

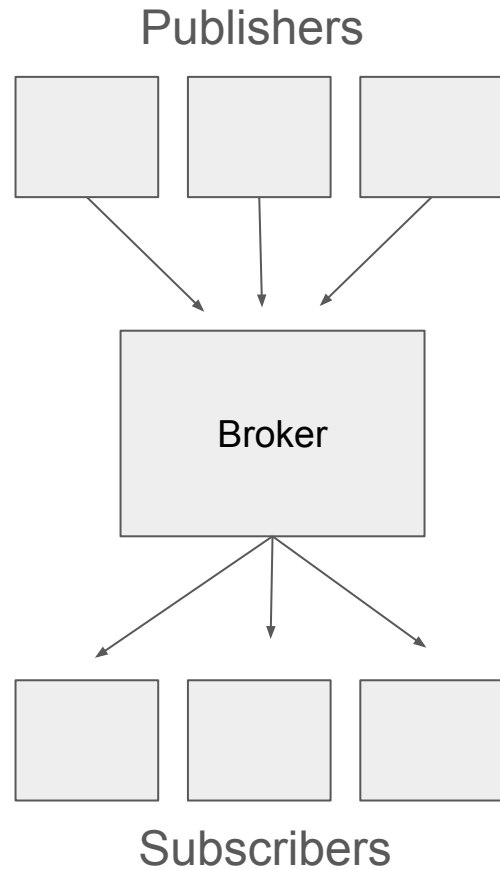
MQTT broker in C

Eduardo Sandalo Porto

*Department of Computer Science
University of São Paulo*


Project overview

- Implement a very simple MQTT broker
 - TCP, PUB/SUB protocol for IoT
 - Clients subscribe to topics, other clients publish to topics
 - Broker relays messages to subscribers
- Key MQTT operations:
 - Client connection (**CONNECT** packet)
 - Topic subscription (**SUBSCRIBE** packet)
 - Message publishing (**PUBLISH** packet)
 - Topic unsubscription (**UNSUBSCRIBE** packet)
 - Client disconnection (**DISCONNECT** packet)
 - (bonus) Keep-alive pings (**PINGREQ** packet)
- Skip AUTH, QoS, encryption, ...




Architecture

- Concurrency model
 - Each client gets a process
 - Each subscription gets another process
- Messaging and state management
 - Temporary base directory to manage state (`/tmp/temp.mac5910`)
 - Each client gets a subdirectory from base with its process ID
 - Each subscription is managed by a named pipe in client's directory
 - Limitation: separating by / (slashes) harder (did not implement)

```
→ / ls -R /tmp/temp.mac5910.1.  
5340 5398
```

```
/tmp/temp.mac5910.1./5340:  
topico1 topico2
```

```
/tmp/temp.mac5910.1./5398:  
topico1 topico3  
→ / 
```

Handling connections and subscriptions

- First packet must always be CONNECT
 - Followed by CONNACK
- When a client sends the SUBSCRIBE packet:
 - If the topic file already existed, then the client is already subscribed. Ignore.
 - If it didn't:
 - 1. Create a topic file (named pipe) in the client's directory
 - 2. Fork a process to read the pipe
 - 3. Use `select` on the pipe to read or check with a timeout if it has been deleted
 - Subscription processes can stop on their own when a pipe is deleted
 - Subscription processes allow clients to send more packets (non-blocking)
- When a client sends the UNSUBSCRIBE packet:
 - Delete the topic file, automatically stopping the subscription process

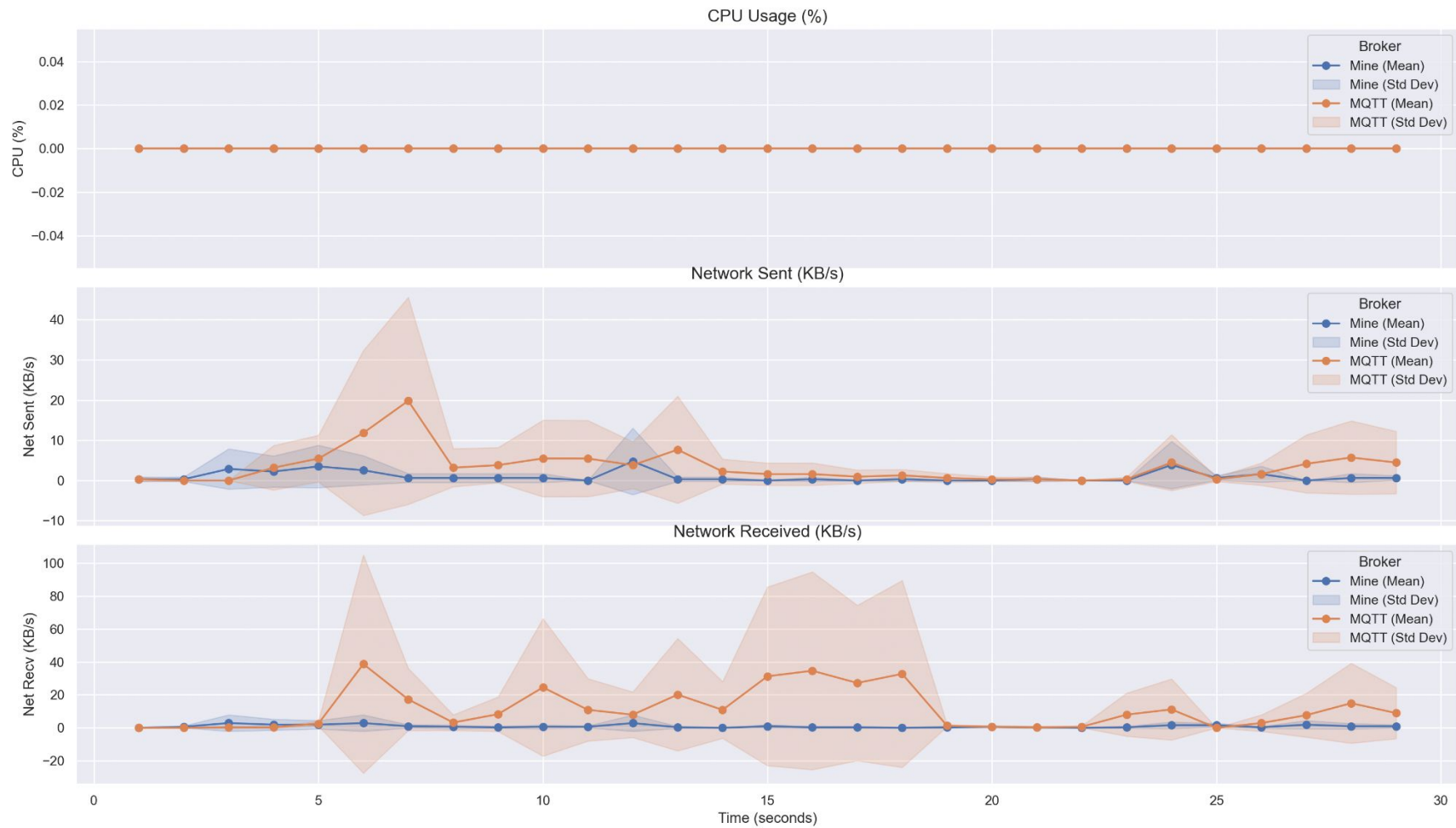
Handling publishing, disconnecting, and pinging

- When a client sends a PUBLISH packet:
 - Broker will search for all named pipes whose name include the topic
 - `/tmp/temp.mac5910/<id>/<topic>`
 - Broker will write the message to each pipe
 - Subscriber processes will handle sending messages to subscribers
 - Weird behavior: PUBLISH packets may be followed by 2-byte DISCONNECTs
- When a client sends a DISCONNECT packet:
 - Delete their folder, automatically stopping all subscriber processes (where applicable)
- When a client sends a PINGREQ packet:
 - Just answer with a PINGRESP

Performance analysis: setup

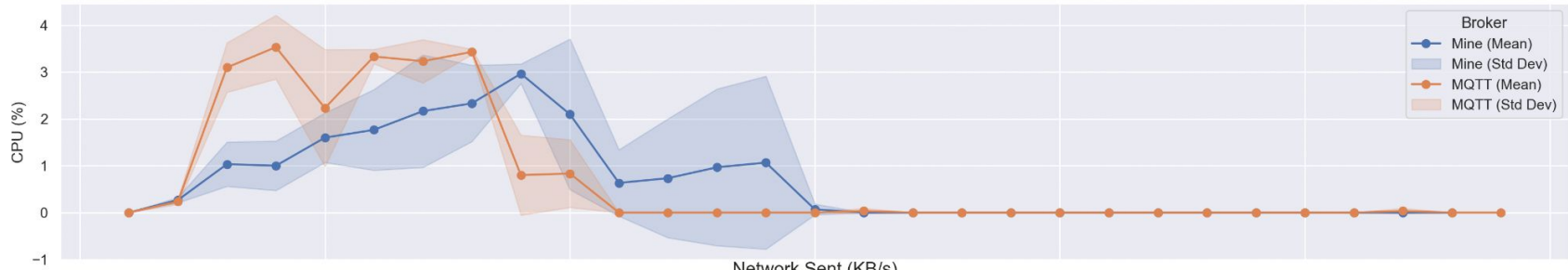
- Three experiments comparing CPU and net usage with *mosquitto*:
 - 1. Idle server with no clients connected
 - 2. 50 subscribers, with 50 publishers each sending 3 PUBLISH
 - 3. 500 subscribers, with 500 publishers each sending 3 PUBLISH
- Test environment:
 - Broker: MacBook Air M2, 16GB RAM, macOS 15.6.1
 - Clients: Dell Vostro 15 5510, 20 GB RAM, Ubuntu 24.04 LTS through WSL2
 - Router: Samsung Galaxy S20 FE (yes, a smartphone!)
- Methodology
 - Automated scripts in Python to collect data through `psutil`
 - Metrics: CPU usage and network throughput
 - Results are the mean of three executions with standard deviation

Mean Performance with Standard Deviation - Experiment 1

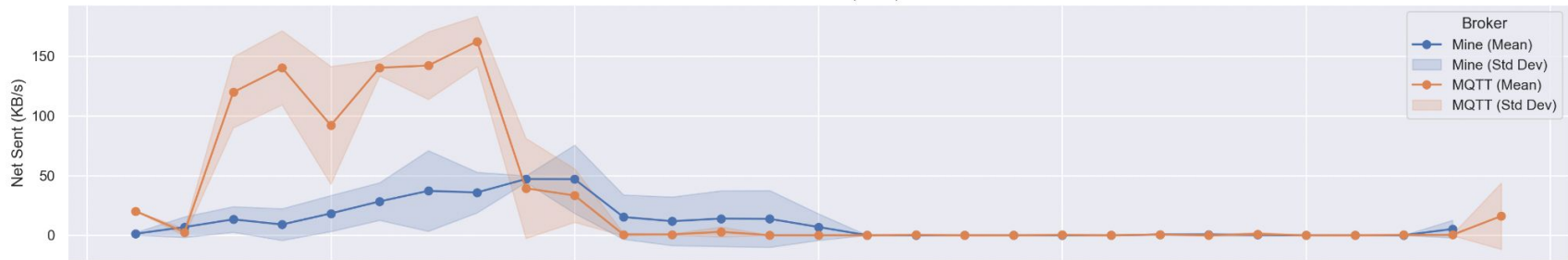


Mean Performance with Standard Deviation - Experiment 2

CPU Usage (%)



Network Sent (KB/s)

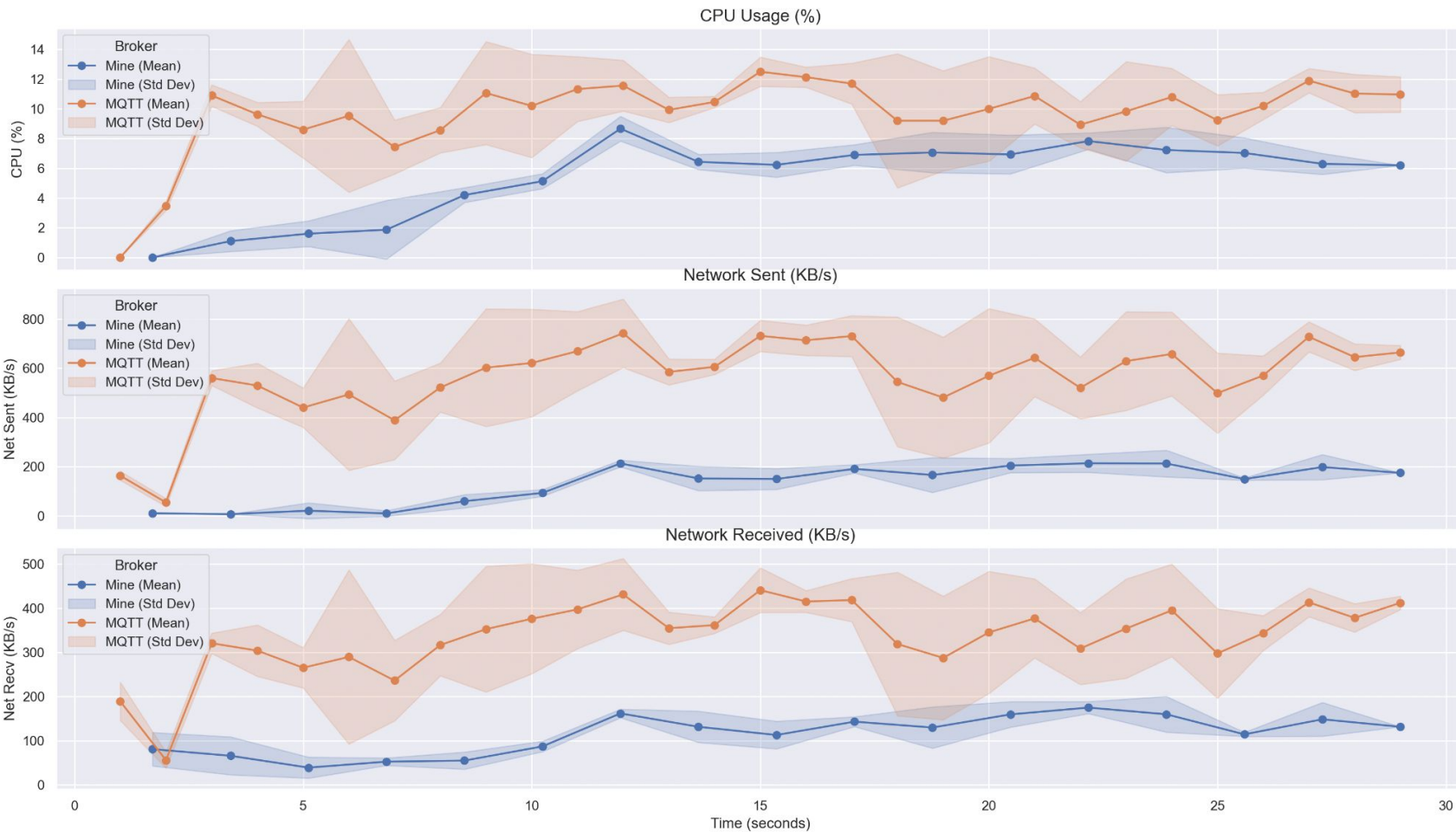


Network Received (KB/s)



Time (seconds)

Mean Performance with Standard Deviation - Experiment 3



Strengths and weaknesses

- Strengths

- Fork and pipe-based is easy to implement
- Isolates clients quite well, *hopefully* not causing failure in a client to affect another

- Weaknesses

- Very poor scalability
 - As more and more processes are created, the slower everything gets
- Mass process creation slows down the system due to priority shifting
 - Can be seen in the last experiment: less analytics generated, since the analytics process got less priority
- File system I/O is always slower than in-memory

Conclusion

- The MQTT implementation works, although is slow or fails with large amounts of connections
- Design is simple, but not scalable nor performant
- Unable to match *mosquitto*