Group 6 - Final Project Report

1. Group members

Davindran A0167009E

Low Jun Wei A0168303J

Shanon Seet A0172753B

2. Introduction

In this project, our group aims to build a tracing application running on a TI CC2650 SensorTag, similar to the TraceTogether token released by the Singapore government. Our tracing device would make use of the SensorTag's IEEE 802.15.4 (ZigBee) radio.

Our application will be able to detect two devices in close proximity, defined as being within 3 meters of each other. Through its print output, users will be able to see when a nearby device is detected, when a nearby device moves away, and an estimated duration of contact. Our device is also able to constantly keep track of how many devices are within close proximity. This will be done while ensuring power consumption is minimised.

Our group tackled this project by breaking it down into smaller steps. First, we experimented by implementing a probabilistic neighbour discovery protocol so that 2 devices can discover each other with adequate probability while sending packets indicating their individual node IDs. The RSSI value and timestamp of the received packet is calculated in real time by the device who received the packets. Then, we determined an appropriate duty cycle to minimise power consumption, while ensuring discovery happens within 30 seconds with high probability. Next, we implemented an algorithm that keeps track of the node IDs detected and stores their first and most recently received packet times. Then, we added logic to check if there are any stored node IDs from which the device has not received packets for more than 30 seconds, following which these nodes are considered to have 'left'. We further implemented an appropriate RSSI threshold to define proximity logic, wherein nodes sending messages with RSSI values lower than the threshold are also considered 'left'. Finally, we looked at refining the RSSI estimation in different environments.

In this report, we will discuss our neighbour discovery protocol, how we implemented proximity detection and our evaluation of the system. We will also cover the challenges faced along the way, lessons learnt and the contributions made by each member.

3. Neighbour discovery protocol

3.1. Probabilistic "birthday" protocol

Our group decided to implement a probabilistic "birthday" protocol for neighbour discovery as the requirements only required a high probability of discovery with no hard upper bound for discovery time. A probabilistic approach would also be able to minimise power consumption while satisfying the requirements.

The protocol we implemented closely follows the neighbour discovery protocol provided in Assignment 4. Each node wakes up for one time slot, transmits and listens for transmissions from nearby devices, then turns its radio off and sleeps for a random number of time slots. We defined the following parameters:

- TIME_SLOT: the length of one time slot, in milliseconds
- SLEEP_CYCLE: the average number of time slots the device sleeps for

3.2. Determining duty cycle to reduce power consumption

The following subsection describes our choice of duty cycle, which determines the average number of time slots the device sleeps. Referring to our program parameters, duty cycle is given as TIME_SLOT/ ((1 + SLEEP_CYCLE)*TIME_SLOT).

As we discovered in Assignment 4, power consumption is greatly affected by duty cycle (since a low duty cycle ensures that our devices are awake for as little time as possible). We also wish to have a high probability of discovery within 30 seconds. Hence, similar to Task 1.3 of Assignment 4, we observed how long the packet intervals were between 2 communicating devices placed in close proximity. This time, we observed the average interval as well as the maximum interval. Each time we varied duty cycle, we observed the interval between packet reception over 6 minutes.

The results are shown in Table 1 below.

Time slot (ms)	Sleep cycle	Duty cycle	Avg interval (s)	Max interval (s)	# samples
100	9	10.00%	5.000	25.874	70
100	10	9.09%	5.069	24.812	73
100	11	8.33%	6.429	34.192	63
100	12	7.69%	8.348	39.002	46

Table 1: Average and maximum packet intervals between 2 devices placed 3m apart

From the experiment results, we determined that a duty cycle of 9.09%, or SLEEP_CYCLE = 10, was the smallest duty cycle (and therefore most power efficient duty cycle) such that the maximum observed interval over 6 minutes remained under 30 seconds.

We acknowledge the fact that the maximum interval varies quite a lot due to the inherent randomness of the algorithm, despite there being a clearly increasing trend in average packet intervals. Instead of choosing a higher duty cycle as a safety measure, we still decided on choosing a 9.09% duty cycle, or SLEEP_CYCLE = 10. This is because even at duty cycle = 8.33%, or SLEEP_CYCLE = 11, only one interval out of 63 data points was over 30 seconds. This translates to a 1.59% probability that the interval exceeds 30 seconds at duty cycle = 8.33%. From this, we can conclude that a 9.09% duty cycle has a probability greater than 98.41% of receiving packets within 30 seconds. We deemed the probability 98.41% sufficiently high, hence a duty cycle value of 9.09% was chosen.

The program we used to get the results in Table 1 was similar to our final program. This ensured that the time taken to run other instructions in each iteration was taken into account. We ran this experiment with a simpler code in the beginning, but after implementing the proximity logic we found that the packet intervals frequently exceeded 30 seconds. This unforeseen challenge is further discussed in section 6.1.

4. Neighbour proximity sensing

4.1. High-level algorithm

The high-level steps of our algorithm for proximity sensing is as follows:

1. Create an N x 4 array named nodes, where each row stores [node_id, first_received, latest_received, proximity_flag]
2. Initialise variables num_nodes = 0, nodes_in_proximity = 0
3. If a message from node_id is received with an RSSI above threshold:
    a. If node_id is not in array:
        i. Initialise nodes[num_nodes] = [node_id, curr_timestamp, curr_timestamp, 1]
        ii. Output “curr_timestamp DETECT node_id”
        iii. nodes_in_proximity++, num_nodes++
    b. If node ID is in array and proximity flag = 0:
        i. This node had previously been in proximity and moved out of range; get index i of node_id in nodes
        ii. nodes[i][1] = nodes[i][2] = curr_timestamp
        iii. Output “curr_timestamp DETECT node_id”
        iv. nodes_in_proximity++
    c. If node ID is in array and proximity flag = 1:
        i. nodes[i][2] = curr_timestamp
4. Before the device sends a message, check for nodes that have left proximity:
    a. Loop through every node in nodes array
    b. If curr_timestamp - latest_received > 30 seconds:
        i. Output “curr_timestamp LEAVE node_id”
        ii. Output “Time in proximity: latest_received - first_received”
        iii. nodes_in_proximity--

The following subsections will discuss why and how we stored node data, detected new and leaving nodes, and determined RSSI threshold values in further detail.

4.2. Storing node data

The node data is stored in a N x 4 array, where N refers to the number of nodes that can be detected by the devices. Each row stores the following values:

- node_id: the corresponding node ID of the detected node
- first_received: indicates the time at which a new node was detected in seconds
- latest_received: indicates the time of the most recent message received by a node in seconds
- proximity_flag: keeps track of whether a particular node is still within close proximity of the device

An example where data of 2 nodes are stored in the array is shown below:

node_id	first_received (s)	latest_received (s)	proximity_flag
7170	8	50	1
1268	10	23	1

Table 2: An example 2 x 4 array with stored data of 2 detected nodes

4.3. Checking if new nodes enter proximity

When a message is received with RSSI above threshold, the device checks if the node is already within proximity by checking proximity_flag of the corresponding node_id. If the node_id does not exist in the array, or the proximity_flag value is 0, the first_received and latest_received timestamps are updated to curr_timestamp. The device outputs “curr_timestamp DETECT node_id”.

The code for this can be found in the broadcast_recv function.

To illustrate this logic, let us take the example of a device with no detected nodes that receives one message from a node 1310 at curr_timestamp = 10. Since the node_id 1310 does not exist in the empty array, a row is added. The device also outputs “10 DETECT 1310”. The updated table will be as shown in Table 3.

node_id	first_received (s)	latest_received (s)	proximity_flag
1310	10	10	1

Table 3: An example 1 x 4 array with stored data of 1 newly detected node

4.4. Checking if already nearby nodes remain within proximity

When a message is received from a node with RSSI above threshold, the device checks if the node was already within proximity by checking the proximity_flag of the corresponding node_id. If the node_id exists in the array and its proximity flag is 1, the node was previously already in proximity. The device then updates the latest_received time for that node.

The code for this can be found in the broadcast_recv function.

A bigger challenge was to discover if a node has been out of proximity for more than 30 seconds. Since there is no explicit indicator when a node moves out of proximity, our algorithm loops through the nodes in the array each time it finishes one WAKE and SLEEP cycle. In this loop, the device checks if curr_timestamp - latest_received > 30 for any node_id. If this condition is satisfied for any node_id, we consider the node to have moved away from proximity. In this case, the device will output “curr_timestamp LEAVE node_id”, “Time in proximity: latest_received - first_received”, and then reset the values of first_received, latest_received and proximity_flag.

The code for this can be found in the sender_scheduler function.

To illustrate this logic, let us take the example if the device wakes up at curr_timestamp = 55 with reference to the example in Table 2. When it loops through the nodes array, node 7170 is considered still “in-proximity” since the elapsed time is 55 - 50 = 5 seconds. However, node 1268 is considered to have moved away from proximity since the latest received message from node 1268 was at 23 seconds, and the elapsed time = 55 - 23 = 32 > 30

seconds. The device will also output “55 LEAVE 1268”, “Time in proximity: 13”. The updated array will be as shown in Table 4.

node_id	first_received (s)	latest_received (s)	proximity_flag
7170	8	50	1
1268	0	0	0

Table 4: The updated 2 x 4 array after looping through the array

After the device has looped through the nodes array, it turns its radio on for one TIME_SLOT. Let us take the example that the device receives 2 messages during this TIME_SLOT: one from node with node_id 7170, one from node with node_id 1268. The device finds node 7170 already “in-proximity” as proximity_flag = 1, and only updates its latest_received value. It finds node 1268 in the array, but with proximity_flag = 0. Hence, it prints “55 DETECT 1268”, and updates the values for node 1268. The updated array will be as shown in Table 5.

node_id	first_received (s)	latest_received (s)	proximity_flag
7170	8	55	1
1268	55	55	1

Table 5: The updated 2 x 4 array after receiving 2 messages

4.5. Estimating RSSI threshold

To estimate an RSSI threshold, we averaged multiple measurements of RSSI for 2 devices placed 3m apart in random environments. However, we realised that this method of setting a threshold value was naive, especially since the accuracy of detection relied heavily on the environment the user was in.

To refine the threshold value, we decided to determine the threshold based on a single factor that we know will affect RSSI. Referring to Assignment 3, we know that apart from distance, obstacles, signal interference and orientation of the receiver and transmitter affects RSSI. We chose to focus on refining the RSSI threshold to take into consideration signal interference. This decision is discussed further in section 6.2.

Without a surefire way to accurately determine the signal interference in the environment, we decided to use the number of surrounding nodes as an indicator. Because the total number of nodes in proximity is updated at a maximum interval of 30 seconds, the device is able to accurately estimate the total number of nodes within close proximity, ignoring devices that are in proximity for only a brief window of time.

To choose appropriate locations to conduct experiments, we assumed that the amount of signal interference was directly proportional to the number of people with transmitting devices in the environment. We identified 3 types of environments with different amounts of signal interference:

1. an enclosed room with 1 other stationary node,
2. an open area with low human traffic and few other stationary nodes, and

3. an enclosed area with high human traffic and many other stationary nodes.

We assumed that if there are less than 5 detected nodes, the user is likely to be stationary in an enclosed environment like a room, with few other users surrounding him. For a scenario where there are more than 5 surrounding nodes detected, the user can either be stationary in an enclosed environment surrounding by others eg. office/classroom, or they could be travelling with a group of other users. Our last case covers the scenario where there are more than 10 nodes detected in proximity, which could indicate that the user is stationary in a crowded environment eg. restaurant/lecture.

We chose 3 locations that match the 3 environments described above, and recorded the average values of RSSI values for data transmission between 2 transmitting nodes. In all environments, we assumed that each person or node was running the TraceTogether application. Since ZigBee operates on the same ISM band as Bluetooth 4.0/5.0, there would be a proportional amount of signal interference recorded in our results. After collecting a set of 50 data samples for each location, we identified the average, minimum and maximum RSSI values as shown below.

Environment	Average RSSI	Min RSSI	Max RSSI
1	-60.8	-65	-50
2	-56.1	-67	-51
3	-61.0	-72	-54

Table 6: Results of RSSI threshold

To determine RSSI thresholds, we wanted to ensure our device accepts the lowest recorded RSSI value at 3 meters. Hence, we took the minimum recorded RSSI with an extra margin for error in order to prevent misinterpreting a node's weak signal as an indication that it had left the proximity of the device.

For the case of many surrounding nodes (more than 10) and high human traffic, similar to environment 1, we set an RSSI threshold of -75 so that we accept -72, which was the minimum RSSI observed. For the case of low human traffic and a few surrounding nodes (5 to 10), we chose a threshold of -70 based on the minimum RSSI observed in a environment 2. For an environment with very few nodes (0 to 5), we set an RSSI threshold of -65 based on the minimum RSSI observed in environment 1.

5. Results

For the first time a node is detected, the system prints out the corresponding timestamp and the ID of the detected node. When the node has left the proximity of the system for at least 30 seconds, the system prints out the timestamp at which it has detected this as well as the corresponding node ID. The system also estimates the total duration the two nodes were in proximity and prints it out. This is as shown below:

```
20 DETECT 8446
60 LEAVE 8446
Time in proximity: 9
```

The system can also detect more than 1 node in proximity, as shown below:

```
5 DETECT 7170
New node: 7170 || Nodes in proximity: 1
13 DETECT 8446
New node: 8446 || Nodes in proximity: 2
```

We conducted tests to evaluate our system's performance based on 2 factors:

1) Detection accuracy, using experimentation
2) Power consumption, using Cooja

5.1. Evaluation of system's detection accuracy

The system's detection accuracy refers to the ability of the system to detect a new node moving near, an existing node moving away and the time it takes to detect these events.

5 trials were conducted in each of the following scenarios:

1) Clear line-of-sight (LOS), no obstacle
2) Many wireless devices in between
3) Many physical obstacles in between
4) Inside and outside closed room, within proximity

Our findings are shown in the table below:

Scenario	Average time taken to detect arriving node (s)	Success Rate	Average time taken to detect leaving node (s)	Success Rate
Clear LOS, no obstacle	12	100%	22.2	100%
Wireless devices in between	11	100%	20.2	100%
Physical obstacles in between	11.8	100%	19.6	100%
Inside and outside closed room	9.5	40%	15	40%

Table 7: Performance of system under different scenarios

As shown in Table 7, the system was reliably able to detect the other node for the first 3 scenarios. However, for the last scenario, the system failed to detect the other node for distances greater than 1.5m when the other node was placed in a closed room and the system was outside of it. This illustrates the effect of having thick obstacles, which can heavily affect RSSI readings and cause issues in detecting the other node. We decided not to account for such obstacles in the RSSI threshold estimation, as discussed in section 6.2.

5.2. Evaluation of radio power consumption

During our code optimization phase, we use the powertrace logs retrieved from cooja to tweak our algorithm. We observed that as we iterate through a larger array of nodes to check if their most recently received message exceeds the 30 seconds timeout, our average CPU power consumption increased marginally. This is in comparison to iterating up to a variable, num_nodes, to keep track of how many active nodes need to be checked.

Approach	Average CPU consumption	Average LPM consumption
50 nodes	6065	6076
num_nodes (2)	157799	157810

Table 8: Power consumption comparison

As shown above, the difference in CPU and LPM energy consumption is negligible, which leads us to conclude that adding more nodes will only lead to a small increase in computation and CPU energy consumption. This is in contrast to using a higher radio duty cycle as we did in Assignment 4, which resulted in larger differences in energy consumption.

The full powertrace logs can be found attached.

6. Challenges faced

6.1. Duty cycle miscalculation

One mistake we made initially was to calculate the duty cycle without using the final algorithm within which we implemented extra logic. The original nbr_discovery.c code was used to measure the average and maximum packet intervals, and the results are as shown below:

Time slot (ms)	Sleep cycle	Duty cycle	Avg interval (s)	Max interval (s)	# samples
100	9	10.00%	3.240	6.000	10
100	12	7.69%	4.355	9.804	10
100	15	6.25%	8.221	21.203	10
100	16	5.88%	7.426	24.797	58
100	17	5.56%	7.830	32.890	39

Table 9: Average and maximum packet intervals between 2 devices placed 3m apart, using nbr_discovery.c

However, after implementing the proximity logic, we observed that packet intervals of over 30 seconds were more frequent than expected. We concluded that this was due to the computational overhead incurred when running our logic to check for “expired” nodes each time the device woke up. We carried out the experiment again using the final code that simulated having 50 nodes detected, to redetermine a duty cycle that minimises power

consumption while maintaining packet intervals of less than 30 seconds. The results for the second round of experiments are shown in section 3.2.

From this, we learnt that computation time for more complex algorithms is not negligible when compared to very short waking intervals (~100ms). Calculating an appropriate duty cycle to minimise power consumption should often be left to the later stages of a project, after the computation time is determined (when the algorithm has been written).

6.2. Detecting external factors

Another challenge faced was determining external factors that could affect the RSSI readings in the environment. From Assignment 3, we learnt that obstacles could affect the resulting RSSI values. Looking up the datasheet of the SensorTag, we found that it has 10 other sensors that include support for light, digital microphone, magnetic sensor, humidity, pressure, accelerometer, gyroscope, magnetometer, object temperature, and ambient temperature. From this list, we were unable to find appropriate sensors to detect surrounding obstacles as a measure of the environment.

Instead, we chose to focus on signal interference in the environment, another factor that affects RSSI as we proved in Assignment 3. To do this, we considered the number of surrounding nodes as an estimator of signal interference. This approach is discussed in further detail in section 4.5. Because the total number of nodes in proximity is updated at a maximum interval of 30 seconds, the device is able to estimate how many nodes are within a 3 meter radius. These nodes' signals are likely to interfere with each other, causing their RSSI values to fall.

6.3. Size of nodes array

With our current implementation, the number of detectable nodes is constrained by the 2D array size. Currently, we are not clearing the node_id of nodes that have left proximity due to the tradeoff between memory and computation time. We wanted to avoid looping through the whole array to check which nodes had not been heard from for over 30 seconds to reduce computation time, by keeping track of the total number of detected nodes with the variable num_nodes. To continue being able to do this, we would need to shift the rest of the array up by one row if we wanted to remove a node completely from the array. This process could take a significant amount of time, especially if there are many nodes in the array. If the user is on the move and nodes frequently leave proximity, this could even affect packet interval timings. However, our approach then constrains the number of detectable nodes a device can store.

To overcome this, we ensured the size of our array is appropriately large. Currently, the size of our array in our application is 50, but can be easily changed to a higher number.

Another method to overcome this challenge is to ensure the array is reset frequently. For example, once the user's device detects no nodes in proximity, we could reset the array completely as well as the counter num_nodes.

7. Conclusion

To summarise the lessons we learnt, this project has exposed us to the inner workings of, and the effort that goes into, an application close to our lives.

This application and TraceTogether run on different protocols - we implemented the application on ZigBee, while TraceTogether runs on Bluetooth. However, the inner workings of both applications must be similar in that it involves nodes frequently broadcasting messages and listening for surrounding nodes. To learn about how a government issued application such as TraceTogether works was an interesting project.

Other than learning about how TraceTogether works, we can now understand and appreciate the work that is needed to make it into an application suitable for islandwide rollout. For example, calibrating RSSI values is essential to the usefulness of the application since knowing the distance between users is crucial to tracing close contacts of confirmed COVID cases. It was also insightful to understand the importance of fine-tuning the duty cycle of the radio to achieve a practical design which aims to reduce power consumption without crippling functionality, in order for users to want to download the application. TraceTogether has to run smoothly in a phone's background to increase adoption rate; if power usage was too high, users would shy away from running the application.

8. Member contribution

Our group brainstormed the neighbour discovery protocol and proximity logic together, before beginning the implementation.

Junwei focused on the code implementation as well as power measurements using Cooja. He also helped with determining appropriate RSSI threshold values. His focus was more on implementing the code.

Shanon helped to debug the code, and carried out the experiments to determine an appropriate duty cycle. Her focus was on writing the report.

Davin took charge of testing the application and determining appropriate RSSI threshold values. His focus was more on refining and testing the code.

The group wrote the report together.