Cloud P2P Project with Raft Leader Election

This project implements a distributed image encryption system with Raft consensus algorithm for leader election.

Architecture

- **Multiple Servers**: Run 3+ servers that use Raft to elect a leader
- **Client**: Discovers and connects to the leader server for image encryption
- **P2P Viewing**: Clients can view encrypted images peer-to-peer with embedded access control

Key Features

- **Raft Consensus**: Automatic leader election with fault tolerance
- **Leader Discovery**: Client automatically finds the current leader
- **LSB Steganography**: Hides metadata and access control inside images
- **Quota Management**: Track view counts per user

Setup

1. Build the Project

```
bash

cargo build --release
```

2. Create Required Files

servers.conf (list of all servers):

```
127.0.0.1:8080
127.0.0.1:8081
127.0.0.1:8082
```

unified_image.png: Place this file in each server's working directory. This is the "Access Denied" image shown to unauthorized users.

3. Set Environment Variable for Logging

```bash
# Linux/Mac
export RUST_LOG=info

# Windows
set RUST_LOG=info
```

Running the System

Start Servers

Each server needs:

- Port number
- Unique server ID
- List of peer addresses

Terminal 1 - Server 1:

```bash
cargo run --bin server -- 8080 server1 127.0.0.1:8081 127.0.0.1:8082
```

Terminal 2 - Server 2:

```bash
cargo run --bin server -- 8081 server2 127.0.0.1:8080 127.0.0.1:8082
```

Terminal 3 - Server 3:

```bash
cargo run --bin server -- 8082 server3 127.0.0.1:8080 127.0.0.1:8081
```

Note: Servers use two ports:

- Main port (8080, 8081, 8082) for client requests
- Raft port (9080, 9081, 9082) for consensus messages

Leader Election Process

Watch the server logs - you'll see:

1. Servers start as **Followers**
2. After election timeout (3-6 seconds), one becomes a **Candidate**
3. Candidate requests votes from peers
4. Server with majority votes becomes **Leader** 🎉
5. Leader sends periodic heartbeats to maintain leadership

Encrypt an Image

```
bash

cargo run --bin client -- encrypt --input your_image.png --owner alice
```

The client will:

1. Try each server from `servers.conf`
2. Find the current leader
3. Send the image to the leader
4. Receive and save `encrypted_lsb_image.png`

View an Encrypted Image

```
bash

cargo run --bin client -- view --input encrypted_lsb_image.png --user alice
```

Results:

- **Authorized**: Saves `viewable_image.png` and decrements quota
- **Unauthorized**: Saves the "Access Denied" image

Raft Protocol Details

States

- **Follower**: Passive, waits for leader heartbeats
- **Candidate**: Actively seeking votes
- **Leader**: Processes client requests, sends heartbeats

Timeouts

- **Election Timeout**: 3-6 seconds (randomized to avoid split votes)
- **Heartbeat Interval**: 1 second

Leader Failover

If the leader crashes:

1. Followers stop receiving heartbeats
2. After election timeout, a follower becomes candidate
3. New leader is elected automatically
4. Clients automatically discover the new leader

Testing Leader Election

Test 1: Normal Operation

1. Start all 3 servers
2. Wait for leader election
3. Encrypt an image through the leader

Test 2: Leader Failure

1. Start all 3 servers and note the leader
2. Kill the leader process (Ctrl+C)
3. Watch remaining servers elect a new leader
4. Try encrypting - client finds new leader automatically

Test 3: Split Brain Prevention

1. Start all 3 servers
2. Network partition simulation: Kill one server
3. Remaining 2 servers maintain leadership (majority still exists)
4. Client can still connect to leader

Configuration

Modify Timeouts (in `lib.rs`)

```rust
let raft_config = RaftConfig {
    election_timeout_min: 3000,  // 3 seconds
    election_timeout_max: 6000,  // 6 seconds
    heartbeat_interval: 1000,    // 1 second
    // ...
};
```

Change Ports

Update `servers.conf` and restart all servers with matching parameters.

Troubleshooting

"Address already in use"

- Kill previous server instances
- The `socket2` crate with `set_reuse_address(true)` should prevent this

"No leader elected yet"

- Wait 3-6 seconds for election to complete
- Check that all servers are running
- Ensure majority (2 out of 3) servers are online

"Connection timeout"

- Verify `servers.conf` addresses match running servers
- Check firewall settings

No logs appearing

- Set `RUST_LOG=info` environment variable
- Use `RUST_LOG=debug` for more detailed logs

Project Structure

```
src/
├── lib.rs          # Shared types and structures
├── lsb.rs          # Steganography implementation
├── raft.rs         # Raft consensus algorithm
├── server.rs       # Server with Raft integration
└── client.rs       # Client with leader discovery

Cargo.toml          # Dependencies
servers.conf        # Server addresses
unified_image.png   # Access denied image
```

How It Works

Encryption Flow

1. Client reads image and creates permissions
2. Client queries servers to find leader
3. Leader embeds permissions + unified image via LSB steganography
4. Client receives and saves encrypted image

Viewing Flow

1. Viewer loads encrypted image
2. Extracts permissions and unified image from LSB
3. Checks quota for current user
4. If authorized: shows actual image, decrements quota
5. If denied: shows unified "Access Denied" image

Raft Consensus

1. All servers start as followers
2. Random election timeout triggers candidate state
3. Candidate requests votes from peers
4. Server receiving majority votes becomes leader
5. Leader maintains authority via heartbeats
6. If leader fails, new election begins automatically

Advanced Usage

Run with 5 Servers

servers.conf:

```
127.0.0.1:8080
127.0.0.1:8081
127.0.0.1:8082
127.0.0.1:8083
127.0.0.1:8084
```

Start each server:

```bash
cargo run --bin server -- 8080 server1 127.0.0.1:8081 127.0.0.1:8082 127.0.0.1:8083 127.0.0.1:8084
# ... repeat for other servers
```

Custom Permissions

Modify `client.rs` in `handle_encrypt()`:

```rust
let mut quotas = HashMap::new();
quotas.insert("alice".to_string(), 5);
quotas.insert("bob".to_string(), 3);
quotas.insert("charlie".to_string(), 1);
```

License

This is an educational project demonstrating distributed systems concepts.