Fault-Tolerance Algorithm for Sensor Networks with Data Mule

Course Project

CSC 714 Real Time Computer Systems

by

Karthik Parasuram (kparasu@ncsu.edu)

Manav Vasavada (mmvasava@ncsu.edu)

Amey Deshpande (asdeshp2@ncsu.edu

April 24, 2009

Index

1 Introduction

1.1 Motivation

Wireless sensors are typically deployed over a distributed geographical area to gather sensor readings. In many cases, deploying multiple access point as gateways for sensor nodes to the back-end network is not feasible, for example because of remote location of the sensors. In such cases, the data collection has to be performed using a multi-hop network, in which the intermediate nodes forward the data gathered from the nodes that are farther from the gateway to the nodes that are closer to the access point. Another motivation behind using multi-hop networks is potential power-saving achieved, as the nodes can send data at lower radio transmission power, as they send it over a shorter distance. Because of this distributed-nature of the network, there are some issues that need to be addressed. One very common case is failing nodes. If a node fails, it can potentially isolate a bunch of sensors from the access point, as it will no longer perform data forwarding. We consider this scene and implement multiple solutions to achieve robust self-recovery in the network.

1.2 Goals Achieved

We took up the implementation in Contiki OS, so it involved an initial learning curve to get familiar with it. We studied communicatiion stacks in Contiki and chose Rime stack for implementation. We designed a fault-recovery algorithm to be used within the sensor nework and have successfully implementated it. We also use a novel concept of data mule. The data mule is a Lego RCX robot. We have a host PC application that is the final destination of sensor data. It interprets the current state of the network and commands the Lego RCX bot using the LNPD protocol to move around in the sensor network.

1.3 Terms Used

1. Coordinator Node: The sensor node connected to the host PC. It is the bridge between the remaining sensors, which use Zigbee and the host PC using UART.
2. Router Node: These are the nodes in the sensors network that can perform data forwarding.
3. Slave Nodes: These are the sensor nodes that only gather sensor data, and send it to a known node for forwarding
4. Sink Node: A sink node acts as a group-leader for several slave nodes., and sends the gathered data to a router. The sink and router functionalities can co-exist on a single sensor node.

1.4 Task Distribution within Group

Study existing fault-tolerant algorithms and choose possible functions	Karthik
Discussion for comparison of OSes and deciding which OS to use (we decided to use contiki)	Manav, Amey
Study and get familiar with Rime protocol API in Contiki	Common
Design and implement the failure identification functionality of coordinator mote	Karthik, Amey
Identify and resolve the timing issues in mote-mote communication	Karthik
Getting familiar with the simulation tool that comes with Contiki OS.	Amey
Study and modify the routing table API in Contiki	Amey, Karthik
Implement routing functionality in sensor nodes	Amey, Karthik
Design and implement dynamic re-routing feature for fault-recovery	Amey
Setting up LNP on the system and getting it working with the RCX	Manav
Design Lego RCX rover, establish rover to PC communication using LNP	Manav
Writing the coordinator application on PC to maintain network graph and detect isolation.	Manav

2 High Level Design

2.1 Sensor Network

Among the sensor nodes, we identify three generic functionalities. The implementation details are discussed in the next section.

Funcitonality	**Who can do that**
Sensing	Slave / Sink / Router / Coordinator
Data Gathering	Slave / Router / Coordinator
Data Forwarding	Router (to next router) / Coordinator (to PC)
Re-Routing	Sink / Router
Isolation Detection	Coordinator

Out of these, data sensing and data gathering are relatively straight forward and will be discussed in implementation.

Data forwarding is the key funcitonality in any multi-hop network. A node supporting this feature maintains a routing table in memory. The routing table is formed on-the-fly when the node starts. Also, the routing table is dynamically updated to reflect newly joining router nodes. Re-routing feature should be provided for all routers to adapt the routing table because of changed next-hop nodes, either because of mobile nodes or failing nodes.

Isolation detection is rather a specialized (and hence centralized) feature, present at the coordinator end. The coordinator node continuously gathers sensor data from the nwetwork. It periodcally checks if it is receiving data from all nodes within the network. In this way it is able to identify failed nodes and reports the address of failed node to the application running on PC.

A more detailed consideration of these aspects can be found here.

2.2 Data Mule and host PC

The data mule comes into play when the node failure at one point results in isolation of the sensor nodes that were forwarding data through the node before failing, and have no alternative path because of the failure. The application running on host PC maintains graph topology of the network. Because of isolation, the coordinator node may conclude that all the isolated nodes have failed and report the host PC accordingly. The host then figures out the exact location of failure in the network. Based on node-address and physical location mapping, it commands the data mule to move to the location of failed node. The data mule has a sensor node with router functionality. As a result, the data mule replaces the failed node and bridges the isolated network. This bridging is eventually recognized at the coordinator node.

3 Implementation Details

The implementation of the project was on two different hardware platforms: the sensor nodes and the Lego RCS kit.

3.1 Sensor Nodes

We used TelosB motes as sensor nodes. We carried out the implementation in Contiki OS. We discuss implementation of the sensor node functionalities described above.

- **Data Sensing**

Contiki has sufficiently rich API for reading sensor values. For example, reading Sensirion temperature sensor is just a function call, the CC2420 radio tranceiver can be turned on or off as required, and so on. Contiki also has timer API with timer-expiry event delivery. Using these, periodic sensing can be

easily achieved.

Data Gathering

This funcitonality calls for inter-mote communication. Contiki provides a number of options to choose from for data communication between motes. Two prominent of these are the Rime communicatiion stack and the TCP/IP stack. The Rime communication stack provided primitive send-receive calls, with options for best-effort sending, reliable sending (based on packet sequence numbers and ACKs) and broadcasts. These sending features have associated receive callbacks, that are invoked upon reception of data packet. The TCP/IP stack complies existing standards and even features IPv6. We believe that the TCP/IP features are too sophisticated for our application. For example, the reliability feature of TCP will call for muliple retransmissions, resulting in network traffic and battery power consumption. Also, we don't have a large degree multi-processing where the sockets API of TCP would be convenient. As a result, we chose Rime communication stack and decided to implement the algorithm using primitive send-receive calls.

We first tested with best-effort send-receive calls. During simulation with multiple nodes, we discovered that the Rime stack has network buffer that can hold only one packet at a time. As a result, if two packets arrive in succession, the earlier of them is lost if it is not processed before arrival of the second. Detecting such packet collisions at receiver is not possible. Hence we switched to the reliable send-receive calls in Rime stack.

Routing

Contiki has built-in implementation of managing a routing table. The existing implementation in Contiki has some sophisticated features, such as a time value associated with each entry and flushing that entry if it is not during that much period of time. In our implementation, the next hop for the coordinator as destination is relatively constant, we do not require this feature. On the other hand, we required additonal feature to keep track of failed deliveries while sending data to coordinator via some next-hop router. For this, we suitably modified the existing routing table implementation in Contiki.

As of now, the unicast data flow is always from a sensor node to the coordinator node. Hence, all entries in the routing table have the same destination address. Thus each entry in routing table refers to an alternative path to the coordinator. When a router receives a unicast data, it checks if the data is to be forwarded, based in the message type field in the packet. If yes, it picks up the first entry in the routing table and sends the packet to the next-hop node as per that entry. When this unicasting fails, a timeout call in invoked. (The timeout callback is provided by the reliable unicast API in Contiki). Such failure occurs when the node is unreachable, either it has moved out of transmission range, of node-failure. Upon such timeout, the node increments counter associated with this next-hop for failed deliveries. If the counter exceeds some threshold, the node decides that the next-hop node has failed, removes that entry from routing table. If upon removing, the routing table is empty, it marks itself as “isolated”. In this way, we achieve dynamic re-routing in presence of node failures.

A router also “advertises” itself periodically. For this, it broadcasts its own address and “hop-count”. The hop-count field indicates the number of hops required to reach coordinator from that node. When the neighboring routers receive this broadcast, they update the routing table only if the hop-count in the

recieved advertisement is better (lower) that their own hop-counts. This periodic advertisement allows any router node to join a functioning sensor network. This results in support for mobile sensor networks and hence also for the data mule feature.

- **Coordinator Node**

The coordinator node is essentially a bridge beween the Zigbee network of sensors and the monitoring application running on a host PC. It starts the network formation by sending out advertisement message with its own id, and hopcount. This node is initialized with zero hopcount (while all other routers described above are initialized with hopcount as infinity). This node, like other routers, can also perdiodically broadcast to support dynamically changing coordinator nodes, possibly because of node failures.

During the network setup phase, each router unicasts its routing table to the coordinator mote. The coordinator gathers the routing tables and stores in memory. It keeps these tables updated based on updates it receives. Upon request, this data is delivered to the applicationon host PC, where it is later used for graph formation.

The key function of coordinator is to detect node isolation. It maintains timestamps for each node inm the network. It updates these timestamps for each node when it receives data. Typically, a functional sensor node will send sensor data perdiodically to the coordinator. If the coordinator sees timeout for any node, it conludes that the node has been isolated and reports the host PC application accordingly.

- **Issues in Implementation:**

A major issue was single-packet buffer in Rime API of Contiki. Also, the buffer is shared for all channels and for best-effort / reliable sending and broadcasting. We solved this issue by waiting for randmo amount of time before sending the data. This is useful when the nodes are sending data perdiocally. The packet collision in Rime buffer would result in unicast timeouts and the node would mark the next-hop as failed. With random waits before sendiing, such false-detection of failure is avoided. The single-packet issue called for reliable unicast as against best-effort unicast. This results in increased network traffic and more battery power consumed.

3.2 Data Mule

The data mule is basically a mobile unit which will carry the mote around the network to bridge the broken connections or collect data from isolated nodes.

- **Design: Mechanical and Code**

The data mule was built with the Lego RCX Mindstorm kit. The role of the data mule is to go to the isolated node and bridge the network. For this the rcx will track a line till it comes to the appropriate node. The location of nodes are marked by markers and the order of nodes is known to the PC application beforehand. The PC application will send the proper number of the node to the rcx which

will then go the required position to bridge the network.

- **Issues**

Since the Data Mule is developed with Lego Mindstorm, its functionality is quite limited in certain extents. For example it can only trace lines and cannot move freely around a given area. Also once it leaves the infrared range of the PC tower there is no way to communicate with the data mule. With a sophisticated data mule PC application could communicate with the rcx through the nodes in the

3.3 Host PC Application

The host PC application resides on the coordinator PC which is connected to one mote. The mote will update the PC application with the updates in the sensor network. The PC application on receiving the updates will maintain a graph of the network. Whenever a node fails, the PC application will mark the node as failed and do a Depth-First Search from the root node, discarding the failed node when it comes across it. After the depth first search it will check the graph for any unvisited and unfailed nodes. These nodes are the ones which have been isolated because of failure of some node. It will then signal the rcx with the position of the first found isolated node(since we have only one rcx) and dispatch it to go and either recover the node or bridge the network.

4 Simulation

4.1 Cooja

The developers of Contiki provide a Java-based simulation tool named Cooja. It has following features

- Java-based GUI
- Emulation mode support for Sky motes
- Emulates radio transmission and interference ranges: allowed us to test failure and re-routing capability
- Log listener to monitor debug messages
- Traffic Visualizer: allowed us to detect the “single Rime buffer” issue

5 Open Issues

1. When a router receives advertisement, it can associate received signal strength with this router and keep the routing table in sorted order of received signal strengths. This will allow the sender to send data over more reliable path. The same can be achieved by sending battery voltage while advertizing and then keeping the routing table in sorted order of battery voltages. The latter approach associates higher battery voltage with more reliability
2. The data mule can be constructed with sophisticated hardware with Zigbee capability so that it can directly talk with the sensor nodes. It can also have GPS feature to locate itself in the network.
3. There can be multiple coordinator nodes, providing hardware redundancy at coordinator level.
4. Contiki features IPv6. We can investigate this use of TCP/IP stack to see if it has any advantages as compared to Rime stack based on features of TCP itself.
5. We can investigate the scalability of this algorithm, specifically considering the timing issues and size of routing tables on sensor nodes.
