Group 6 - Assignment 4

1. Group Members

Davindran	A0167009E
Low Jun Wei	A0168303J
Shanon Seet	A0172753B

2. Introduction

This report explores how different duty cycles and length of wake timing affects efficiency and power consumption of neighbour discovery. It is carried out in 2 environments: using 2 CC2540 SensorTag devices, as well as using the Cooja wireless sensor networks simulator which allows us to perform power measurements of the code.

In this report, the wake time and sleep time refers to the length of the wake and sleep intervals of the device respectively. Sleep time is calculated from (sleep slot * sleep cycle). All measurements are written in the units of milliseconds (ms). In all experiments carried out on a pair of SensorTags, the SensorTags were placed side by side to avoid packet losses, which would affect the data collection.

3. Task 1

3.1. Task 1.1

In this task, we observed and recorded how long each device took to discover each other using the default settings of WAKE_TIME = 100ms, SLEEP_SLOT = 100ms and SLEEP_CYCLE = 9.

We recorded the intervals of 20 packets received by one device from the other. Using the Excel NormDist function, we plotted a cumulative distribution of the cumulative probability against the intervals of packet reception. The resulting graph is shown below.

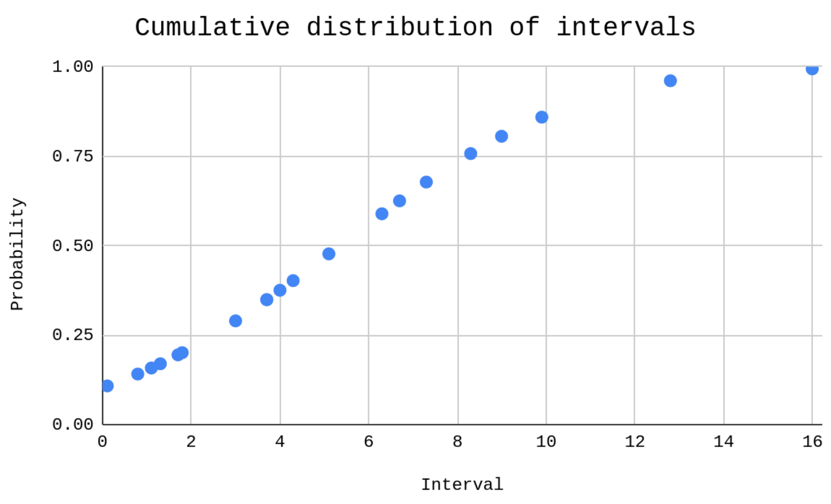

Fig. 1. Graph of cumulative distribution of intervals, where WAKE_TIME = 100ms, SLEEP_SLOT = 100ms, SLEEP_CYCLE = 9.

3.2. Task 1.2

In this task we recorded how long it took for device A to discover device B after device B reboots, using the default settings of WAKE_TIME = 100ms, SLEEP_SLOT = 100ms and SLEEP_CYCLE = 9.

We recorded the intervals of 20 packets received by one device from the other. The cumulative distribution is shown below.

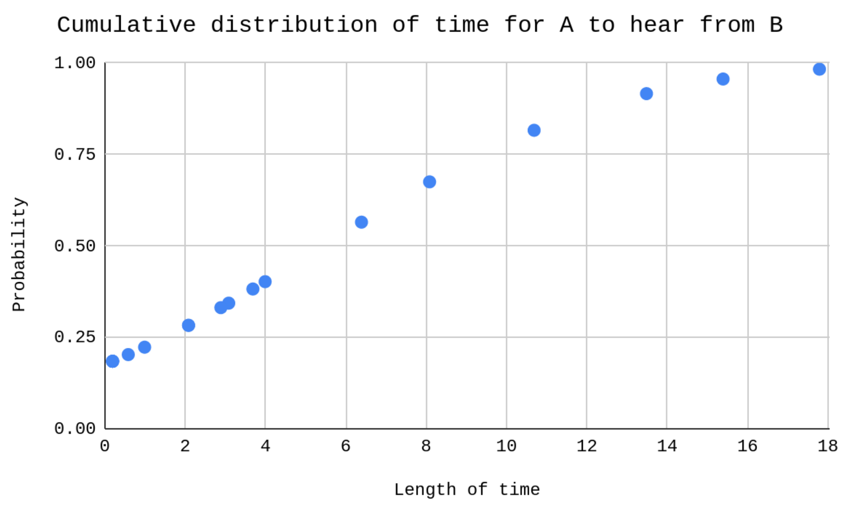

Fig. 2. Graph of cumulative distribution of the discover interval, where WAKE_TIME = 100ms, SLEEP_SLOT = 100ms, SLEEP_CYCLE = 9.

3.3. Task 1.3

3.3.1. Experiment description

We conducted 2 sets of experiments: one where we varied settings on both devices, and one where we varied settings on only the sending device (device B).

In each experiment, we varied 2 things: firstly we varied duty cycle by varying sleep cycle, and secondly we varied wake time and sleep slot equally, with a fixed duty cycle of 10%. For each record, we took at least 6 readings and recorded the average. The parameters used and results are recorded in Tables 1.1 and 1.2 below.

3.3.2. Experiment results

Wake time (ms)	Sleep slot (ms)	Sleep cycle	Duty cycle	Interval (s)
100	100	6	14.29%	3.100
100	100	9	10.00%	5.344
100	100	12	7.69%	6.919
100	100	15	6.25%	13.359

100	100	18	5.26%	14.339
120	120	9	10.00%	3.413
100	100	9	10.00%	5.344
80	80	9	10.00%	2.526
60	60	9	10.00%	2.205
40	40	9	10.00%	2.416

Table 1.1. Parameters for both A and B, and the corresponding measured interval

Wake time (ms)	Sleep slot (ms)	Sleep cycle	Duty cycle	Interval (s)
100	100	6	14.29%	2.950
100	100	9	10.00%	5.344
100	100	12	7.69%	10.632
100	100	15	6.25%	5.020
100	100	18	5.26%	10.278
120	120	9	10.00%	4.868
80	80	9	10.00%	6.466
60	60	9	10.00%	2.683
40	40	9	10.00%	2.074

Table 1.2. Parameters for only B, with A fixed at default settings, and the corresponding measured interval

3.3.3. Discussion when varying both devices' parameters

From Table 1.1, we can make two observations: firstly, as duty cycle decreased, the interval between packet receptions increased. Secondly, as wake time and sleep time decreased while duty cycle remained constant, the intervals decreased.

For asynchronous discovery where two nodes randomly wake up k time slots out of a total n slots, the probability of them waking up in the same time slot is given by $Q = 1 - \frac{C(n-k,k)}{C(n,k)}$. Q can be calculated using different n and k values, and the following calculations are taken from *Lecture 5 Pg 38*:

- n = 100, k = 1, duty cycle = 0.01, Q(100,1) = 0.01
- n = 100, k = 5, duty cycle = 0.05, Q(100,5) = 0.23
- n = 1000, k = 10, duty cycle = 0.01, Q(1000,10) = 0.096
- n = 1000, k = 50, duty cycle = 0.05, Q(1000, 50) = 0.928

From the above, it can be seen that when the duty cycle increases, Q increases, and therefore the interval between packet receptions would decrease. This corresponds with our experiment data, where we varied duty cycle by varying sleep cycle.

It can also be seen that when n and k increases by the same factor such that duty cycle remains the same, Q increases. This corresponds with our experiment data as well, where we varied wake time and sleep slot equally while keeping duty cycle constant. However, we observed that when wake time and sleep slot were decreased to 40ms, the interval increased from when they were at 60ms. This could be due to the inherent randomness of the algorithm, with extremely long time interval outliers skewing the average reading. A more accurate measurement would require observing and averaging the time intervals between packet receptions over a long period of time.

3.3.4. Discussion when varying one device's parameters

From Table 1.2, we observed that the behaviour of packet interactions when the two devices have different duty cycles was a lot more unpredictable than when the devices have the same parameters. When the duty cycle decreased on only one device, the interval between time receptions fluctuated greatly. Although this could be due to the randomness of the algorithm, we increased the number of readings and yet could find no observable trend.

It can also be seen that when the duty cycles of the two devices were the same, the same decreasing trend in intervals could be observed. It is also worth noting that even when only one device varied their wake and sleep times, the readings were very close to the readings of that in Table 1.1. From the perspective of the transmitter, the number of slots that the receiver is awake (k) and the total number of slots to choose from (n) has also increased, regardless of whether the receiver's wake up time and sleep slot are actually changed.

4. Task 2

The powertrace log is included as a separate file, cooja_powertrace.txt.

5. Task 3

5.1. Algorithm - quorum-based protocol

To achieve a deterministic bound approach to two-way discovery, we chose to implement the quorum-based protocol described in *Lecture 5 Pg 39*.

In this algorithm, time slots can be modelled as a NxN matrix. Every N^2 slots, each device randomly chooses a row and column in the range [0, N), and wakes up for every slot in that row and that column. This ensures that each device wakes up exactly (2N - 1) times from the N^2 slots. It also ensures there are at least 2 points of intersection in time slots where both devices are awake and ready for two-way discovery. For this task, we modified the code such that the wake and sleep time slots are of the same duration T.

The algorithm is seeded with node ID in order to ensure the two devices generate different pseudo numbers, when simulated in Contiki. Devices randomly select a row and column every N*N intervals, and then for each of the N*N time slots, checks whether its index falls within the row or column. Thus this determines when the devices wake up.

The algorithm is attached in Appendix A.

5.2. Worst case discovery time

We determined that in this algorithm, two devices will take N^2 - N time slots before their chosen slots first intersect in the worst case scenario. An example of the worst case scenario in a 4x4 matrix is illustrated in Fig. 3 below:

Device 1's 4x4 matrix			
0	1	2	3
4	5	6	7
8	9	10	11
12	**13**	**14**	**15**

Device 2's 4x4 matrix			
0	1	2	3
4	5	6	7
8	9	10	11
12	**13**	**14**	**15**

Device 1’s chosen slots	**0**	1	2	3	**4**	5	6	7	**8**	9	10	11	**12**	**13**	**14**	**15**
Device 2’s chosen slots	1	2	3	**4**	5	6	7	**8**	9	10	11	**12**	**13**	**14**	**15**	**0**

Fig. 3. An illustration of the worst case scenario in a quorum-based protocol, for N = 4, when both devices randomly chose col = 1 and row = 3.

Therefore, the worst case discovery time for 2 devices is N^2 - N time slots. Let T = length of a time slot, the worst case discovery time = (N^2 - N)*T. We would need to find an N and T value such that the worst case discovery time stays within the 10 second maximum latency requirement.

N and T are therefore related by the following equation:

$$(N^2 - N) * T \leq 10 \Rightarrow N \leq \frac{1 + \sqrt{1 + 40/T}}{2}$$

5.3. Low duty cycle for power consumption

To adhere to the requirements of minimising power consumption, we concluded that it would require a minimal duty cycle. A low duty cycle ensures that our devices are awake for as little time as possible, minimising power consumption.

In this implementation of discovery, duty cycle = $\frac{2N-1}{N^2}$. As N increases, duty cycle decreases since N^2 increases at a faster rate than 2N - 1. Hence, we need to find the largest value of N such that the worst-case maximum latency was less than, but as close to 10 seconds as possible.

$$N = floor(\frac{1 + \sqrt{1 + 40/T}}{2})$$

5.4. Finding optimal values of N and T

We varied T from 25 ms to 100 ms in 25 ms increments, along with the corresponding calculated values of N, rounded down to the nearest whole number.

Table 2 below shows the radio energy and time consumption after 10 minutes of simulation in Cooja for different sets of parameters, along with the respective duty cycles. We took energy consumption as directly related to power consumption.

Radio % of energy consumption refers to the statistic (accumulated transmission energy consumption + accumulated listen energy consumption) / (accumulated CPU energy consumption + accumulated low power mode energy consumption). Radio cumulative refers to the percentage of time spent on the transmitting and listening states of the radio.

T (ms)	Rounded N	Duty cycle	Max latency observed (s)	Radio % of energy consumption	Radio cumulative
50	14	13.78%	9.750	15.91%	15.143%
60	13	14.79%	9.812	15.14%	21.06%
70	12	16.00%	9.921	16.14%	19.03%
75	12	16.00%	9.8430	17.45%	17.166%
100	10	19.00%	7.398	20.27%	20.190%

Table 2. Parameters and readings for cumulative percentage of time spent on radio over 10 simulated mins

As can be seen from Table 2, when duty cycle increases, the percentage of radio energy consumption over total energy consumption increases as expected, and corresponds to the amount of time spent in the radio states. The maximum latency of packet receptions for all intervals were also within 10 seconds.

While running the simulation, at T = 50 ms and lower, packets were no longer being received after a short period of time, despite the devices being awake at similar time slots. We believe that this is because as T decreases, the amount of time it takes to send a packet becomes more significant, leaving less awake time for packet discovery. Another possible reason could be because as T decreases, the time window for the nodes to receive becomes shorter, and this could result in insufficient time to complete packet reception, leading to packet drops. A small T could also be detrimental to radio power consumption, as the radio has to frequently transition states between awake and asleep, and this could also result in time overheads. Hence, only T larger than 50 ms was considered.

By changing the parameters in Cooja, we determined that the smallest T such that packets were being reliably received after 10 minutes, and where maximum latency was kept below 10 seconds was T = 60 ms. When T = 60 ms, N = 13, Duty cycle = 14.79%, and the overall radio duty cycle was 21.06%.

Appendix A - Quorum-based Protocol Algorithm

```
char sender_scheduler(struct rtimer *t, void *ptr) {
  PT_BEGIN(&pt);

  rand_row = random_rand() % N_size;
  rand_col = random_rand() % N_size;
  curr_timestamp = clock_time();
  start_timestamp = clock_time();

  printf("Start clock %lu ticks, timestamp %3lu.%03lu\n", curr_timestamp, curr_timestamp / CLOCK_SECOND,
((curr_timestamp % CLOCK_SECOND)*1000) / CLOCK_SECOND);

  while (1) {
    // if curr_box matches either row or col
    if (((curr_box >= rand_row * N_size) && (curr_box < (rand_row+1)*N_size)) || (curr_box % N_size) ==
rand_col) {
      // radio on
      NETSTACK_RADIO.on();

      // send a packet
      for(i = 0; i < NUM_SEND; i++){
        leds_on(LEDS_RED);

        data_packet.seq++;
        curr_timestamp = clock_time();
        data_packet.timestamp = curr_timestamp;

        //printf("  Sending a packet at slot %d\n", curr_box);
        printf("Send seq# %lu  @ %8lu ticks   %3lu.%03lu\n", data_packet.seq, curr_timestamp-start_timestamp,
          (curr_timestamp-start_timestamp) / CLOCK_SECOND,
          (((curr_timestamp-start_timestamp) % CLOCK_SECOND)*1000) / CLOCK_SECOND);

        packetbuf_copyfrom(&data_packet, (int)sizeof(data_packet_struct));
        broadcast_send(&broadcast);
        leds_off(LEDS_RED);

        if(i != (NUM_SEND - 1)){
          rtimer_set(t, RTIMER_TIME(t) + TIME_SLOT, 1, (rtimer_callback_t)sender_scheduler, ptr);
          PT_YIELD(&pt);
        }
      }
    } else { // if curr_box does not match row and col, sleep for that 1 slot
      leds_on(LEDS_BLUE);
      NETSTACK_RADIO.off();

      rtimer_set(t, RTIMER_TIME(t) + TIME_SLOT, 1, (rtimer_callback_t)sender_scheduler, ptr);
      PT_YIELD(&pt);

      leds_off(LEDS_BLUE);
    }

    curr_box++;

    if (curr_box >= N_size*N_size) {
      curr_box = 0;
      data_packet.seq = 0;
      rand_row = random_rand() % N_size;
      rand_col = random_rand() % N_size;
      start_timestamp = clock_time();
      printf("\n=========================\n");
    }
  }
  PT_END(&pt);
}
```