CS4222

Wireless Networking

Final Project

TraceTogether

By:

CS4222 Group 5

Tiu Wee Han	A0190278X
Yehezkiel Raymundo Theodoroes	A0184595M
Zhao Pengfei	A0164637X

Disclaimer:

All the content in this report is our work effort based on personal opinion and acknowledged references. We hereby declare that we have complied with the NUS Code of Student Conduct.

National University of Singapore

2021

Neighbour Discovery Protocol

Algorithm Implemented

We implemented the Asynchronous Quorum-based Neighbor Discovery Scheme, with the following modifications to the asynchronous quorum based neighbour discovery scheme in order to further save on power consumption:

- Transmission only happens during column slots
- Listening only happens during row slots

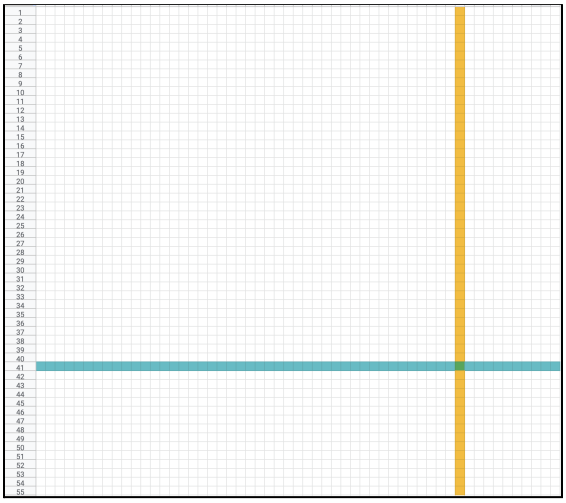

Fig. 1. Illustration of Modified Asynchronous Quorum-based Neighbor Discovery Scheme. Transmission happens in columns (yellow) and Listening happens in rows (blue)

Despite the modifications, the discovery time is still upper bounded by (N^2 x slot time), as there will always be an intersection between the row slot of the receiver and the column slot of the transmitter.

We use a value of N = 55, upper bound by 30s. This allows a near 100% (save for packet loss / possible fluctuations in internal clock) neighbour discovery within a span of 30 seconds. Each slot duration will be ($30s / 55^2$).

This is determined to have a low duty cycle of **1.98%** compared to the other neighbour discovery algorithms, based on our calculations and the simulation results in cooja. These values can be found in the Radio Energy Consumption section.

Comparison with other Protocols

Birthday Protocol

We started the project by trying out the simplest algorithm given in assignment 4. With the default settings of Wake Time 100ms, Sleep Slot 100ms, Sleep Cycle 9, the duty cycle is expected to be around 10%. This number is similar to our cooja simulation result, where we observe a radio duty cycle of 8%.

While this resulted in a very high probability of detecting a neighbour within 30 seconds, the amount of energy consumed is about three times higher than our old deterministic discovery implementation in Assignment 4. In order to be on par with that, we had to adjust the Sleep Cycle to be 49, which would give us around 2% duty cycle by expectation. We tried it in our simulation but it takes around 100 seconds just for a single discovery. As a result, we decided to use the asynchronous quorum approach and tried to improve it further by adjusting the constants for 30s deterministic discovery.

Each node wakes up with probability *p* every time slot

To achieve a duty radio cycle of 2%, both nodes will wake up for 2% of the slots, i.e. once every 50 slots. The probability that this happens is (0.02)*(0.02) = 0.0004. We can model this as a geometric distribution Ge(0.0004). In order for it to wake up expected once every 30 seconds with 95% probability, each slot needs a duration of 30s / (7467) = 0.004 seconds. Based on our experiments in cooja, this slot time is too short for receiving and hence may not capture broadcast packets. Hence, the asynchronous quorum approach is more suitable as the slot time is longer and has a near 100% probability of discovering within 30s.

Proximity Detection Algorithm

Implemented Algorithm

RSSI Readings	Results
>= -45 (Strong)	Within 3 meters
Between -45 and -80	Use machine learning to determine if within 3 meters or not
<= -80 (Weak)	Beyond 3 meters

From our experiments in Assignment 3, we have established that extreme RSSI value tends to be an effective indicator of proximity. This is further supported by the data we collected for this project: If RSSI is >= -45, the statistical probability that the sensors are within 3 meters is >99%. Conversely, If RSSI is <= -80, the statistical probability that the sensors are outside the range of 3 meters is >97%. Since the false positive and false negative rates are reasonably low, we have decided to use these values as thresholds to directly determine if 2 sensors are within 3 meters.

When RSSI readings are between -45 to -80, RSSI is no longer the only major factor that we can use to determine whether 2 sensors are within 3m. For instance, 2 SensorTags can be within 3m, but have relatively weak RSSI signal strength due to obstacles; conversely, 2 SensorTags can be more than 3m apart, but have relatively strong RSSI signal strength if they are in the line of sight. Hence, we decided to include other sensor measurements to complement the RSSI values in deciding whether 2 SensorTags are within 3m.

The 5 sensor measurements are as follows (More information can be found in the Sensing Environment section)

- Light level
- Temperature
- Humidity
- Acceleration
- Angular Velocity

When a SensorTag sends a broadcast message, it includes its own sensor readings in the broadcast. Hence, when a SensorTag receives a broadcast message, it has a total

of **11 variables** it can use to make a decision (**5** from its own environment, **5** from the sender's environment and **1** RSSI reading).

Early on, we tried to argue from a theoretical perspective how each of these 11 variables will affect the overall decision, and attempted to derive threshold values to construct a decision tree. Ultimately, we concluded that this was infeasible, for a few reasons; (i) It is difficult to assign a weight for each variable (ii) The effects of a combination of variables can be different from the sum of the individual effects from each variable. (iii) There are too many permutations of variable values to manually determine these thresholds. (More information can be found in the Previous Attempt & Motivation section below.) Hence, we have decided to take a different approach to solve this problem, which is to collect large amounts of data and use machine learning to learn the relationships between the 11 variables and the proximity between SensorTags.

When the RSSI range is between -45 and -80, the proximity detection algorithm employs machine learning techniques to predict if another SensorTag is within 3m. We adopt this data-driven (not to be confused with data-centric) approach.

To collect data, we walked around NUS with our SensorTags on to mimic how we expect them to interact in real life. We tried to go to as many different environments as possible to get a good range of readings for each variable. More information is available in the data collection section below.

For our machine learning model, we use the DecisonTreeClassifier model from the Python library **Sci-kit Learn**. This model takes in the data set containing the 11 variables as well as the proximity (within 3m), and constructs a decision tree that decides whether the sensors are within 3m of each other. To utilize this model in SensorTag, we use an existing tool **Sci-kit Porter** to output the model as a C function. This C function is included in our implementation and is called when a prediction on proximity needs to be made.

To validate the machine learning model, we tested it against a test data set of 1500 data points with known proximity values. The model was able to predict whether 2 SensorTags were within 3m with an accuracy of **94.3%**.

We chose to use the DecisionTreeClassifier model for a few reasons; (i) It has a very high accuracy rate. (ii) Few machine learning models can be ported from Python to C. For example, we wanted to use a Logistic Regression initially, but there were no available tools to port the model to C. (iii) The model must be small enough to fit in SensorTag memory. We initially tried to use a more robust model called

RandomForestClassifier that gave us a higher accuracy score, but the model size was 15MB so the compiled binary could not be loaded.

Sensing Environment

As mentioned previously, our machine learning model takes into account environmental factors to determine if another device is within 3 meters of proximity. Consequently, our implementation collects some environmental data through sensors every some interval.

The sensors used are as follows:

1. Opt-3001-sensor: Light level
2. Hdc-1000-sensor: Temperature, Humidity
3. Mpu-9250-sensor: Accelerometer, Angular Velocity (Gyroscope)

For each sensing event, we sample all sensors 4 times over one second. To guard against anomalies, we take the average value of the 4 samples as the final measurement.

To minimise energy consumption, we did not want to activate the sensors too often. On the other hand, when the environment is changing dynamically (e.g. the user is moving to a new environment), the SensorTag should be reactive to these changes.

To strike a balance between energy consumption and reactivity, we implemented an adaptive environment assessment policy, inspired by adaptive beaconing. If no significant changes in the environment are detected in the current measurement (as compared to the previous measurement), the duration to the next measurement will be doubled until some maximum threshold. On the other hand, if a significant change is detected, then the interval will be reduced back to the minimum interval duration.

We define a significant change to be 2 standard deviations, which is determined empirically based on data collected in a static environment. Finally, we set the minimum interval to 30 seconds and the maximum interval to 2 minutes. The maximum interval can be further extended if an even lower duty cycle is desirable, but would result in lower responsiveness to change of environments and less accurate proximity detection.

Data Collection

To collect data efficiently, we implemented another protocol where the SensorTags transmit and sense the environment every 1 second. Afterwards, we travelled around the NUS campus twice using similar routes.

In the first tour, we collected the positive data set by ensuring our SensorTags are within 3 meters of one other throughout. In addition, we simulated some real scenarios to diversify our data points such as getting on the bus or putting the SensorTags in our pockets.


In the second tour, we collected the negative dataset by keeping a distance of more than 3m apart from each other while travelling around the same route. Similarly, we simulated some real scenarios including the scenario when people accidentally walked in the same direction but were not travelling together.

Finally, after all the effort of ensuring little noise and diverse data points, we feed our data into the machine learning model.

Previous Attempt & Motivation

In this section, we describe our initial attempt at tackling this problem, and the motivations for us to integrate machine learning into this project.

We initially attempted to solve this task with experimental designs resembling many natural science experiments, where we identified different parameters potentially influencing the RSSI values, analyse the dependency of the RSSI value on such parameters through control experiments, followed by setting different threshold RSSI values for proximity detection based on dependency analysis.

The parameters we have identified at this point that the RSSI value can potentially depend on include Distance, Obstacle, Angular velocity and Acceleration. We have

hence performed control experiments to determine the impact of each parameter on the RSSI value while keeping other parameters constant. However, after the completion of such experiments, we realized that the RSSI values under different conditions can be fluctuating and rather arbitrarily determined. It was also very difficult to make sure the other parameters stay constant while varying one specific parameter. For instance, even if we placed the SensorTag in the same fixed location and did not disturb it, every time the sensors' readings could be different from previous readings. Furthermore, the sensors' reading tends to be machine-dependent. Even if we place different sensors at the same location, their readings could still differ. (For instance, even when we placed 3 SensorTags next to each other on the same table in the same room at the same time, their sensor readings could still differ roughly by up to 0.5 ºC.) With all these constraints, it is not feasible to determine well-defined RSSI threshold values that would be general enough to be applied on most SensorTags like what we have initially planned.

outdoors	sender obstacle	sender dynamic	recevier obstacle	receiver dynamic	suggested RSSI threshold value
0	0	0	0	0	49
0	0	0	0	1	58
0	0	0	1	0	60
0	0	0	1	1	54
0	0	1	0	0	58
0	0	1	0	1	58
0	0	1	1	0	51
0	0	1	1	1	57
0	1	0	0	0	60
0	1	0	0	1	51
0	1	0	1	0	67
0	1	0	1	1	64
0	1	1	0	0	54
0	1	1	0	1	57
0	1	1	1	0	64
0	1	1	1	1	63
1	0	0	0	0	50
1	0	0	0	1	55
1	0	0	1	0	58
1	0	0	1	1	57
1	0	1	0	0	55
1	0	1	0	1	55
1	0	1	1	0	50
1	0	1	1	1	55
1	1	0	0	0	58
1	1	0	0	1	50
1	1	0	1	0	65
1	1	0	1	1	62
1	1	1	0	0	57
1	1	1	0	1	55
1	1	1	1	0	62
1	1	1	1	1	62

Legend	
outdoors:	0 == indoors; 1 == outdoors.
sender obstacle:	0 == sender not placed in a container (e.g. a schoolbag); 1 == sender placed in a container (e.g. a schoolbag).
sender dynamic:	0 == sender not moving; 1 == sender moving.
receiver obstacle:	0 == receiver not placed in a container (e.g. a schoolbag); 1 == receiver placed in a container (e.g. a schoolbag).
receiver dynamic:	0 == receiver not moving; 1 == receiver moving.

Table: Experimental Results for our First Attempt

As can be seen, the values here are rather arbitrary and indicative only. Accurate RSSI threshold values have to be calibrated for every different SensorTag for each specific environment setting to be meaningful, which is unrealistic and defeats the purpose of building a TraceTogether application.

This hence motivates us to use more advanced Computer Science techniques to collect data differently and to process and analyse the collected data in a new light, applying what we have learnt in other fields of Computer Science. This unsatisfactory experimental outcome hence provides us with the heuristics to approach the problem differently by employing Machine Learning Techniques, which can more effectively capture the dependencies of the RSSI value on varying parameters.

Correlation between sensor variables

	Receiver Light	Receiver Acceleration	Receiver Angular Velocity	Receiver Temperature	Receiver Humidity	RSSI	Sender Light	Sender Acceleration	Sender Angular Velocity	Sender Temperature	Sender Humidity	Proximity (Within 3 m)
Receiver Light	1	0.01220966225	0.003030456271	0.2009137154	0.001576461688	0.03922759009	0.1941504766	0.01098747823	0.02610709458	0.1645039479	0.05549918588	0.01277896081
Receiver Acceleration	0.01220966225	1	0.3916561442	0.07747182294	0.0891767785	0.05848535264	0.02313908322	0.02121779607	0.06696256823	0.06764340894	0.02341449053	0.02836306796
Receiver Angular Velocity	0.003030456271	0.3916561442	1	0.08998579486	0.313263857	0.08509848351	0.03239859113	0.07764923473	0.1734439955	0.1501666582	0.01030116416	0.07768014567
Receiver Temperature	0.2009137154	0.07747182294	0.08998579486	1	0.209936994	0.2130556965	0.1752070384	0.06229185723	0.1523912291	0.6297964469	0.1816720463	0.1295994889
Receiver Humidity	0.001576461688	0.0891767785	0.313263857	0.209936994	1	0.1473846513	0.05149703479	0.02555435474	0.002501431115	0.1810564231	0.5696107779	0.370808941
RSSI	0.03922759009	0.05848535264	0.08509848351	0.2130556965	0.1473846513	1	0.03068914014	0.0293730649	0.03434384944	0.2129006769	0.189328867	0.7026630722
Sender Light	0.1941504766	0.02313908322	0.03239859113	0.1752070384	0.05149703479	0.03068914014	1	0.01326593401	0.006408682048	0.1972757289	0.007380005146	0.01216211677
Sender Acceleration	0.01098747823	0.02121779607	0.07764923473	0.06229185723	0.02555435474	0.0293730649	0.01326593401	1	0.3851030961	0.06839238059	0.09696603611	0.04019693962
Sender Angular Velocity	0.02610709458	0.06696256823	0.1734439955	0.1523912291	0.002501431115	0.03434384944	0.006408682048	0.3851030961	1	0.07130275014	0.3272573222	0.09894406065
Sender Temperature	0.1645039479	0.06764340894	0.1501666582	0.6297964469	0.1810564231	0.2129006769	0.1972757289	0.06839238059	0.07130275014	1	0.2049701624	0.1261377758
Sender Humidity	0.05549918588	0.02341449053	0.01030116416	0.1816720463	0.5696107779	0.189328867	0.007380005146	0.09696603611	0.3272573222	0.2049701624	1	0.4024058183
Proximity (Within 3m)	0.01277896081	0.02836306796	0.07768014567	0.1295994889	0.370808941	0.7026630722	0.01216211677	0.04019693962	0.09894406065	0.1261377758	0.4024058183	1

Correlation Matrix between various input variables

As expected, there is a high correlation coefficient of ~0.7 between the RSSI strength and the proximity of the SensorTags (within 3m). Interestingly, there is also a correlation of ~0.4 between humidity and proximity (within 3m). This could be because humidity values tend to be lower indoors; based on our experiment results in Assignment 3, a closed environment supposedly amplifies signal strength due to reflection off the walls and ceilings.

The other variables have a correlation coefficient of less than 0.15 with proximity, suggesting that they have little to no correlation with proximity (within 3m).

Radio Energy Consumption

The results from cooja are summarized as follows:

Parameter	Value
Clock Time	464004 (1 hour 25 seconds)
Sequence Number	724
Accumulated CPU energy consumption	5195292
Accumulated Low Power Mode energy consumption	113589401
accumulated transmission energy consumption	207822
accumulated listen energy consumption	2152689
accumulated idle transmission energy consumption	0
accumulated idle listen energy consumption	0

State of the radio	Corrected cumulative / percentage of the last cycle
Radio	1.987% / 0.17%
Tx	0.175% / 0.17%
Listen	1.812% / 0.00%

Formulae:
Accumulated energy consumption = Accumulated CPU energy consumption + Accumulated LPM energy consumption
Tx cumulative percentage = Accumulated transmission energy consumption / Accumulated energy consumption
Listen cumulative percentage = Accumulated listen energy consumption / Accumulated energy consumption
Radio cumulative percentage = Listen cumulative percentage + Tx cumulative percentage

Lessons Learnt and Challenges

One of our biggest takeaways is that a data-driven approach should be adopted as much as possible, as compared to force-fitting logic that makes sense. This is because it is hard to reason about how the sensor readings will respond to environmental factors. For example, we initially expected RSSI readings to have a strong correlation with the angular velocity, since in Assignment 3 we observed the wind to cause fluctuations in RSSI readings. However, this was not the case after we tested it out. Even though the guesses made sense to us, our hypotheses were eventually disproved by data.

In the end, we decided to go with a more data-driven approach where we collected large volumes of data and used machine learning to infer the pattern for us. With this approach, we are finally able to build an accurate predictor in a much more efficient manner than simply applying our intuitions and logic.

Another lesson that we learnt throughout the project was to manage the tradeoffs, such as between energy consumption and accuracy of detection. During our brainstorming stage, we spent most of our discussions on the tradeoffs. In addition, some protocols were also implemented just to compare their performances before we chose the best out of all of the protocols.

One major challenge that we faced was that the sensor measurements were not consistent across SensorTags. For example, when we placed our 3 SensorTags together in the same environment (e.g. the same room), they could produce different measurements of temperature and humidity. To address this, we mixed our training data from all 3 devices in the hope that our classifier is general enough to be applicable to other SensorTags.

Finally, it was also a challenge to understand the Contiki code structure, and faced many major issues throughout the project. For instance, we could not move parts of our code to other functions due to Contiki's C-macro. In addition, we encountered an unexplainable bug when printing too many statements could cause the program to block. We suspect this to be a synchronization issue where too much I/O ended up blocking the process.

Contributions

Group Member	Contributions
Yehezkiel Raymundo Theodoroes	● Birthday Protocol Algorithm ● Sensors Implementation & Testing ● Data Collection ● Experimental Design ● Code Testing
Tiu Wee Han	● Asynchronous Quorum-based Neighbor Discovery Algorithm ● Decision Tree Machine Learning model ● Cooja ● Experimental Design ● Data Collection & Analysis
Zhao Pengfei	● Proximity Detection Algorithm Pruning ● Implementation Testing ● Experimental Design ● Data Collection & Analysis ● Overall Logistics & Admin