CIS 505 Software Systems Project 3

Yisi Xu, Dongni Wang, Wei Hu

March 24, 2016

1 Project Overview

Project Name: "dchat"
Operating System: Linux
Programming Language: C or C++
Network Protocol: fully-ordered, centralized multi-cast protocol on top of UDP datagrams (via standard socket interface).
Project Basics: fully distributed, text-based, group "chat" system.

The system allows:

- arbitrary size groups of Internet users to send and receive messages to the group in (approximately) real time.
- users in a group to type messages at any time, and the messages typed by the various group members should be received by all members in the same order.
- users either start a new chat group (thereby becoming the initial member) or join an existing chat group (by connecting to any member of a the group they want to join).
- users to leave the group they are in at any time, even if they started the group. The chat continues with the remaining members (if any are left).

2 Design Specification

2.1 Basic Functionality (later in User Manual)

- **Start a new chat group:** `$$ dchat USER`
- **Join an existing chat:** `$$ dchat USER ADDR:PORT`
- **Send a chat message:** `MESSAGE on stdin`
- **Exit the chat:** `EOF on stdin` (control-D)

2.2 Members and the Sequencer

One chat group member is a normal user-invoked "dchat" client who joined a chat group. A member could send, receive messages within a group or exit a chat group by stop the program. A member will constantly ping the sequencer to verify its existence. If a sequencer is detected missing, the member can be elected to be a Sequencer by Bully Algorithm and turn into a Sequencer.

One chat group member will also be serving as a sequencer. The user-invoked "dchat" client who initialized a chat group would be the default Sequencer. The Sequencer has all the functionalities of a Member and also will be in charge of assigning message sequence numbers, detect group member failures, etc. The Sequencer will maintain a sent_list in case any message delivery failure exists.The user who initialized

a chat group will be the default sequencer. And we will use Bully Algorithm to elect new sequencer if the current one leaves or crashes.

At the same time, the end users should not be able to tell if it has been selected as the Sequencer or not.

2.3 Failure Detector

- Message delivery failure
- Failure of other clients: leader constantly contact each client, if no response
- Failure of the leader: leader election (Constantly contact leader, if no response received)

2.4 Leader Election

All processes know other process' ids When a process finds the coordinator has failed (via the failure detector)

- If it knows its id is the highest, it elect itself as coordinator, then sends a Coordinator message to all processes with lower identifiers. Election is complete.
- Else it initializes an election by sending Election message.
  - Only to processes that have a higher id than itself
  - If receive no answer within timeout, calls itself leader and sends Coordinator messages to all processes with lower id
  - Else wait for Coordinator message

A process receives an Election message replied with OK, and start its own leader selection protocol (unless it has already done so)

2.5 Data Structures

```
class user {
private:
    string username;
    string ip_addr;
    int port;
    bool isLeader;
    string[] client_list
    string[] message_queue
public:
};
```

2.6 Message Formats

Each message has a sequence number
Join: send message to any client, this client send leaders information to new user
Leave: Client send leaving message to leader, leader update client list and send this list to clients
Chat message from client to leader: user, content of message, local sequence number
Chat message from leader broadcast to clients, message sequence number
Leader contact client to detect client failure
Leader send client list to each client
Client contact leader
Client start leader election
Other clients response to leader election message

Length: 512 bytes maximum?
First portion: Message type

Second portion: one bit indicating if it is the end of a message or expecting more?
Third portion: Message body.

2.7 Detailed Mechanism

Start a new chat group:

1. Initialize as a user with sequencer privilege. Add itself to the client list.
2. Start working as a Sequencer (start Sequencer Routine) until it crashes or exits. Note also it could still participate in the chat as a normal Member.

Join an existing chat:

1. Send a join request to any member of a chat group. If contacted a member: receive the Sequencer ip and port. Send the join request to the Sequencer.
2. Anticipate join confirmation from the Sequencer. If no confirmation was received in time, resend the join request. If the connection failed, end the join attempt, display error message and exit.
3. Continue as a Member (start Member Routines) after receive the join confirmation.

Send a message to the group:

1. Send the message to the Sequencer.
2. Wait for the broadcast for confirmation, otherwise resend after time-out(with local sequence number).

Sequencer Routine:

1. Constantly ping the Members to verify their existence: if any Member is detected missing, update the client list, and broadcast a Exit Notice (with message sequence number, new member name and its ip and port) to every other Members.
2. Listening to incoming messages: parse incoming message, determine its type and handle it.
3. Handles Join Request: send confirmation message (including the current client_list) to the requester, broadcast a Join Notice (with message sequence number, new member name and its ip and port) to every current Member about the new member.
4. Handles Messages: (check local sequence number to eliminate duplicates) assign a sequence number to each message. If it is able to be sent, send immediately and add it to the sent queue, otherwise add it to the message_queue.

Member Routine:

1. Constantly ping the Sequencer to verify its existence: if it exists, do nothing. Otherwise start the election.
2. Listening to incoming messages: parse incoming message, determine its type, verify the message sequence number if needed and handle it.
3. Handles Join Request: send redirect message to the request, include the ip and port of the current Sequencer.
4. Verify Sequence Number: if is passed the verification, proceed to handling the Join/Exit Notice or Message. Otherwise, it queues up the current message and send a Message Request to the Sequencer.
5. Handles Join Notice: verify the message sequence number, add the new Member to the client_list.
6. Handles Exit Notice: verify the message sequence number, delete the Member from the client_list.
7. Handles Messages: verify the message sequence number, display the message.

3 Detailed Log

Mar. 24 First Group Meeting: Design protocol

- data structure (user), failure detection (heartbeats), leader election (Bully Algorithm), message formats, basic mechanism.

4 Proposed Schedule