CS912: Trickle simulation project

Jiyu Yan
Student ID:1851015

I. Introduction

Trickle[1] is a self-regulating algorithm for code propagation and maintenance in wireless sensor network. In this project, Cooja in Contiki operating system[2] is used for simulation as Trickle is a standard library in Contiki. A sensor network application is proposed as project requirement and a number of simulations are executed to analyze the relationship between Trickle parameters.

II. Trickle

How to update the code, application or parameter that has been put in nodes including discovering bugs has always been an important issue in wireless sensor network. However, code propagation for these tasks is expensive for several reasons: large size change, all nodes need to know as soon as possible, redundant messages, when to propagate.

The balance between propagation speed and maintenance cost is achieved by three parameters on each mote: a consistency counter c; a redundancy constant k to repress broadcast redundancy; and a gossiping interval I to control the time interval of propagation, together named Trickle tuple. The transmission time t is within an interval $[I/2, I)$. In addition, Trickle defines some other configuration parameters, namely I_{min}, the minimum interval size in units of time; the maximum interval size I_{max}, which is described as a number of doublings of I_{min}. Suppose I_{min} is 100ms, I_{max} is 8, then the interval value I is set to a random value between $[I_{min}, I_{min}*2^{I_{max}}]$ = [100, $100*2^8$]ms.

When Trickle starts, after the interval I is set byI_{min} and I_{max}, transmission time t would be pick at $[I/2, I)$. When a node hears a same message, it increments c by 1. At time t, a node transmits data only if c is less than k. Otherwise($c > k$) the transmission would be suppressed. The intuition behind it is that if a node hears a lot identical messages during a period, it doesn't need to transmit its own data anymore since the network already known about it as the node receives that data quite often. Until reaching to the time specified by I_{max} and fixing that interval length, Trickle doubles the interval length after I expire. I would set to I_{min} if a node receives inconsistent data and $I > I_{min}$. c is reset to 0 and t to a random value in $[I/2, I)$ after a new interval begins. The whole flowchart of Trickle is shown in Figure 1.

III. Experimental setup

For this project, the essential implementation is to write a sensor network application based on Trickle library in Cooja to meet requirement. Rather than a single number in 'token' be transferred between nodes, an integer array with size 5(up to 5 different proposers) is used to store all potential values proposed by proposers.

Apart from that, the main change is located in *trpic_handler* method. Each node must compare the array it received with its own to determine whether it is consistent. If all numbers(not count 0, since 0 is default initial number and proposer only proposes a number in the range 1 to 100) are same, it is consistent.

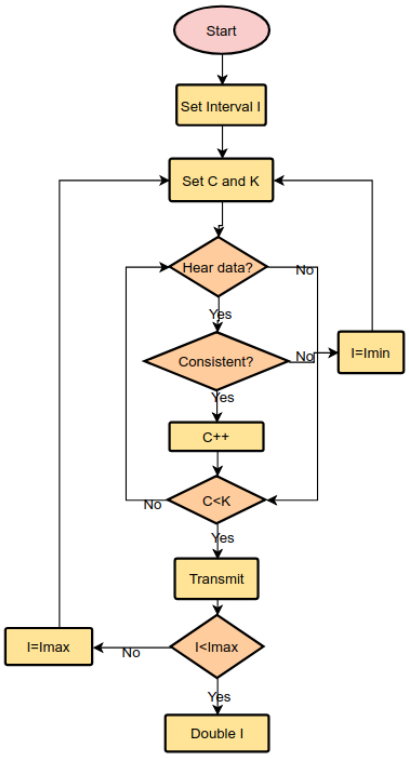

Fig. 1. Trickle algorithm flowchart

If there's at least one number one mote has, another mote doesn't have or vice versa, it is inconsistent. The node will add new value to its own array if it does not has that number before. For example, node A has array(40, 50) now and it received an array(40, 50, 90), then it add new value 90 to its array and will send(40, 50, 90) from now on. The 'token++' related codes are deleted as requirements.

The only difference between proposers and other nodes is the initial array for proposer has one none-zero vale between 1 to 100, other nodes have an initial all zero array. Each node prints the average value in its array(not count 0), if the number of none zero value in its array is equal to the number of proposers in a simulation, it prints 'ok' to represent that it already gets the correct average number.

A python script is written to accept the testlog file(output log, has all time, node id and event) from simulation, and show the average value computed by each node at the end of the simulation, and the first time when all nodes get the correct average number if it happened, and it also could print the number of all messages in this simulation.

All simulations are tested with script without GUI to be efficient, with the setting: new random seed on reload, to produce different results at each running. The mote startup delay is set to 1000ms as default and all motes are using default transmit range. All results are the average of 5 times running.

Both dense and sparse network need to be define and set before simulation. Simulations with all default value are tested at first, with 3 proposers and 61 normal nodes, default transmit range and default position interval 100 (x:0 to 100, y:0 to 100), the time for all nodes getting correct average value is shown in tableI.

The position interval is changed to some big value, however it makes the network too sparse that some nodes may never receive a single message since there's no other nodes in its range. After several tests, 200 is chosen as the x and y position interval to make a sparse network, 80 is chosen to make a dense network. Only simulations for diameter(max hop) vs latency are using linear position, others using random position. It happened once for sparse network, one mote has no neighbor in its range so it couldn't get the correct average value, it is moved to the network center with mouse to make sure every mote could get the correct value in the end.

IV. Results and Discussion

TABLE I
PROPOSER 3, INTERVAL 100

time for all correct value	75.1	47.4	54.8	53.3	66.8
avg time	59.5				

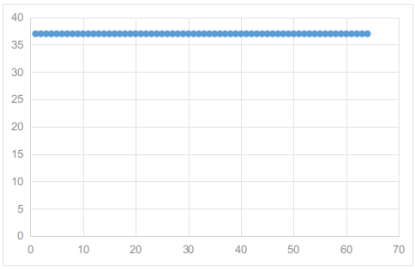

Fig. 2. id vs average value

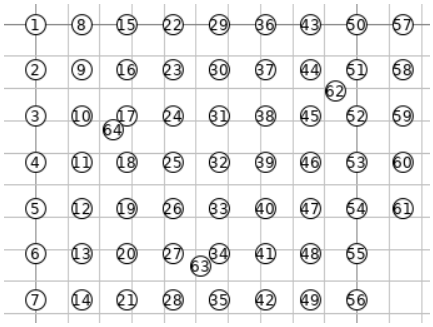

Fig. 3. One topology for diameter vs latency

A. *id vs computed average value*

Dense network(position interval=80) is used in this subsection. The simulation is set with 3 proposers and 61 normal nodes, in this project all simulations use the default transmit power. For convenience the proposed value is the same (5, 83, 24) with 3 proposers and the average value need to be 37. For all 4 different configurations(no need for detail here), the results end in the same shown in Figure2.

B. *diameter vs dissemination latency*

The diameter is defined as max hop between any pair of nodes in network here, later another version of diameter(max straight length between nodes) would be analyzed as a supplement. The position of motes is set to linear(3 proposers with random positioning) for convenience counting the max hop. For example, in Figure3 the max hop is 14. The simulation time is long enough to make sure all motes get the right average value, and the latency is computed by a python script, to find out the first time when all nodes print out 'ok' which means get the correct average value.

1) Configuration 1:k=2,Imin=12,Imax=10
2) Configuration 2:k=2,Imin=16,Imax=10
3) Configuration 3:k=3,Imin=16,Imax=10
4) Configuration 4:k=4,Imin=12,Imax=10

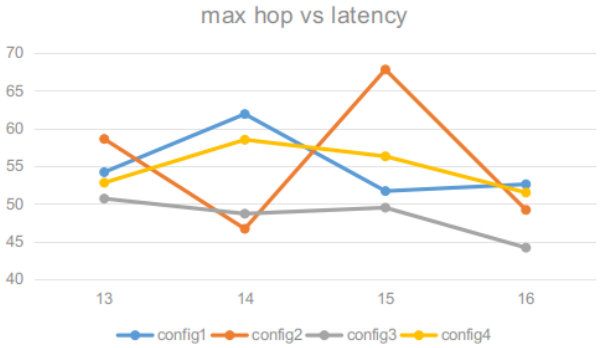

Fig. 4. diameter vs latency for dense network

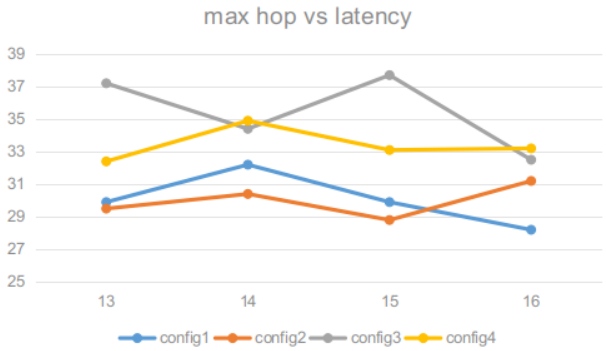

Fig. 5. diameter vs latency for sparse network

The diameter varies from 13 to 16 for both network, and 4 different configurations are shown in above. Figure4 shows the dense network and sparse network is shown in Figure5. By comparing the Y axis between Figure4 and Figure5, the average latency is much smaller in sparse network since the collusion appears a lot in dense network.

Supposing the latency increases as the diameter increases since messages may need more transmission between motes in a network with large diameter. However, the results couldn't prove that, as this trend isn't clear enough. This may due to the hop variance is too small to show the difference.

For these 4 configurations, the best configuration couldn't be decided as they perform differently in different networks. The most obvious one is config3, it performs best in dense network but worst in sparse network. Config 1 and 2 is intended to compare the effects of *Imin*, basically no difference could be found, except config2 is very unstable in the dense network. Theoretically, motes with smaller *Imin* tend to be active faster when inconsistent. However, this configuration is changed for all motes in one simulation so the impact of *Imin* is not significant.

Between Config2,3 and 1,4, the purpose is to analyze the redundancy constant k. Motes with larger k supposed to have more chance to transmit messages. That effects is more obvious in sparse network since the dense network already has too many messages no matter how k changes. In these sparse network, config1 and 2 has smaller latency than 4, 3, shows with smaller k, the latency is shorter when other 2 parameters are equal. This phenomenon shows that when k is small, there is less collusion in the network, communication is smoother and there is no need to increase k to make the load unbalanced, with unnecessary energy consumption.

C. number of proposers vs number of messages

The simulation time is set to 100000 as a script shown below, and by counting 'Received' in testlog by a python script, the total number of messages transferred in network during simulation is gathered. The only variable in four different configurations is the *Imax*, to figure out the effects of *Imax* for network.

```
1 TIMEOUT(100000);
2 while (true) {
3 log.log(time + ":" + id + ":" + msg +
      "\n");
4 YIELD();
5 }
```

1) Configuration 1:k=2,Imin=12,Imax=10
2) Configuration 2:k=2,Imin=12,Imax=8
3) Configuration 3:k=3,Imin=12,Imax=9
4) Configuration 4:k=3,Imin=12,Imax=11

Different tendency is observed between dense and sparse network, unexpectedly, shown in Figure6 and Figure7.

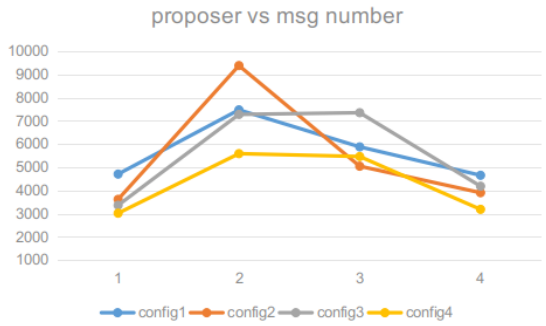

Fig. 6. number of proposers vs number of messages for dense network

For both network, number of messages increased as proposer change to 2 from 1, means adding one another proposer brings more message inconsistent and increases the number of

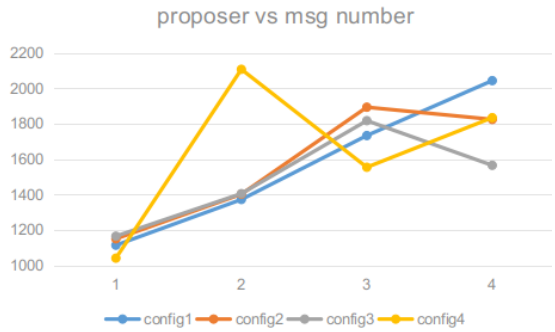

Fig. 7. number of proposers vs number of messages for sparse network

messages spread on the network, which is easy to explain. However, as the proposer increases from 2 to 4, the number of messages in the sparse network increases shown in Figure6, while the overall trend in the dense network decreases shown in 7.

Comparing the y-axis, the number of messages in dense network is much larger than sparse network. In dense network, each mote has much more neighbors in its range and more message inconsistencies are detected, which causes more collisions as the number of proposers increases. On the contrary, the network capacity in sparse network is big enough to hold increasing proposers.

Theoretically, when the network tends to be stable, nodes with smaller *Imax* transmit more frequently, while suppressing nodes with larger *Imax*. If the network is stable, this will result in long term unbalanced transmission load, and nodes with smaller *Imax* will quickly run out of energy. Since the *Imax*changes concurrently among all motes in each one simulation, the effects is unclear as the change of *Imin*.

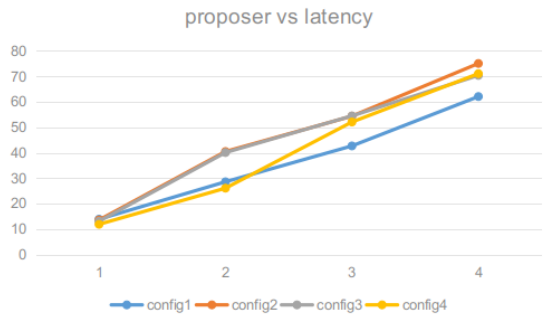

Fig. 8. number of proposers vs dissemination latency for dense network

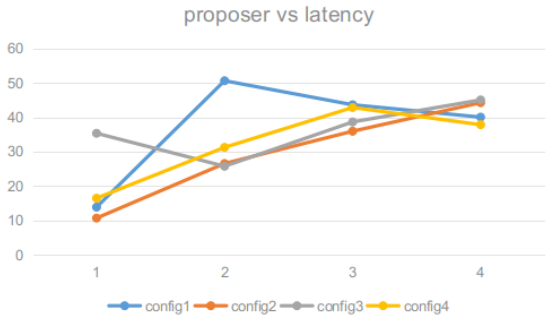

Fig. 9. number of proposers vs dissemination latency for sparse network

D. number of proposers vs dissemination latency

1) Configuration 1:k=4,Imin=10,Imax=9
2) Configuration 1:k=2,Imin=17,Imax=11
3) Configuration 1:k=3,Imin=14,Imax=10
4) Configuration 1:k=5,Imin=12,Imax=12

The results for number of proposers vs latency is clearly shown in Figure8 and Figure9.

The overall latency in sparse network is smaller than dense network, which is consistent to question2. As the proposer increases, it's more difficult for network to compute all correct average value since it needs more message updates for every mote, as the latency increases shown in Figure8 and Figure9.

The configurations are set differently for all 3 parameters, however, the results are very close to each other. It may due to the effect of the proposer's number on the latency is much larger than the parameters with little difference between each other.

E. network diameter(Straight line distance) vs latency

This section is a supplement, as I thought the network diameter is the straight line distance of the pair of nodes farthest from each other in the network. The difference of configuration is intended to analyze *Imin*, results shown in Figure10 and 11.

1) Configuration 1:k=2,Imin=4,Imax=10
2) Configuration 2:k=2,Imin=8,Imax=10
3) Configuration 3:k=3,Imin=16,Imax=10
4) Configuration 4:k=4,Imin=24,Imax=10

Since this simulation uses random position for all motes, the location of proposer has huge

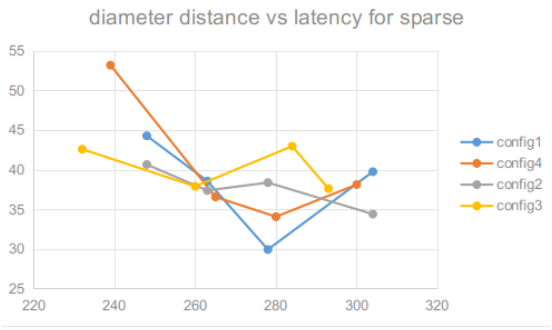

Fig. 10. diameter distance vs latency for sparse network

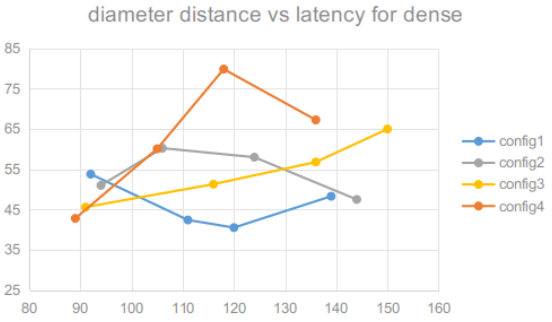

Fig. 11. diameter distance vs latency for dense network

impact on results rather than configurations or the length of diameter. The diameter is changed by move two farthest motes without changing the whole topology, which makes the results irregular and difficult to interpret.

V. Conclusion

Trickle algorithm requires nodes in the network to continuously detect whether an upgrade is needed when new code is upgraded. This process is called the maintenance; when a new update occurs, all nodes in the network can receive the code and upgrade at the fastest speed. This process is called the propagation. Instead of flooding packets over the network, Trickle algorithm controls the send rate so that the node receives very few packets and is only enough to keep continuously updated. In order to achieve both high propagation rate and low maintenance cost, the nodes will adjust the frequency of ”polite gossip”, and when new code appears, they will ”communicate” more frequently.

In this project, Trickle algorithm is analyzed with 4 different questions, each with 4 configurations for both sparse and dense network. The dense network(with 80 positioning interval) tend to have more messages, more latency than sparse network(with 200 positioning interval). A mote with a larger redundancy constant k has more chance to transit and it may cause collusion and transmission load unbalanced. *Imin* and *Imax* together decide the gossip interval. When a mote detects ”inconsistency”, the mote with smaller *Imin* will respond faster. When the network tends to be stable, motes with smaller *Imax* transmit more frequently.

Several other factors may have a greater impact on the simulation results than configurations, such as the topology of network(whether random or linear etc.), the position of proposers(far from the network center or not), many issues deserve further exploration.

References

[1] P. Levis, N. Patel, D. Culler, and S. Shenker, “Trickle: A Self-Regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks,” p. 14.

[2] “Contiki: The Open Source Operating System for the Internet of Things,” http://www.contiki-os.org/index.html.