Reviewer: Debasish Saha Pranta(a1963099)
Reviewed for: Group 45

Date: 19 October 2025

1. General Feedback

I had the opportunity to test and review your SOCP chat system implementation.

The overall structure and modularity are impressive — it's clear that you have a solid grasp of how a secure, distributed communication system should be organized.

The architecture (master server + local server + clients) is well-defined and intuitive.

Using main.py as a single entry point for both server and client roles makes the project very clean.

Your logs and message prefixes ([peer], [user], [server]) were particularly helpful for tracing the runtime flow.

---

2. What I Liked

- Clear multi-server design with JOIN/WELCOME handshakes.
- Readable and well-commented Python code with modular files (server.py, client.py, sdb.py).
- Use of public/private key handling shows security awareness.
- Console feedback gives a good picture of system state and connections.

---

3. What Could Be Improved

- **Runtime stability:** Clients disconnect when executing commands such as /list or /tell; likely due to uncaught exceptions in message handlers.
- **User state cleanup:** Re-connecting with the same user ID after disconnection shows "user already exists." The server probably does not remove users on disconnect.
- **Incomplete message handling:** /tell and /list do not return expected responses across servers.

- **Error feedback:** More graceful responses (e.g., “unknown command” or “user not found”) would improve usability.

---

4. Security & Vulnerability Analysis

Even though the system focuses on secure communication, several potential vulnerabilities and risks were identified during static inspection and runtime tests:

1. **Unverified Peer Connections**
    - The JOIN/WELCOME process between servers does not appear to validate digital signatures or certificate authenticity.

      This leaves room for spoofed nodes to join the network.
2. **Plaintext Message Exchange in Local Testing**
    - Some message payloads (e.g., user list responses and direct messages) are transmitted without encryption in the current prototype.

      If deployed over public networks, these could be sniffed or modified by attackers.
3. **Insufficient Input Validation**
    - Usernames, commands, and message fields are not sanitized before processing.

      An attacker could inject malicious payloads (e.g., control characters or oversized messages) to crash a node or disrupt message loops.
4. **DoS via Persistent Connections**
    - Clients that disconnect improperly leave a residual state on the server.

      A malicious actor could connect repeatedly with new IDs to consume memory and exhaust resources.
5. **Lack of Rate Limiting**
    - No obvious mechanism prevents a client from spamming commands (/tell, /list) in a tight loop, potentially causing high CPU usage.
6. **Verbose Logging of Identifiers**

    - The logs print internal IDs, host addresses, and connection details. In a real deployment, this could leak topology information useful to attackers.

7. **Missing Session Authentication**

    - Clients connect and identify only by UUIDs. Without cryptographic session tokens or challenge-response authentication, session hijacking is possible.

---

5. Suggestions for Hardening

- Implement signature validation for all peer JOIN and message packets.
- Encrypt client-server WebSocket traffic (using TLS or symmetric session keys negotiated via RSA).
- Sanitize and validate user inputs before processing commands.
- Add heartbeat timeouts to detect dead clients and free their IDs.
- Introduce basic rate limiting or command throttling.
- Minimize sensitive information in log output.

---

6. Final Thoughts

Group 45 has created a technically ambitious prototype that demonstrates an excellent understanding of distributed system communication and modular Python design.

Although the core functions still need stability and proper cleanup logic, the security foundation is clearly there.

By strengthening peer verification and session controls, this system could evolve into a solid model for a secure, federated chat architecture.