Implementation of Tag protocol on tinyOS

TinyOS free simulator

Fragkoulis Logothetis
Electrical and Computer Enginnering
Chania ,Crete

Abstract

Tiny Aggregation (TAG) service for aggregation in low power, distributed, wireless environments. TAG allows users to express simple, declarative queries and have them distributed and executed efficiently in networks of low-power, wireless sensors.

I. INTRODUCTION

These project consists some pieces based on each other

- A C-program that creates a grid topology ,to make a various of tests in the simulation
- The basic part of the project is to simulate the Tag protocol ,using timers and send/receive messages implementing a cumulative function according to the Tag .

II. ANALYZE PARTS OF THE PROJECT

A. Building a grid -Topology

It is useful to control the operation of our program creating topology files, including multiple sensors, by a "Automatic" way. In this programm we are prompted to create a program which :

- It takes as integer 1 parameters (we will refer to it as diameter D) and a floating point number (we will refer to it as signal range).
- Will create D * D nodes
- Considering that the horizontal and vertical distances of the nodes in the grid are equal to 1, it is easy for any node to find all the nodes which are less than or equal to its range . For example, if the range is 1.5, then a central one node has 8 neighbors .
- If for each node you find its neighbors, then you can this information use it to create a topology file.

B. Simulate Tag Protocol

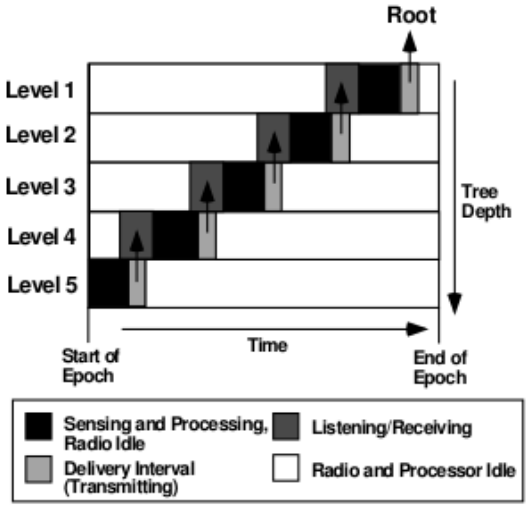
Each sensor would produce a random value every 60 seconds in order to give his father appropriate information to find average and dispersion of measurements across the network.AVG and VARIANCE functions are simultaneously calculated . The simulation ends after 60 seconds. The base station (node 0) should prints the final result in each season.The value of each node with a K-identifier in each season will be a random integer number in interval [K..K + 20].

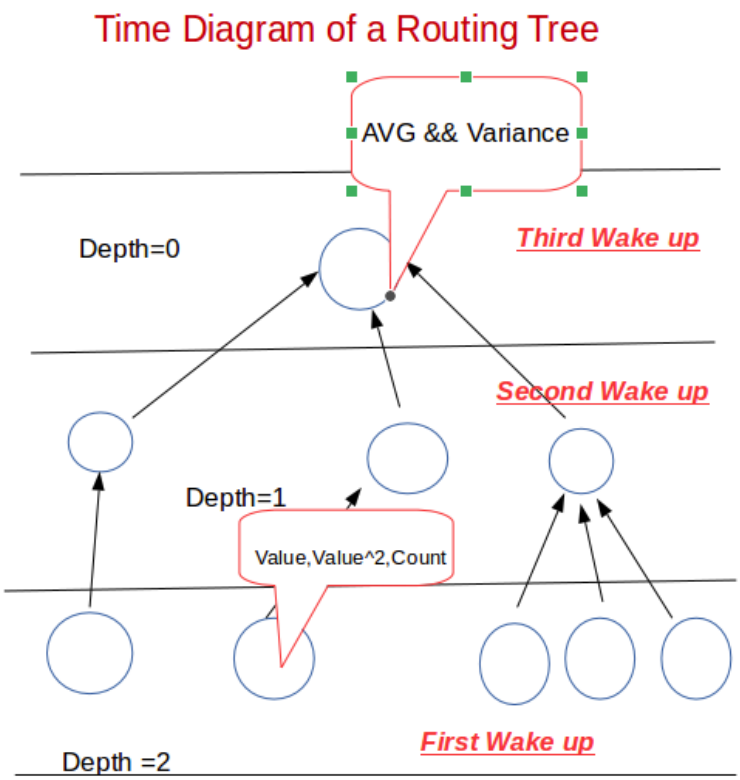
C. Routing Procedure and Wake Up Time

The EPOCH DURATION lasts 10.000 ms. During the half of the first period a routing procedure is token place .Each node sends a broadcast message to incorporate some new nodes in their sub-tree. When a new node receives a Broadcast message ,then it defined the sender of the message as its parent . Then the node re-broadcast the routing message . This node will wake up from sleeping mode after :

ROUTING_TIME - (PROCCESSING_TIME*(curdepth+1)) + (MAX_SENSORS – curdepth) * PROCCESSING_TIME + rand()%80

The worst latency will be took place when the route of the nodes are like a connected list. So I have used this approach to built the Timer .





D. ***Calculate AVG and VARIANCE***

```
typedef nx_struct AggrMessage
{
    nx_uint16_t senderID;
    nx_uint16_t parentID;
    nx_uint16_t product;
    nx_uint16_t sum;
    nx_uint16_t count ;
}  AggrMessage ;
```

AVG = sum /count and

VAR = product/count -AVG^2

Where Product =Value_Of_Sensor^2 + Children_Product

and Sum = Value_Of_Sensor + Children_Sum

E. ***Helpful Code and Changes***

A helpful code was given to us , to have a first-abstract view of the Routing Process . But some parts of these code was useless .For example I have deleted the following Parts :

- Code for Serial Communication
- Notify Parent for their children

F. ***Optimization to avoid Collisions***

- Small Scripts in Event Handlers
- Small Atomic Blocks

G. Child Cache

To improve the quality of aggregates, we propose a simple caching scheme: parents remember the partial state records their children reported for some number of rounds, and use those previous values when new values are unavailable due to lost child messages. As long as the duration of this memory is shorter than the interval at which children select new parents, this technique will increase the number of nodes included in the aggregate without over-counting any nodes. Of course, caching tends to temporally smear the aggregate values that are computed, reducing the temporal resolution of individual readings and possibly making caching undesirable for some workloads. Note that caching is a simple form of interpolation where the interpolated value is the same as the previous value. More sophisticated interpolation schemes, such as curve fitting or statistical observations based on past behavior, could be also be used.