UNIVERSITÀ DEGLI STUDI
DI TRENTO

Wireless Sensor Network
2012-2013

Project:
Round Robin Routing Scheme

Sandro Cavallari

Abstract

This document briefly describes the algorithm implemented to solve a Round Robin scheme for collect data among a set of wireless nodes running TinyOS as a backbone. It also compare the performance of a round robin scheme respect of a simply algorithm where node forward message to a single parent.

Contents

1 Assignment

The main goal of this project is to create an round robin algorithm for collect data among a tree network. In this protocol every node can have from 0 to 4 parent, so a node have to select the parent to which to forward messages according to a round robin scheme: every parent of a node have to receive the same number of messages.
Here are the specification of this protocol :

- Suppose that a node(*k*) have P_a and P_b as parent. The node *k* have to forward messages to P_a and P_b in a way that: $0 \leq \#msgP_a - \#msgP_b \leq 1$.Where $\#msgP_i$ is the number of messages forward to node P_i
- Messages that are forward do not need acknowledgement. If a node lose a packet, the system does not require to resend it(*Best Effort* transmission)
- The metrics, used to build the tree, is the hop count distance for the sink. No estimation of the channel is used.
- The resulting topology must be a tree without loops: a directed acyclic graph.

The protocol uses some fixed value for better perform the tasks:

- Data collection sending interval is $5s$
- The tree is build every $5min$
- Every node broadcast the building tree messages a random number of time for $2s$
- Every node can have at most 4 parent, the system will drop the excess parent.
- For implementation reasons the infinity value (∞) is set at 65500

The protocol is implemented using **TinyOS** as backbone and **Tmote Sky**(1) as type of node.

2 Tree Building

Under the *Round Robin data collection application* runs the **Tree Building** application, that create a new tree every $5min$. In the protocol the sink node that start the construction is the node with $ID = 0$, so all the resulting trees have node 0 as sink.

In both applications every node has at most 4 parent, in fact 4 is a reasonable number respect to topology of 15, 20 and 30 nodes. When a new parent is found, the application save in an appropriate data structure:

- The cost to arrive at the sink
- The sequence number, that identify the tree
- The ID of the parent

Figure 1: A Tmote Sky wireless sensor network platform

```
typedef nx_struct p_data {
    nx_uint16_t parent;
    nx_uint16_t seq_no;
    nx_uint16_t cost;
} parent_data;
```

The tree is build in a top down strategy from the sink. After 1 second that the node start, the sink send in *Broadcast* messages containing:

- Distance from the sink
- Sequence number of the tree

```
typedef nx_struct TreeBuilding{
    nx_uint16_t cost;
    nx_uint16_t seq_no;
} TreeBuilding;
```

All the node that receive the message update their parent vector and start to send in *Broadcast* another message for expand the tree. Node send messages in *Broadcast* for $2s$ as many time they can so that more node can be contacted.The receive event take care that the resulting tree is well formed and there are no loop or overlapping node. Every time that a node receive a message witch create a better path or receive a message with a different sequence number, it restart to send messages for $2s$, so that every node will update his status. Since a random number of messages were broadcast, some nodes can found a costly parent and have to notify it at all his child. So every time that a node found a better path, it restart to forward messages.
In figure 2 is possible to see the construction of a tree, notice that node 14, 11 and 9 found a better path.

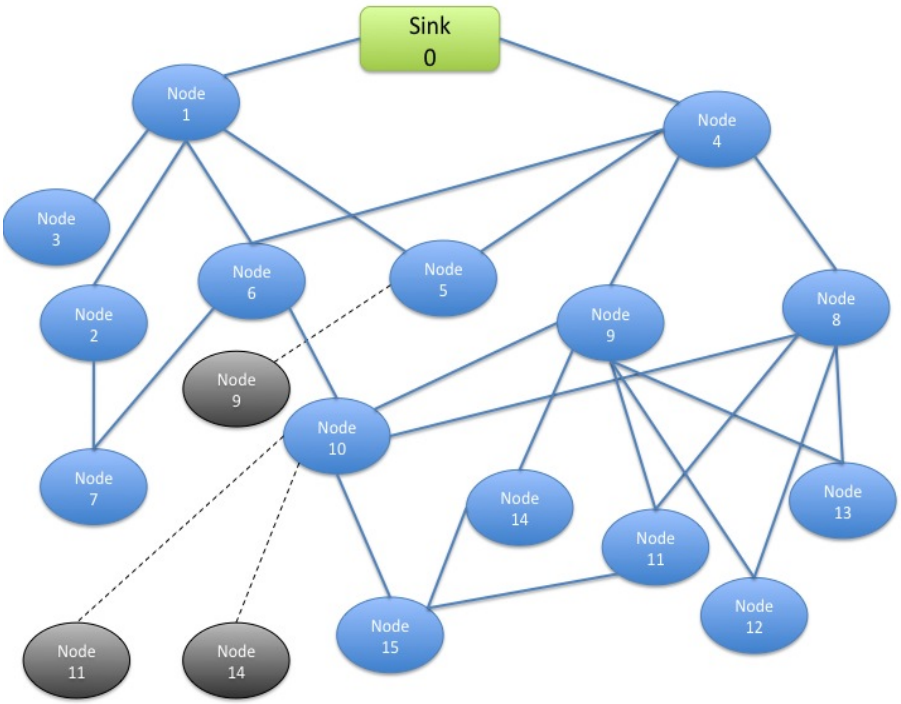

Figure 2: The resulting tree of a topology with 15 nodes, note that some node found better path after the first one

3 Tree Connection

The two application, the *Tree Building* and the *Data Collection*, communicate using a interface call *TreeConnection* that implement the event *parentUpdate*. The parentUpdate event is implement in the Data Collection module-component and allow to update the parent of Data Collection node's. When a new parent is found in the Tree Building application the parentUpdate event is signalled and update the Data Collection nodes. The parentUpdate event is signalled because the invocation of an interface event need the signal keyword.

4 Data Collection

The **Data Collection** application after receiving the tree structures, start to perform the *round robin sending*. Every node have two queue, one for the receiving messages and one for the sending messages.

```
interface Queue<DataMsg> as Queue;      //receiving queue
interface Queue<DataMsg> as QueueSend;  //sending queue
```

Every time that a node receive a message will enqueue it in the receiving queue, so every $5s$ the *round robin procedure* will be execute. The round robin function, knowing how may parent have a node(parent_n) and the last parent that receive a messages(index_rount_robin), can perform the round robin scheme in this way:

- Check if a node have some messages to forward to the parent
- Dequeue the messages form the receiving queue
- Set the address of the message so that the round robin will perform
- Enqueue the updated message in the sending queue
- Update the round robin index(index_rount_robin)

When the round robin function complete his work every node start to forward all messages that are in the sending queue using the address that the round robin function choose. In this way every $5s$ a node can perform the round robin if have some message to send.

For tracking the round robin performance every node count how many messages forwarded to his parent using an appropriated data structure:

```
typedef nx_struct SentToParent{
    nx_uint16_t parent;
    nx_uint16_t num_msg_send;
}num_msg;
```

This data structure is reset every time a new tree or a parent with a lesser cost is found. In this way is possible to check the performance of the round robin scheme.

5 Less Selection

For comparison reasons, an other version of the Data Collection protocol is developed. This version is more simple: instead of performing a round robin over all the parent, every node choose the parent with the lesser ID and forward messages only to him. This protocol, called **Less Selection**, have a similar implementation respect to the *Round Robin* version, in fact only the round robin function change. Every time that a node found a new parent compute the address of the parent with the lesser ID and save it in a global variable *address*. So before forwarding the messages every node can set the address witch to send the messages.

6 Performance

To verify the performances of the Round Robin protocol and the Less Selection protocol, 3 different topology of 15, 20 and 30 node were used. Each node has been connected to 4 randomly chosen nodes with a symmetric link quality randomly chosen from -100dB up to -60dB, so that the largest part of the link are good enough and allow good information exchange.

Every node has as 2 queue of 15 slots, so that can save 15 incoming messages and 15 outgoing messages and every node have to forward 10 messages at every sequence.

How is possible to see form the test, the **Round Robin version** can balance the charge of the network between all the parent of a node. More parent have a node and more the network will be balanced. In order to perform the round robin this protocol require more complex and longer computation than the Less Selection. Due to the intrinsic complexity of the *round robin algorithm* this version perform better with bigger topology in witch a single node has not enough resource

Table 1: Round Robin

Avg. of messages dropped	Topology
18	15
26	20
51,4	30

Table 2: The resulting analysis of the Round Robin protocol

Table 3: Less Selection

Avg. of messages dropped	Topology
15,2	15
32,4	20
58,38	30

Table 4: The resulting analysis of the Less Selection protocol

to manage all the traffic. Farther due to the restrict queue size, with topology bigger that 15, the Round Robin version drop less messages only at the top level of the tree (one can use more powerful node only at the top of the tree).

Instead, the **Less Selection version** is more simple in a computational point of view: in fact forward all the messages to a single parent. This protocol, for his simplicity, perform well in small networks where the few amount of resource that a node has can not create bottleneck in the network. Instead, respect to the Round Robin version, this protocol can not scale really well because can not balance the traffic trough the different parent that a node has. So with big topology the Less Selection protocol will:

- Droop packet due to the queue size
- Create a lot of collision and bottleneck
- One have to use more powerful node in more layer of the tree for improve the performance

After some experiment, seems that with topology containing less that 20 node the Less Selection protocol is better that the Round Robin protocol. Opposite, if the topology contain more that 20 node the Round Robin version can balance the work better and lose few packet.