provides metadata (type, operation, sequence, username, payload length).

# Summary and Limitations

This project provides a basic implementation of the SIMP protocol, and implements all of the core required functionality as specified in the assignment. The application demonstrates network concepts such as the three-way handshake, stop-and-wait, and socket programming.

**Limitations:**

* **Basic Error Handling:** The error handling is functional but not highly detailed.
* **No Message Loss Recovery:** While the stop-and-wait protocol handles retransmissions, it doesn't handle all possible error scenarios. For example, a delayed ACK may cause issues.
* **Simplified Chat:** The chat is synchronous and does not handle multiple clients interacting with a single client at the same time.
* **Basic Security:** There is no security or encryption implemented.
* **No message ordering:** If messages arrive out of order (which can happen in UDP) they will not be corrected.
* **Message limit:** There is no size limit for the message, which may lead to an issue if messages are extremely large.

# Execution Guide

To run the project:

1. **Start the Daemon:**

python simp\_daemon.py

content\_copydownload

Use code [with caution](https://support.google.com/legal/answer/13505487).Bash

1. **Start Clients:**

python simp\_client.py

content\_copydownload

Use code [with caution](https://support.google.com/legal/answer/13505487).Bash

The client will ask for the daemon's IP address and a client port.

1. **Start a Chat:** On one client, type "start". The client will ask you to enter the address of your partner and will then connect to that user.
2. **Send Messages:** On either client, type "send <your message>".
3. **Quit:** On either client type "q".