Project Report - Group 13

Chan Xu Hui
Poon Chuan An
R Ramana
Wamika Malik

Table of Contents

Overview

In this project, we implemented a neighbor discovery mechanism based on the “birthday protocol” between sensor tags. Subsequently, in the second part of the project, we enhanced this neighbor discovery mechanism to establish a delay-tolerant sensing application. The following sections provide a detailed description of these components.

Bonus

ENHANCEMENTS DONE - TASK 2

1. One enhancement done includes implementing receiver acknowledgment of light packets received. Once a receiver receives light packets, it sends out an ack to the sender. The sender/transmitter waits for the ACK for a certain timeout after sending the light packets. Since the light readings are updated and packets are sent every 3s, we have a timeout of 1s and wait for ack up to 3 times after sending a packet.
2. Another enhancement includes filtering out invalid packets that may be coming in from other nodes and have a different packet structure.

TASK 1

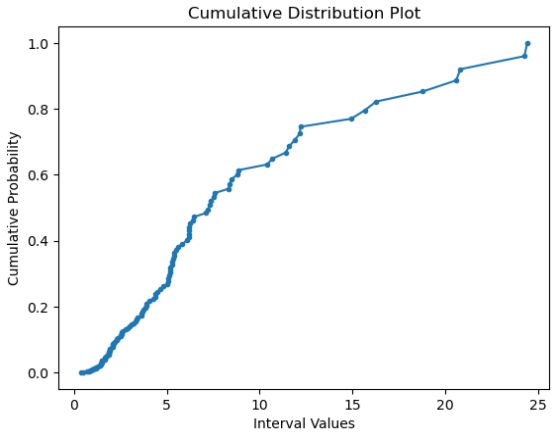
[1] Using the default settings, observe and record how long the devices take to discover each other. Pick one of the devices as A and plot the cumulative distribution of the intervals between packet receptions on device A hearing from device B.

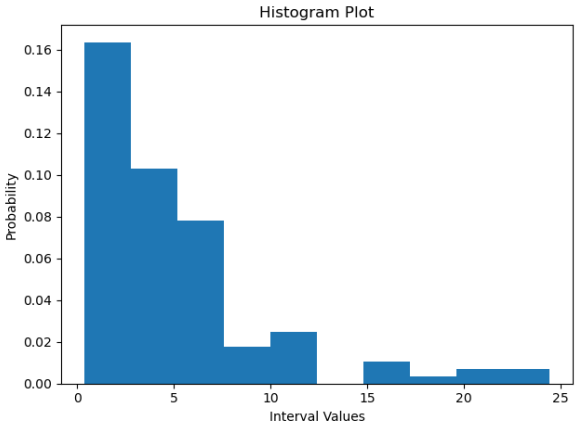
We ran the given code, *nbr.c*, for about 5 minutes and printed out the timings when device A hears from device B. Then, we calculated the intervals between the timings and plotted them on a cumulative distribution graph. Sometimes if the timeslots are aligned, two transmitted packets are received. To plot a more accurate distribution of discovery we removed the second packet that got transmitted in the same timeslot. This also presented a more realistic view of the mean and standard deviation (STD) of the dataset.

The default settings are:

- WAKE_TIME: 100ms (R_TIMER / 10)
- SLEEP_SLOT: 100ms (R_TIMER / 10)
- SLEEP_CYCLE: 9

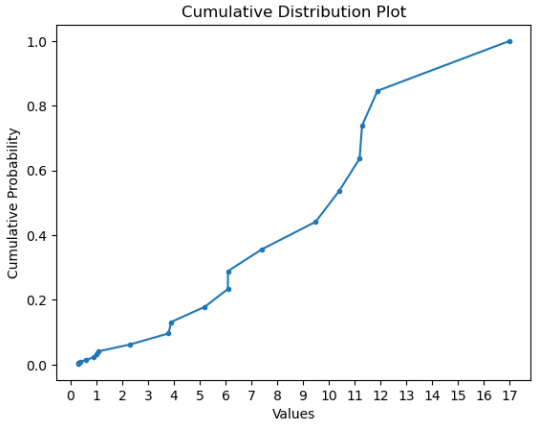
Duty Cycle = WAKE_TIME/(WAKE_TIME + SLEEP_CYCLE * SLEEP_SLOT) = 1/10 = 0.1 = 10%

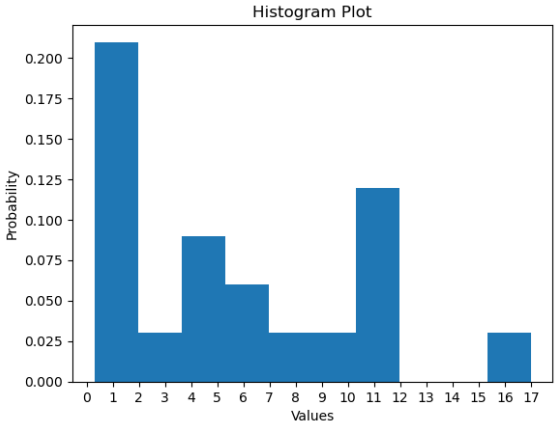




STD: 4.84s Mean: 5.24s

[2] Reset device B and observe how long it takes for device A to hear from device B after device B reboots. You may need to modify the given code to observe this duration. Perform the experiments at least 10 times and plot the cumulative distribution.



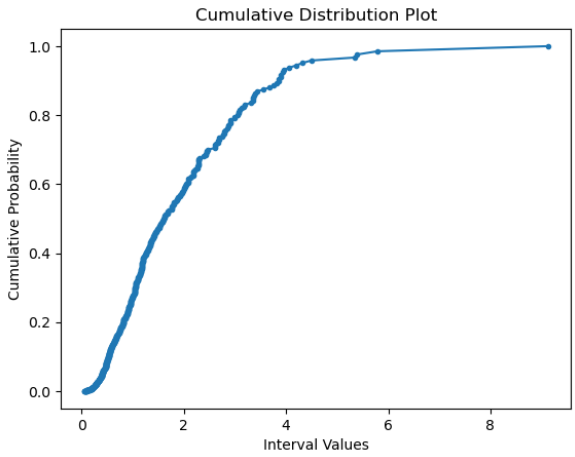


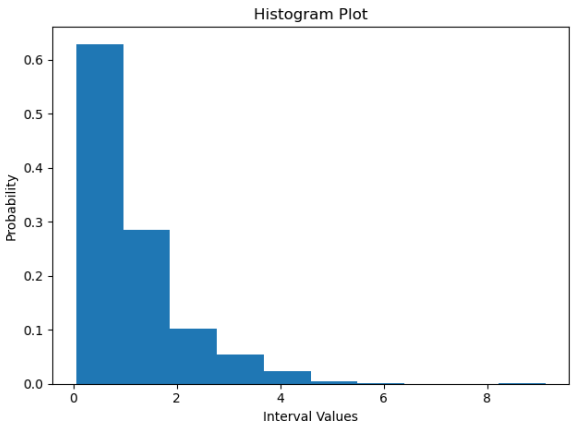
STD: 4.82s Mean: 5.53s

[3] Try out different settings and discuss your observations.

A]

WAKE_TIME = SLEEP_SLOT = RTIMER_SECOND / 20
SLEEP_CYCLE = 5
Duty Cycle = 1/(5+1) = 1/6 = 16.67%

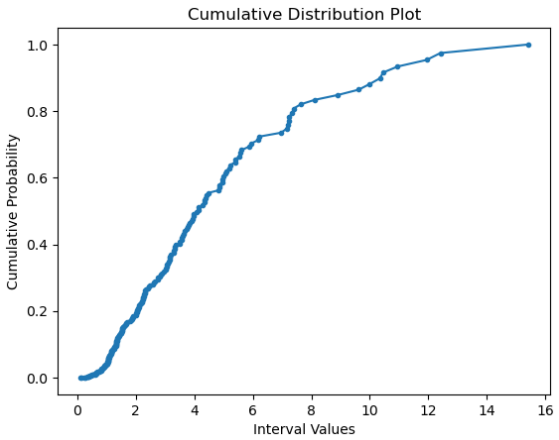


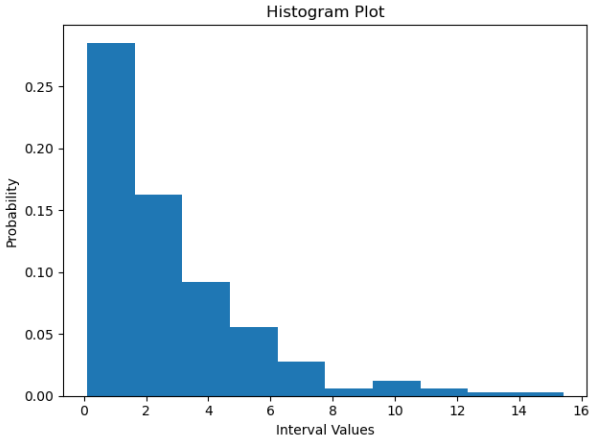


STD: 1.00s Mean: 1.11s

B]

WAKE_TIME = RTIMER_SECOND / 10
SLEEP_CYCLE = 9
SLEEP_SLOT = RTIMER_SECOND / 15
Duty Cycle = (1/10)/(1/10 + 9 * 1/15) = 1/7 = 0.1428 = 14.28%

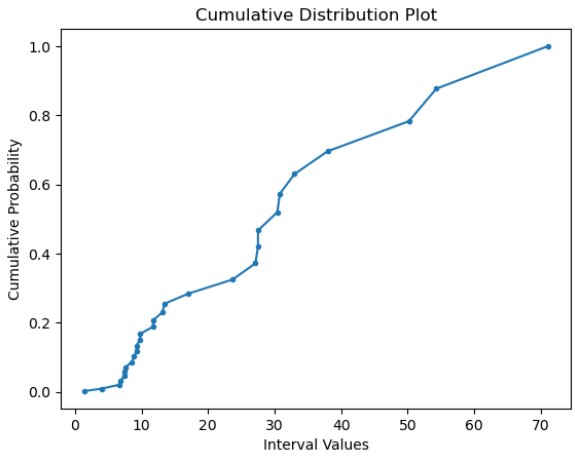


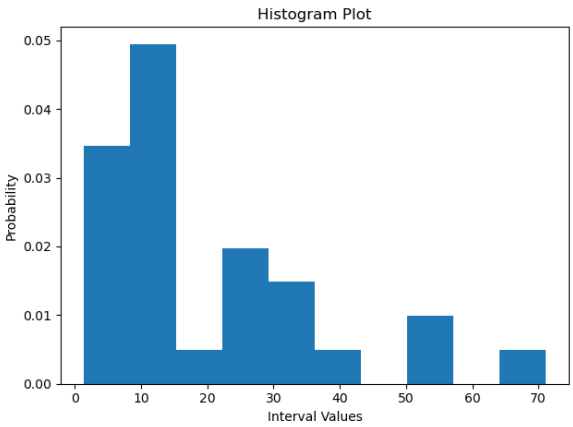


STD: 2.51s Mean: 2.83s

C]

WAKE_TIME = RTIMER_SECOND / 10
SLEEP_CYCLE = 9
SLEEP_SLOT = RTIMER_SECOND / 5
Duty Cycle = (1/10)/(1/10 + 9 * 1/5) = 1/19 = 0.0526 = 5.26%

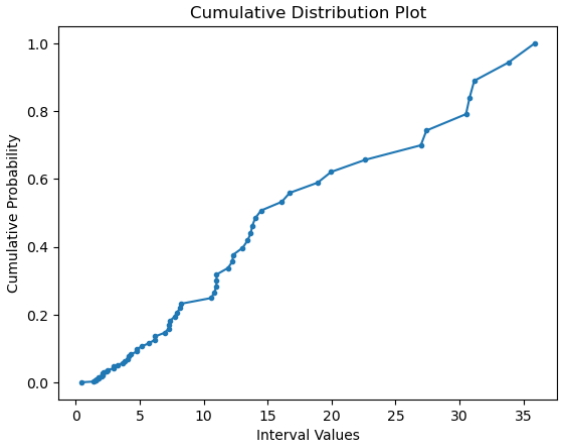


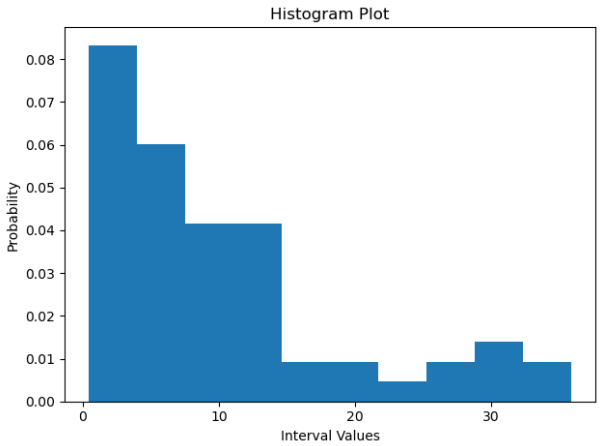


STD: 16.55s Mean: 19.91s

D]

WAKE_TIME = RTIMER_SECOND / 15
SLEEP_CYCLE = 15
SLEEP_SLOT = RTIMER_SECOND / 15
Duty Cycle = (1/15)/(1/15 + 15 * 1/15) = 1/16 = 0.0625 = 6.25%





STD: 9.07s Mean: 10.34s

Increasing the SLEEP_SLOT time by 2 fold (100ms to 200ms), while keeping the WAKE_TIME at a constant of 100ms and SLEEP_CYCLE at 9, shows an almost 4 fold increase in mean discovery time of the nodes (5.24s to 19.91s) with the standard deviation also being more drastic. Additionally, looking at the duty cycles, it seems to obtain accurate/timely detection. We want the duty cycle to be about 10%, if we want to minimize power consumption while maintaining a reasonably quick discovery time. Otherwise, in general, a higher duty cycle

will present with a quicker average discovery time and in general the converse holds true.

[4] Next, please modify the program (nbr.c) so that two-way discovery (A hears from B AND B hears from A) can be completed in a deterministic manner within 10 seconds. You should choose settings so that the radio power consumption is "minimized".

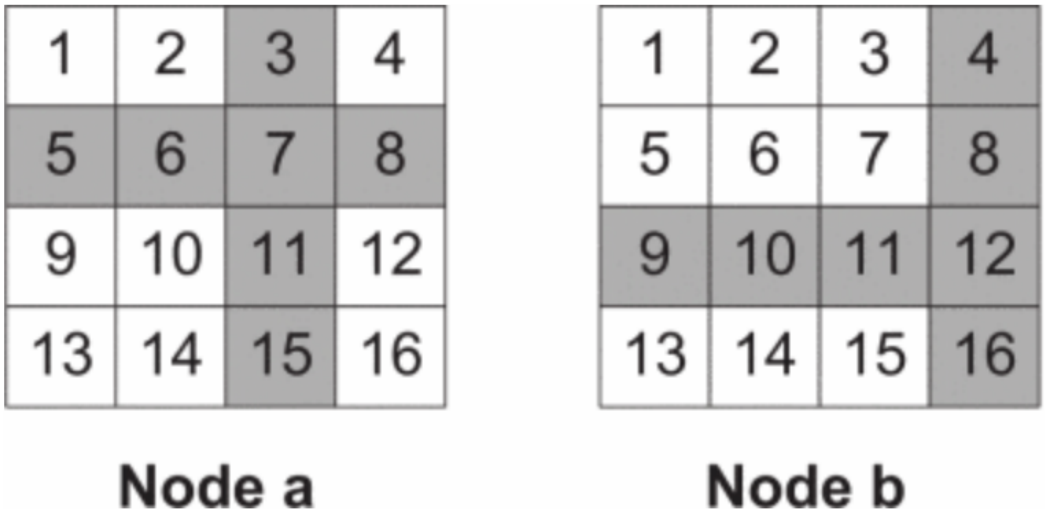
You must include the following in the report (details instruction below in the document): a) the algorithm you have implemented b) the parameters chosen c) the maximum two-way latency observed

a) Chosen Algorithm: Quorum

Implementation:

We used a quorum algorithm which divides the timeslots into a large grid. Nodes will at (pseudo-)random, decide a row and a column to be awake. Each grid is TIME_SLOT duration long.

Referring to the image below, adapted from the paper Self-Adapting Quorum-Based Neighbor Discovery in Wireless Sensor Networks [1], you can see how the quorum algorithm works for a grid size of 4x4.



Essentially by using random_rand() % GRID_LENGTH, where GRID_LENGTH = 4 in the above example, each node will choose a row and column at random. In the example above, node a selected row 1 and column 2 (0-index), while node b selected row 2 and column 3. The nodes will be awake whenever the time_slot reaches the greyed squares. Both nodes will be awake at squares 8 and 11 in the example above. This is deterministic as in the worst case there will always be 2-time slots (squares) in which both nodes will be awake. To ensure a deterministic discovery under 10s,

$TIME\_SLOT * GRID\_LENGTH^2 <= 10 * RTIMER\_SECOND$

To ensure the lowest power consumption, we need higher GRID_LENGTH and lower TIME_SLOT to ensure a lower duty cycle. At the same time, $TIME\_SLOT * GRID\_LENGTH^2$ must be as close to 10s as possible. The parameters chosen can be found in part b.

Algorithm steps:

1. Set row and column to (pseudo-)random values between 0 and GRID_LENGTH - 1
2. Set two variables, curr_row and curr_col to 0
3. If curr_row equals row or curr_col equals col GOTO step 4. Else GOTO step 7.
4. Wake up node and send a packet.
5. Wait for SLOT_TIME.
6. Send a packet.
7. Sleep for SLOT_TIME.
8. If curr_col equals GRID_LENGTH - 1 and curr_row equals GRID_LENGTH - 1 SET curr_col = 0 and curr_row = 0.
9. Else If curr_col equals GRID_LENGTH - 1 SET curr_col = 0 and curr_row = curr_row + 1.
10. Else SET curr_col = curr_col + 1.
11. GOTO step 3.

Other Attempts:

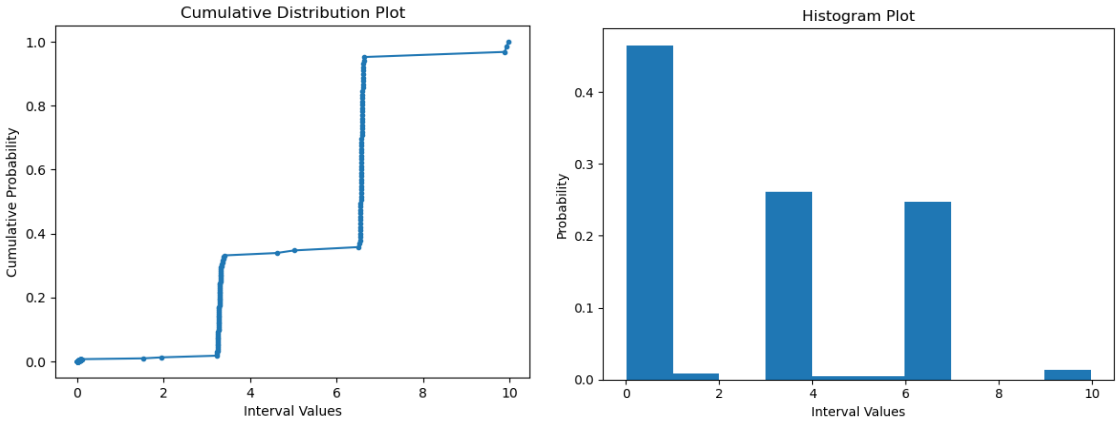
We decided to do a quorum discovery algorithm after trialing with disco and being unable to achieve sending under a deterministic time. While we selected p_i and p_j, where p_i != p_j, and set sleep_time for each node to be p_i -1 and p_j - 1 respectively, and where $TIME\_SLOT * p_i * p_j < 10s$, we occasionally got intervals of ~10.4s. Having a $TIME\_SLOT * p_i * p_j < 8s$ or 9s works but increases the power consumption as the nodes tend to wake up more frequently. As such, we decided to try out other deterministic discovery algorithms that might yield a better outcome, and settled on the quorum discovery protocol.

b) Parameters:

Best Parameters chosen: SLOT_TIME = RTIMER_SECOND/17 = **588.2ms**; and GRID_LENGTH = **13**

Duty Cycle = $(13+12)/13^2$ = 25/169 = 0.1479 or **14.79%**

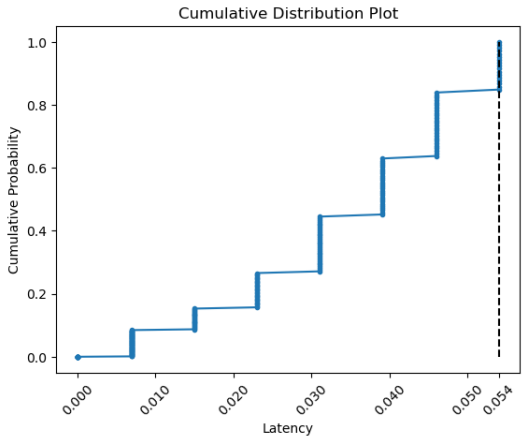
a) As the slot size reduces and grid length increases, the sensor tags can discover each other in < 10s for shorter lengths of time. After a certain point the two devices completely stop discovering each other. This is likely caused due to drift.
b) We start testing with values of 100 ms slot time and a grid length of 9 and then use different combinations by progressively decreasing the slot length to lower power consumption.
c) We try out different combinations to come up with the most optimum value that balances between energy consumption and prevents drift for the maximum time possible.
d) Best combination for deterministic discovery in < 10s: SLOT_TIME = RTIMER_SECOND/17 and GRID_LENGTH = 13
e) We ran the algorithm on 2 nodes for 10s and noted down the time of discovery in each case. Maximum discovery time noted: **9.969**s



STD: 2.44s Mean: 3.33s

c) Maximum Two-Way Latency:

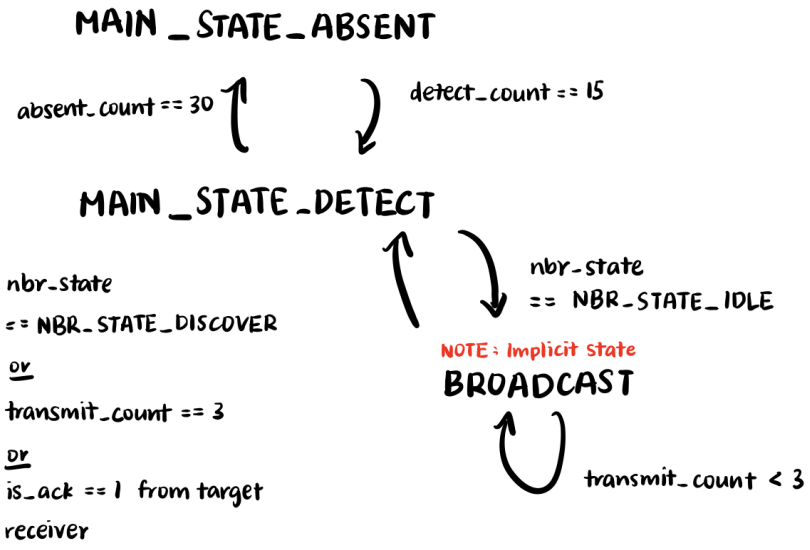
Maximum two way latency or two way discovery time observed for a wake up time of 1/17 = 0.059s is **0.054s**.



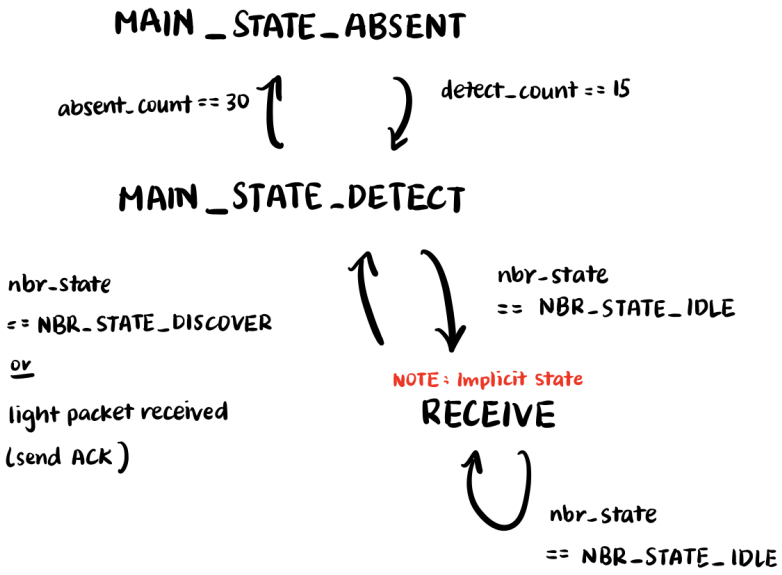
TASK 2

IMPLEMENTATION LOGIC

Transmitter



Receiver



Our SensorTags runs as two state machines, namely main_state, which manages the overall state of the SensorTag, and nbr_state, representing the state of the neighbor discovery process.

Both transmitter and receiver are similar in their states, with the exception of the transmitter having an implicit BROADCAST state while the receiver has an implicit RECEIVE state.

PROXIMITY DETECTION

Proximity detection is done using the RSSI value that is detected during the neighbor discovery process. For our case, after some experimentation, we got a range between -75 to -50 RSSI value for a distance of 3m. However, we found that an RSSI of >= -55 dBm (i.e., values like -50, -47, etc) best represents distances <= 3m and the drastic difference in value (-75) was an anomaly. Thus we use this value as our threshold for DETECT and ABSENT.

During the discovery process, the transmitter will always prioritize the first device it finds for the logging of DETECT and ABSENT events. Transmitters will not be able to detect other transmitters, while receivers will not be able to detect other receivers. This is done to prevent confusion in our discovery process.

RELIABILITY

We implemented an acknowledgement protocol in our system. With every broadcast of a light packet, the transmitter will wait for the receiver's acknowledgement for 1 second. If no acknowledgement is received by the transmitter, the light packet will be transmitted again, provided that the transmitter is not in NBR_STATE_DISCOVERY, until an acknowledgement is received or three broadcast attempts have been made, whichever earlier.

POWER CONSUMPTION

With the implementation of the acknowledgement protocol, for the transmitter, we are able to turn off the radio immediately when an acknowledgement is received, reducing any unnecessary broadcasts. As for the receiver, we are able to turn off the radio immediately upon receiving the light packet as well.

NEIGHBOUR DISCOVERY

1. Neighbour Discovery Algorithm used: **Quorum**
2. Parameters:
    a. SLOT_TIME = RTIMER_SECOND/**10**
    b. GRID_LENGTH = **12**
3. Reasons for choosing quorum: This protocol allows deterministic discovery and we need to ensure that the nodes discover each other within a given time slot. In our case we chose our algorithm to have a maximum discovery time of 14s. This is the value that gives us the best results while minimizing duty cycle. Since the nodes need to be discoverable within 3m, 15s after the first within 3m discovery, it makes sense to have a threshold of 14s for our neighbor discovery as this would ensure that the node is discovered at least once again within 15s after the first discovery.

4. Duty Cycle = $(12+11)/12^2$ = 23/144 = 0.1597 or **15.97%**

While higher grid length values (implying lower duty cycle) are possible by reducing the slot length, we chose not to do so since smaller slot lengths result in increasing drifts. In our experiments, a duty cycle smaller than our current implementation of 15.97%, results in unreliable matching of time slots after about 2-3 minutes. Hence even though reducing power consumption is a requirement, we wanted to ensure a decent enough reliability and robustness of our design.

SOME RESULTS

A receiver detects the transmitter within 3m at 2s and prints DETECT approximately 15s later as seen in the red box below. The light packet is received approximately 16s after discovery as seen in the green box below.

```
Send seq# 72  @     2188 ticks    17.093
Transmission Time:   0.000
2 DETECT 21632
Send seq# 73  @     2316 ticks    18.093
Transmission Time:   0.000
Received neighbour discovery packet 56 with rssi -46 from 21632 at  18.101
Neighbour's time of sending  20.500

Send seq# 74  @     2328 ticks    18.187
Transmission Time:   0.007
Received neighbour light packet from 21632 at  18.187, Light={10.17, 9.53, 10.74, 10.24, 10.24, 10.50}

Send seq# 75  @     2469 ticks    19.289
```

The receiver receives light packets whenever there is a change in light readings approximately every 3s as seen below.

```
Transmission Time:   0.007
Received neighbour light packet from 21632 at  67.382, Light={10.98, 11.04, 11.04, 11.04, 11.20, 11.20, 10.34, 10.66, 11.06, 11.52}

Send seq# 223  @     8766 ticks    68.484
Transmission Time:   0.007
Send seq# 224  @     8779 ticks    68.585
Transmission Time:   0.007
Send seq# 225  @     8920 ticks    69.687
Transmission Time:   0.000
Send seq# 226  @     8933 ticks    69.789
Transmission Time:   0.000
Send seq# 227  @     9073 ticks    70.882
Transmission Time:   0.007
Send seq# 228  @     9086 ticks    70.984
Transmission Time:   0.007
Received neighbour light packet from 21632 at  70.984, Light={10.98, 11.04, 10.50, 11.04, 11.20, 11.20, 10.34, 10.66, 11.06, 11.52}
```

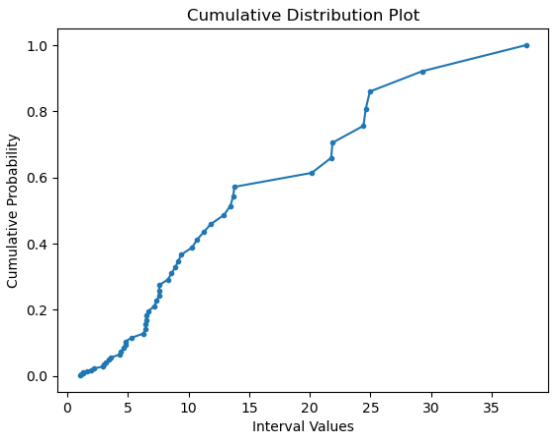
Once the receiver leaves the 3m range, it prints ABSENT when it stops detecting the transmitter within 3m for 30s as seen below.

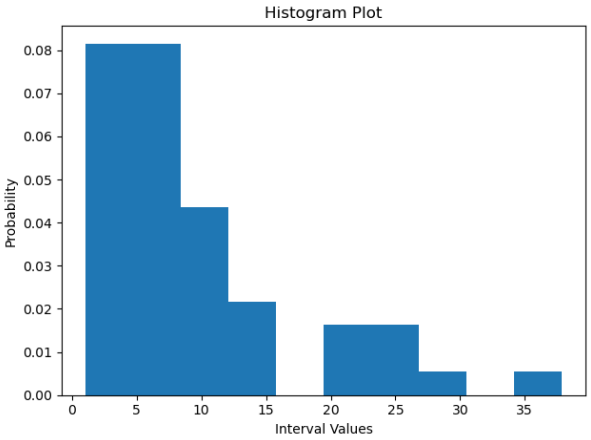
```
Send seq# 316  @    12465 ticks    97.382
Transmission Time:   0.007
68 ABSENT 21632
Send seq# 317  @    12606 ticks    98.484
Transmission Time:   0.007
```

Appendix

Other Parameters tested for Task 1, Part 3

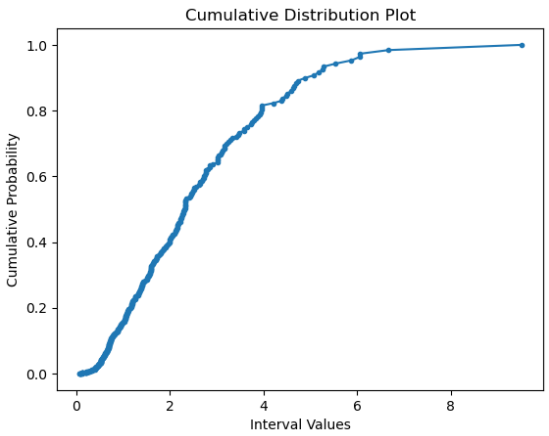
WAKE_TIME = RTIMER_SECOND / 20
SLEEP_CYCLE = 4
SLEEP_SLOT = RTIMER_SECOND / 5
Duty Cycle = (1/20)/(1/20 + 4 * 1/5) = 1/17 = 0.0588 = 5.88%

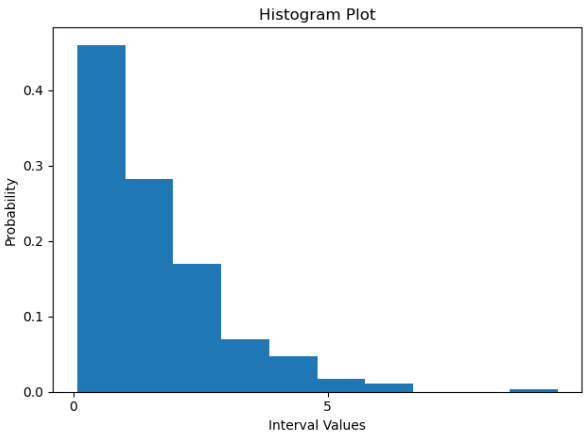




STD: 8.05s Mean: 9.57s

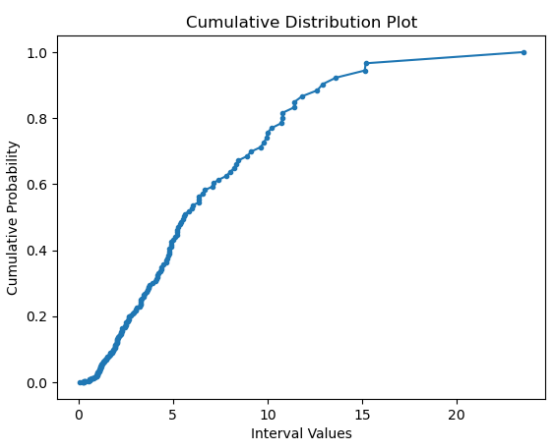
WAKE_TIME = RTIMER_SECOND / 15
SLEEP_CYCLE = 5
SLEEP_SLOT = RTIMER_SECOND / 15
Duty Cycle = (1/15)/(1/15 + 5 * 1/15) = 1/6 = 0.1667 = 16.67%

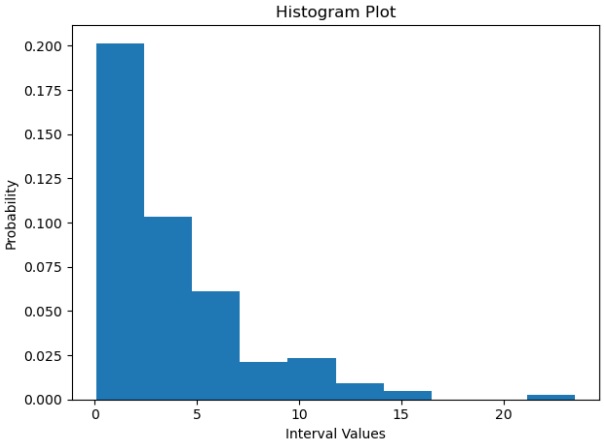




STD: 1.32s Mean: 1.58s

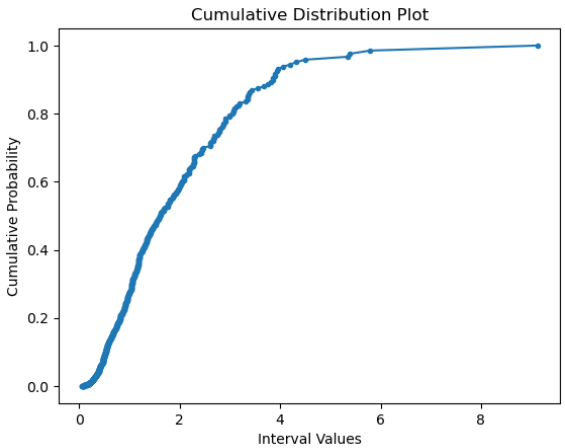
```
WAKE_TIME = RTIMER_SECOND / 15
SLEEP_CYCLE = 9
SLEEP_SLOT = RTIMER_SECOND / 15
Duty Cycle = (1/15)/(1/15 + 9 * 1/15) = 1/10 = 0.1 = 10%
```

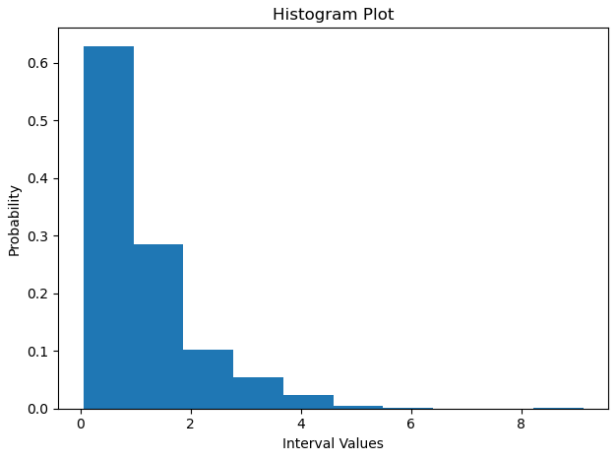




STD: 3.54s Mean: 3.81s

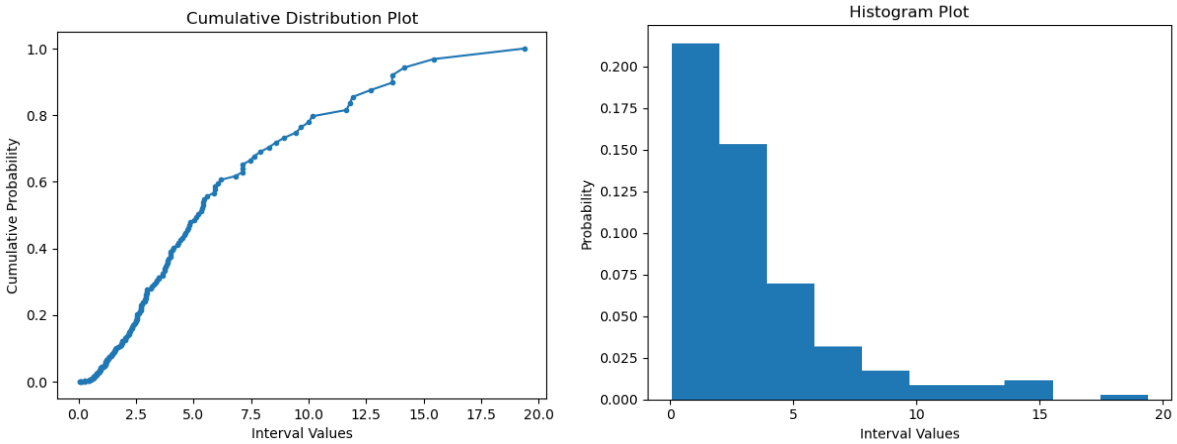
```
WAKE_TIME = RTIMER_SECOND / 20
SLEEP_CYCLE = 5
SLEEP_SLOT = RTIMER_SECOND / 20
Duty Cycle = (1/20)/(1/15 + 5 * 1/20) = 1/6 = 0.1667 = 16.667%
```





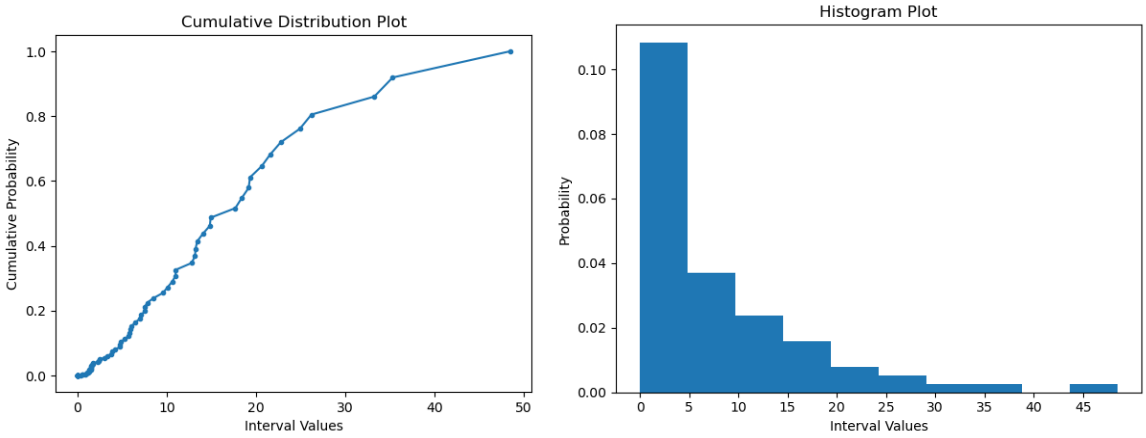
STD: 1.00s Mean: 1.11s

```
WAKE_TIME = RTIMER_SECOND / 20
SLEEP_CYCLE = 9
SLEEP_SLOT = RTIMER_SECOND / 20
Duty Cycle = (1/20)/(1/20 + 9 * 1/20) = 1/10 = 0.1 = 10%
```



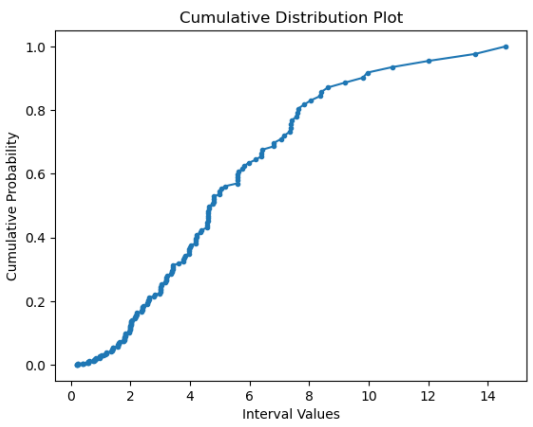
STD: 3.30s Mean: 3.42s

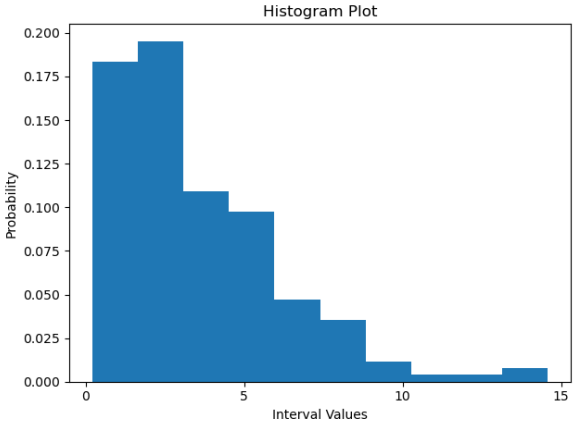
```
WAKE_TIME = RTIMER_SECOND / 20
SLEEP_CYCLE = 15
SLEEP_SLOT = RTIMER_SECOND / 20
Duty Cycle = (1/20)/(1/20 + 15 * 1/20) = 1/16 = 0.0625 = 6.25%
```



STD: 9.36s Mean: 7.69s

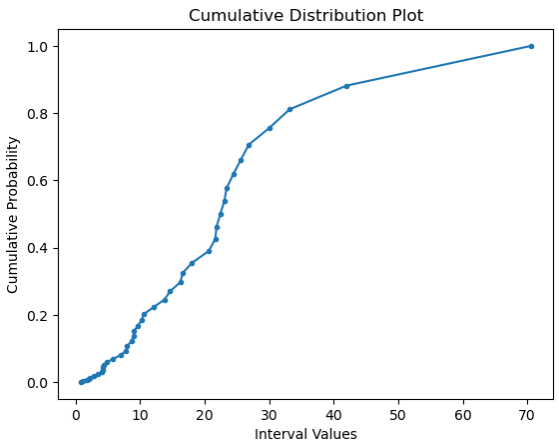
```
WAKE_TIME = RTIMER_SECOND / 5
SLEEP_CYCLE = 5
SLEEP_SLOT = RTIMER_SECOND / 5
Duty Cycle = (1/5)/(1/5 + 5 * 1/5) = 1/6 = 0.1667 = 16.67%
```

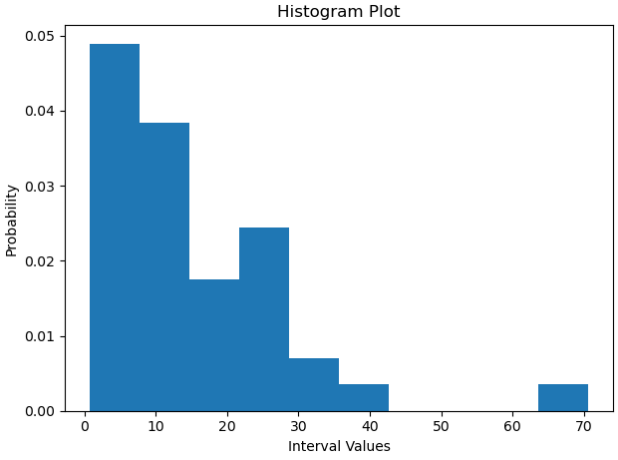




STD: 2.65s Mean: 3.49s

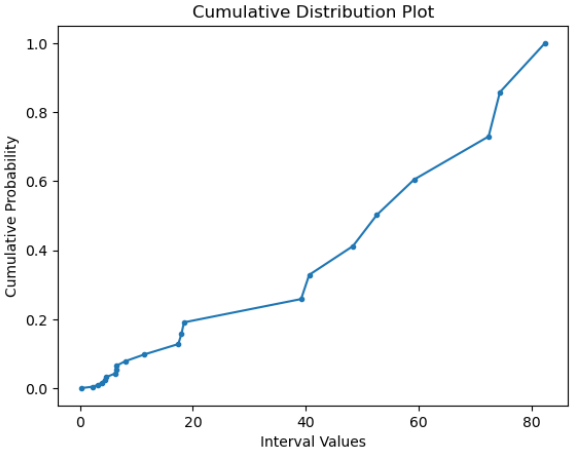
```
WAKE_TIME = RTIMER_SECOND / 5
SLEEP_CYCLE = 9
SLEEP_SLOT = RTIMER_SECOND / 5
Duty Cycle = (1/5)/(1/5 + 9 * 1/5) = 1/10 = 0.1 = 10%
```

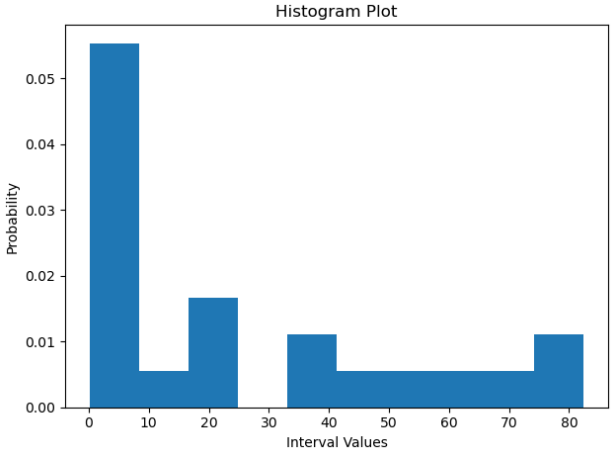




STD: 13.28s Mean: 14.59s

```
WAKE_TIME = RTIMER_SECOND / 5
SLEEP_CYCLE = 15
SLEEP_SLOT = RTIMER_SECOND / 5
Duty Cycle = (1/5)/(1/5 + 15 * 1/5) = 1/16 = 0.0625 = 6.25%
```





STD: 26.49s Mean: 26.35s

References

[1] H. Cai and T. Wolf, "Self-Adapting Quorum-Based Neighbor Discovery in Wireless Sensor Networks," IEEE INFOCOM 2018 - IEEE Conference on Computer Communications, Honolulu, HI, USA, 2018, pp. 324-332, doi: 10.1109/INFOCOM.2018.8486268.