AY 2023/2024

Politecnico di Milano

Group Monitoring in Mobile IoT

Rishabh Tiwari - 10987397

Professor
Luca Mottola

Version 1.1
February 27, 2024

Contents

1 Introduction

1.1 Description of the project

In this project, I have implemented a system where IoT devices carried by individuals detect when they are close to each other, using radio signals as a proximity sensor. When two devices come within range, it means those individuals are "in contact". If three or more people are in contact, forming a "group", it immediately reports this to the back-end. Similarly, when someone joins or leaves a group, it updates the group's size in the back-end. In this project, I also calculate and update statistics about each group, including its lifetime and the average, maximum, and minimum number of members over time.

1.2 Assumptions and Guidelines

- The IoT devices are assumed to be constantly reachable, even across multiple hops, from a single static IoT device that acts as an IPv6 border router. This means network partitions need not be considered.
- The IoT component of the project can be developed and tested entirely using the COOJA simulator. For simulating mobility, nodes can simply be moved around manually.
- Monitoring of the group membership should be implemented with periodic functionality. The scenario where a person leaves a group can be handled with a timeout mechanism.

1.3 Technologies and Implementation

This project has been developed using two technologies Contiki-NG and Node-RED where I will briefly describe their importance in the project and how they have been integrated to perform the tasks.

2 Overall Structure

2.1 COOJA

In this project, COOJA simulator is a key tool used to test how IoT devices carried by people can interact. COOJA lets us run a simulation without worrying about how much memory the software needs, which is great because it means we can do more complex tests. It uses a special setting called the Constant Loss Unit-Disk Graph Model to make sure messages between devices close to each other always get through.

For the project, COOJA shows us how these IoT devices, like a border router or a signaler, talk to each other using the UDP protocol. This is important because it helps to make sure that the main system, which is kind of like the project's control center, can always keep track of the devices. It doesn't matter where they are or how they move around; the system always knows what's going on thanks to COOJA. This helps the project figure out when people come together to form groups and how these groups change over time.

2.2 RPL Border-Router

The rpl border-router mote is a crucial piece of the puzzle. Think of it as the big boss that connects all the small, wireless sensors in the project to the wider world of the internet. It's set up to be the leader or the root of the network, guiding all the data from these sensors through one main path to the internet. This means it's the gatekeeper that makes sure all the information from our sensors can talk to the outside world smoothly. It's like having a single, efficient bridge between our little network of devices and the vast ocean of the internet.

2.3 UDP Signaler

The UDP signaler plays a pivotal role in managing the dynamic interactions among people carrying IoT devices. As these devices move around, forming and dissolving groups based on proximity, the UDP signaler acts as the communicator, efficiently transmitting signals to update the network about changes in group composition. The UDP signaler's role is fundamental in maintaining an effective and responsive monitoring system within the mobile IoT environment.

2.4 MQTT-UDP mote

In this project, the MQTT UDP mote plays a crucial role by handling both MQTT and UDP communications efficiently. It uses a technique called protothreads, allowing it to manage these two types of communications simultaneously. For MQTT, it operates on a simple mechanism where it periodically checks for messages and handles connections with the internet through a broker system. Each mote subscribes to a unique topic, which is essentially a specific channel for receiving messages directly relevant to it. This setup enables direct and efficient communication between the back-end system and each individual mote, ensuring that messages are delivered promptly and accurately.

On the UDP side, each mote acts as both a sender and receiver. They send out simple messages to their neighbors and listen for incoming messages. When a mote receives a message, it triggers a function that responds back, facilitating a dynamic and responsive network of devices. This dual capability of acting as both client and server in the UDP communication allows for a robust network where devices continuously update each other about their presence and status. The integration of MQTT for internet-based communication and UDP for local network chatter makes these motes highly effective in maintaining a seamless flow of information.

3 Implementation

This section deals with the functionalities and their description of what and how they are implemented on the front end and the backend part of Contiki-NG and Node-RED part.

3.1 Frontend (Contiki-NG - COOJA)

1. **UDP Client Process (`udp_client_process`):**
   - This process is responsible for sending UDP packets to neighboring nodes in the network.
   - It periodically sends UDP packets to discover neighboring nodes and update the contact list.

    – Upon receiving UDP packets from neighboring nodes, it triggers the `udp_rx_callback` function to update the contact list and check for group formation.

2. **MQTT Client Process (`mqtt_client_process`):**
    - This process manages the MQTT client's connection, subscription, and message publishing.
    - It handles MQTT events such as connection, disconnection, message publish, and subscription acknowledgment.
    - Implements the state machine to manage the MQTT client's state transitions, including initialization, connection, disconnection, etc.

3. **Contact Management Functions:**
    - Functions such as `update_contact`, `add_to_mutual_contacts_if_missing`, `update_all_mutual_contacts`, etc., manage the contact list and mutual contacts between IoT devices.
    - These functions update the contact list upon receiving UDP packets and maintain information about contact activity and mutual contacts.

4. **Group Formation and Reporting Functions:**
    - The `report_group_formation` function reports group formation to the backend server via MQTT messages.
    - The `check_group_formation` function periodically checks if a group can be formed based on the contact information.

5. **Configuration Functions:**
    - Functions like `init_config`, `update_config`, `construct_pub_topics`, `construct_sub_topic`, `construct_client_id` initialize and update MQTT client configurations, topics, and client ID.
    - These functions set up MQTT client configurations such as organization ID, type ID, authentication token, broker IP, etc.

6. **Helper Functions:**
    - Various helper functions like `trim_ip_addr`, `add_ip_to_json_array`, `add_ip_to_json_array_recursive`, etc., assist in formatting MQTT messages, IP address manipulation, JSON array construction, etc.

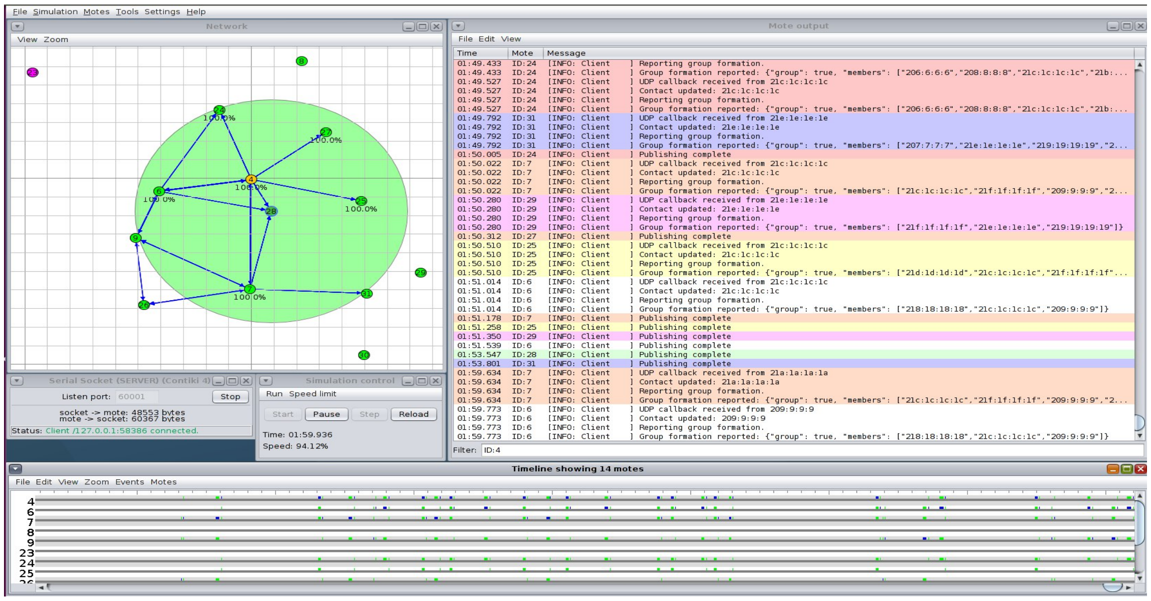
3.2 Backend (Node-RED)

This section outlines the features and checks implemented as part of a Node-RED function node designed to manage groups of entities within a system. The following functionalities are included:

1. **Group Cardinality Updates**: Updates the number of active members within a group.
2. **Lifetime Tracking**: Calculates the lifetime of a group from creation to the current moment.

3. **Cardinality Statistics**: Maintains statistical data for each group, tracking the maximum, minimum, and average number of members over time, along with update counts.
4. **Periodic Monitoring**: Includes a function for periodic monitoring, checking for inactive members based on a timeout threshold and updating group statistics.
5. **Timeout Management**: Filters out members who have been inactive beyond a specified threshold (default 60 seconds).
6. **Group Dismantling**: Dismantles groups that fall below the minimum member count, resetting their statistics and logging warnings.
7. **Member Activity Updates**: Updates the last active time for the sender within their groups upon message receipt.
8. **Membership Changes**: Checks for and updates changes in group membership.
9. **New Group Creation**: Creates new groups for senders not part of any existing group.
10. **Array Comparison**: Includes a function to check if two arrays are equal, to detect changes in group memberships.
11. **Group Survivability Check**: Handles the survivability of groups by dismantling those with insufficient members.
12. **Main Execution Flow**: Processes group memberships, manages group survivability, and logs dismantled group messages.
13. **Message Handling**: Expects a message payload containing results from a process named *cooja*.
14. **Context Management**: Uses a context object to maintain group data across function executions.

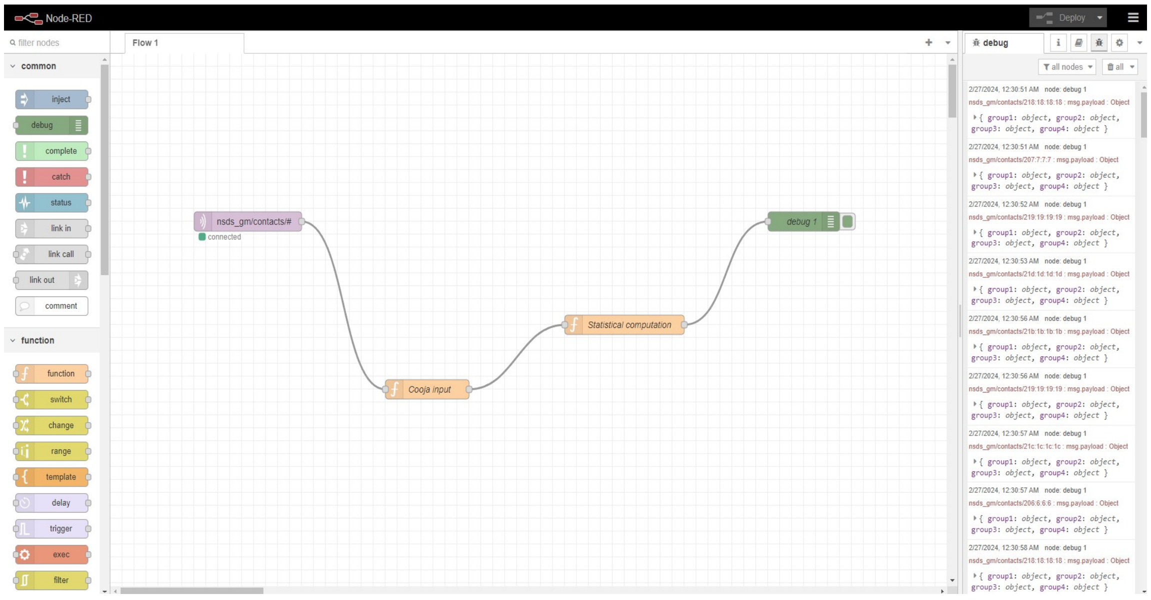
4 Results



File Simulation Motes Tools Settings Help
Network
View Zoom
100.0%
Serial Socket (SERVER) (Contiki 4)
Listen port: 60001
Stop
socket -> mote: 48553 bytes
mote -> socket: 60367 bytes
Status: Client /127.0.0.1:58386 connected.
Simulation control
Run Speed limit
Start
Pause
Step
Reload
Time: 01:59.936
Speed: 94.12%
Mote output
File Edit View
Timeline showing 14 motes
File Edit View Zoom Events Motes

```
Time       Mote   Message
01:49.433  ID:24  [INFO: Client    ] Reporting group formation.
01:49.433  ID:24  [INFO: Client    ] Group formation reported: {"group": true, "members": ["206:6:6:6","208:8:8:8","21c:1c:1c:1c","21b:...
01:49.527  ID:24  [INFO: Client    ] UDP callback received from 21c:1c:1c:1c
01:49.527  ID:24  [INFO: Client    ] Contact updated: 21c:1c:1c:1c
01:49.527  ID:24  [INFO: Client    ] Reporting group formation.
01:49.527  ID:24  [INFO: Client    ] Group formation reported: {"group": true, "members": ["206:6:6:6","208:8:8:8","21c:1c:1c:1c","21b:...
01:49.792  ID:31  [INFO: Client    ] UDP callback received from 21e:1e:1e:1e
01:49.792  ID:31  [INFO: Client    ] Contact updated: 21e:1e:1e:1e
01:49.792  ID:31  [INFO: Client    ] Reporting group formation.
01:49.792  ID:31  [INFO: Client    ] Group formation reported: {"group": true, "members": ["207:7:7:7","21e:1e:1e:1e","219:19:19:19","2...
01:50.005  ID:24  [INFO: Client    ] Publishing complete
01:50.022  ID:7   [INFO: Client    ] UDP callback received from 21c:1c:1c:1c
01:50.022  ID:7   [INFO: Client    ] Contact updated: 21c:1c:1c:1c
01:50.022  ID:7   [INFO: Client    ] Reporting group formation.
01:50.022  ID:7   [INFO: Client    ] Group formation reported: {"group": true, "members": ["21c:1c:1c:1c","21f:1f:1f:1f","209:9:9:9","2...
01:50.280  ID:29  [INFO: Client    ] UDP callback received from 21e:1e:1e:1e
01:50.280  ID:29  [INFO: Client    ] Contact updated: 21e:1e:1e:1e
01:50.280  ID:29  [INFO: Client    ] Reporting group formation.
01:50.280  ID:29  [INFO: Client    ] Group formation reported: {"group": true, "members": ["21f:1f:1f:1f","21e:1e:1e:1e","219:19:19:19"]}
01:50.312  ID:27  [INFO: Client    ] Publishing complete
01:50.510  ID:25  [INFO: Client    ] UDP callback received from 21c:1c:1c:1c
01:50.510  ID:25  [INFO: Client    ] Contact updated: 21c:1c:1c:1c
01:50.510  ID:25  [INFO: Client    ] Reporting group formation.
01:50.510  ID:25  [INFO: Client    ] Group formation reported: {"group": true, "members": ["21d:1d:1d:1d","21c:1c:1c:1c","21f:1f:1f:1f"...
01:51.014  ID:6   [INFO: Client    ] UDP callback received from 21c:1c:1c:1c
01:51.014  ID:6   [INFO: Client    ] Contact updated: 21c:1c:1c:1c
01:51.014  ID:6   [INFO: Client    ] Reporting group formation.
01:51.014  ID:6   [INFO: Client    ] Group formation reported: {"group": true, "members": ["218:18:18:18","21c:1c:1c:1c","209:9:9:9"]}
01:51.178  ID:7   [INFO: Client    ] Publishing complete
01:51.258  ID:25  [INFO: Client    ] Publishing complete
01:51.350  ID:29  [INFO: Client    ] Publishing complete
01:51.539  ID:6   [INFO: Client    ] Publishing complete
01:53.547  ID:28  [INFO: Client    ] Publishing complete
01:53.801  ID:31  [INFO: Client    ] Publishing complete
01:59.634  ID:7   [INFO: Client    ] UDP callback received from 21a:1a:1a:1a
01:59.634  ID:7   [INFO: Client    ] Contact updated: 21a:1a:1a:1a
01:59.634  ID:7   [INFO: Client    ] Reporting group formation.
01:59.634  ID:7   [INFO: Client    ] Group formation reported: {"group": true, "members": ["21c:1c:1c:1c","21f:1f:1f:1f","209:9:9:9","2...
01:59.773  ID:6   [INFO: Client    ] UDP callback received from 209:9:9:9
01:59.773  ID:6   [INFO: Client    ] Contact updated: 209:9:9:9
01:59.773  ID:6   [INFO: Client    ] Reporting group formation.
01:59.773  ID:6   [INFO: Client    ] Group formation reported: {"group": true, "members": ["218:18:18:18","21c:1c:1c:1c","209:9:9:9"]}
Filter: ID:4
```

Figure 1: Cooja Simulation



```
2/27/2024, 12:30:51 AM   node: debug 1
nsds_gm/contacts/218:18:18:18 : msg.payload : Object
{ group1: object, group2: object, group3: object, group4: object }
2/27/2024, 12:30:51 AM   node: debug 1
nsds_gm/contacts/207:7:7:7 : msg.payload : Object
{ group1: object, group2: object, group3: object, group4: object }
2/27/2024, 12:30:52 AM   node: debug 1
nsds_gm/contacts/219:19:19:19 : msg.payload : Object
{ group1: object, group2: object, group3: object, group4: object }
2/27/2024, 12:30:53 AM   node: debug 1
nsds_gm/contacts/21d:1d:1d:1d : msg.payload : Object
{ group1: object, group2: object, group3: object, group4: object }
2/27/2024, 12:30:56 AM   node: debug 1
nsds_gm/contacts/21b:1b:1b:1b : msg.payload : Object
{ group1: object, group2: object, group3: object, group4: object }
2/27/2024, 12:30:56 AM   node: debug 1
nsds_gm/contacts/219:19:19:19 : msg.payload : Object
{ group1: object, group2: object, group3: object, group4: object }
2/27/2024, 12:30:57 AM   node: debug 1
nsds_gm/contacts/21c:1c:1c:1c : msg.payload : Object
{ group1: object, group2: object, group3: object, group4: object }
2/27/2024, 12:30:57 AM   node: debug 1
nsds_gm/contacts/206:6:6:6 : msg.payload : Object
{ group1: object, group2: object, group3: object, group4: object }
2/27/2024, 12:30:58 AM   node: debug 1
nsds_gm/contacts/218:18:18:18 : msg.payload : Object
{ group1: object, group2: object, group3: object, group4: object }
```

Figure 2: Node-RED flow

In the above Figure 1, we can see the simulator where all the motes are connected to the motes nearby or at 1-hop distance. We can see that the console option is printing the log statements such as Reporting group formation, group formation reported along with their group information and cardinality shows the group formation logic has been correctly implemented. We can also see if a new mote is to the existing list, it prints the message as contact updated and returns the new group list which determines the objective or this entire project. So it was a successful operation where the motes, if connected to each other, reports their group formation or group disband to the backend.

In the second Figure 2, we can see the node-red flow where we configure the MQTT broker with our IP address and port settings. The QoS of the server is set to one so that it will ping back once when it receives a message and the topic is assigned as nsds_gm/contacts. Then we have the cooja input which is responsible to clean the received function and goes through all the members in the list. Then with the help of the statistical computation function I calculate all the statistics and finally prints using the debug. On the console log we can see that it is correctly sharing the output of the information received from the front end and also computes the relevant information requested in the projects requirements.

So in the shared image, we can see for example in the first image, mote 7 reports that the group is formed and reports the group information to the backend. The backed then prints the information along with the statistics in the debug console of node-red.

5 Conclusion and Future Work

The project "Group Monitoring in Mobile IoT" has successfully demonstrated the integration and application of Contiki-NG and Node-RED in monitoring group dynamics among mobile IoT devices. While the project outcomes have been promising, showcasing the potential of IoT technologies in real-time monitoring and management, there remains ample scope for enhancement and deeper exploration. I Wwould like to discuss potential future improvements for project's execution.

Future Improvements

(a) **Scalability and Performance Optimization:** The system can be better at handling more devices without slowing down. This means it can grow bigger and still work smoothly.

(b) **Advanced Mobility Models:** Implementing more sophisticated mobility models within the COOJA simulator would provide insights into system performance under varied real-world scenarios, enhancing the system's adaptability and reliability.

(c) **Enhanced Security Features:** It is really important to keep all the data safe from hackers. Stronger security measures can be added to protect all the information that passes through the system.

(d) **User Interface (UI) Enhancements:** Developing an intuitive and comprehensive user interface would significantly improve the usability of the system, enabling more effective monitoring and management of the IoT network.