A Simple Event Detection System for Wireless Sensor Networks

Master's Project Final Report

By

Kyu Wook Cho

Candidate for Degree of Master of Science in the Department of Computer Science at Brown University

PROVIDENCE, RHODE ISLAND
MAY 2006

This project report by Kyu Wook Cho is accepted in its present form

by the Department of Computer Science as satisfying the

project option requirements for the degree of Master of Science

Date _____ _____

Ugur Cetintemel, Advisor

ACKNOWLEDGEMENTS

I would like to thank first my advisor, Professor Ugur Cetintemel, for his guide and encouragement. My heartful thanks next go to Jehyok Ryu and Mert Akdere for their comments and valuable discussion on this report. I truely thank Mee Jin Cho for everything.

Lastly, I would like to add a special note of appreciation to my parents, Wook Hwan Cho and Soon Ja Lee. Without their love and warm support, I would not be able to come this far. My younger brother, Kyu Tae Cho, deserves my thank for being with our parents when I was unable to be with them.

Contents

1 Introduction **1**

2 Basic Event Detection Model **2**
2.1 Primitive Event Specification 2
2.2 Complex Event Specification 3

3 Implementation **5**
3.1 Routing 5
3.2 Time Synchronization 6
3.3 Event Detection 8
3.4 Visualizer 9

4 User Guide **11**
4.1 System Installation 12
4.2 System Usage 12

5 Open Issues **16**
5.1 Limitations and Future Work 16

6 Conclusion **18**

Bibliography **21**

Chapter 1

Introduction

With the advanced technologies in small devices and wireless communication, sensor nodes are used to perform various tasks by forming network among themselves. Examples are following: life scientists have used sensor networks to observe the ecological patterns and habitats without disturbing animals [1]; The sensor networks for the medical care are used to keep track of patient blood pressure and heart beating [2].

In the Wireless Sensor Networks, event detection is one of the main reasons to use sensor networks. An event is defined as an "occurrence of significance" [5] in a system. In turn, "occurrence of significance" can be interpreted as a result of a single predicate which yields that sensors are designed to report a particular event that is interesting to a user. A complex event is a combination of several single events. Also, a combination of complex event is called a complex event. To have the system to detect complex events will allow users to analyze accurately the current situation. However, sensors have limited capability and resources such as computing power, memory, bandwidth, energy, and etc. All of these constrains make event detection challenging in the Wireless Sensor Networks. I focus here on how to minimize the use of energy to process and report the satisfying events no matter what the event is simple or combined. For this purpose, I present the initial centralized version of system for complex (or combined) event detection in this project report.

The remainder of this report is organized as follows. In Chapter 2, I describe the simple event detection model and language. In Chapter 3, the design and algorithm about routing, time synchronization, event detection, and visualizer for the system is described in detail. In Chapter 4, I provide the user guide for the system. In Chapter 5, I discuss the limitation of current system and future work. I conclude in Chapter 6.

Chapter 2

Basic Event Detection Model

In order to specify the complex events, the event specification language should be highly expressive. In this chapter, I describe event specification language.

2.1 Primitive Event Specification

I define a primitive event as an atomic occurrence of an event of user's significance or interest. Primitive events are represented by binary values that mean the events either happened or not-happened. Each primitive event has an event type. An event type provides the metadata information for events of that type. Finally, complex events are those events that result from the application of event language operators to simpler event types.

Event	<event_name>
Scheme	id, t, loc, attr
Where	<predicate_list>

Each primitive event specification will follow the above format. <event_name> can be referred in other event specification using this name. At the second line, id is the node id that detects the event. t is the timestamp when the event occurred. loc is the location(region) where the event happened. attr is the attribute which is used in the primitive event such as temperature and light. For where part, we are trying to give some flexibility to user for setting predicates. Followings are the examples that can be used for <predicate_list>.

- Spatial Predicates – region = R1
- Temporal Predicates – runtime 9am to 10am
- Attribute Predicates – temp > 100 or light > 400

For a better understanding, I first define the primitive event specification and next present the possible primitive event specification examples.

Event	hi_temp
Scheme	id, t, node_region, temp
Where	temp>100

Event	person_detected
Scheme	id, t, node_region, person_id

Event	book_detected
Scheme	id, t, node_region, book_id

Event	lights_on
Scheme	id, t, R2, light
Where	light>200 and region=R2

2.2 Complex Event Specification

Defining complex event is different from the primitive event specification. The template for complex event specification is as follows:

Event	<event_name>
Expr	<event_expression>
Scheme	id, t, loc, attr
Where	<predicate_list>
Within	<window>
Latency	<allowed_latency>

The above template is almost same as the primitive event specification. However, complex events should be expressive enough to support users' various desire for defining their interest. Furthermore, some of the fields in the template must be extended to hold more information, enabling the template to handle multiple primitive event specifications. Expr <event_expression> is the pattern of the complex event. For instance, if a complex event that combines three primitive event specifications such as E1, E2, and E3 occurs, then the event expression could look like $E1 \wedge (E2 \vee E3)$. Unlike the primitive event specification, the Scheme field for complex event specification has a broader meaning. id field is not necessarily generated by a single node and the node id is not necessarily well-ordered. t = (t1, t2), where t1 represents the start time of the complex event and t2 represents the end time of the event, is set by the system and is accessible to the user. The loc field for complex event specification does not necessarily occur at a single node. That is, events can occur at the

multiple locations. attr is the concatenation of attributes of constituent events. Within is a time interval where events happened within the same window defined in <window>. Latency is set by user such that the events that happened at time t with latency Δt will be allowed only if Δt is less than <allowed_latency>.

Chapter 3

Implementation

The system is split up into two main components, the sensor and the visualizer components. The sensor component is also divided into several modules. Each module is responsible to control the behavior of sensor.

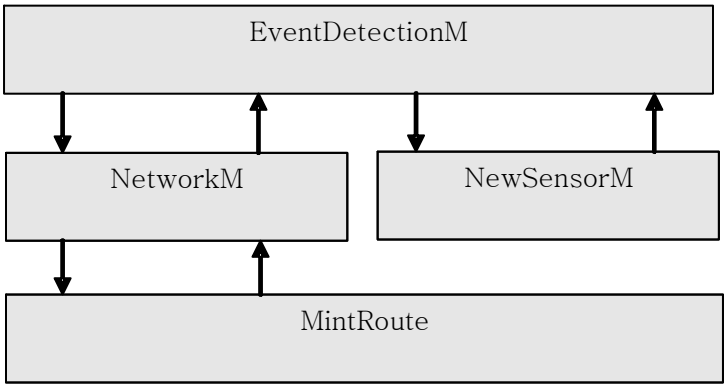


Figure 3.1: **System Layers for Sensors**: EventDetectionM controls system; NetworkM is responsible for general communication; NewSensorM gets sensor readings; MintRoute constructs routing tree

The visualizer sits on the base station and interacts with all the sensors running the sensor component. The initial system starts with a centralized version, which means that all the data are delivered to the base station and predicate checking is processed at the base station as well. I shall discuss specific modules within the sensor and visualizer components.

3.1 Routing

The network module provides the communication with other nodes in the network. It also controls underlying routing layer algorithm called MintRoute which is described in [3]. The network module

is responsible to receive and send messages such as *EventDataMsg* and *EventCmdMsg*. Depending on which message the network module processes, it uses two different routing schemes.

Message like *EventDataMsg* will be delivered to the base station. To deliver data from network, I use a tree-based multi-hop routing for the initial centralized version. The multi-hop tree network is the most common routing scheme used in the Wireless Sensor Networks. The multi-hop tree routing scheme is robust for data delivery. It is proved by the numerous previous applications in TinyOS sources. When the system starts, every node in the network forms a multi-hop tree network using MintRoute developed at U.C Berkeley. MintRoute is a proactive routing algorithm. It is based on the route discovery message exchanges and accumulates neighbor nodes' information. Assume that there exist several nodes. They construct a tree network. A node 'C' finds its level of the network. Once enough information is gathered at the node, a node 'C' decides its level of the tree in the network by using cost metric with the combination of hop count from the base station and link quality. The best node is then selected as parent node 'P'. All the data messages from the current node is handled by parent node only and forwarded to the base station. If one of the neighbor nodes of the node 'C' listens to the data message, it can then snoop the data message to process some in-network data processing. However, the current system lets the neighbor nodes ignore the message from the node 'C' if they are not set as a node C's parent node. For some reason, either if a node 'P' suddenly disappears from the network tree or if a node 'P' moves from its current location, the node 'C' will pick the second best node in the neighbor table as its new parent node by the cost metric function. With this routing layer protocol, the system guarantees data message delivery to the base station. In addition, a new node can join the network by exchanging the route discovery message depending on its distance and location.

Messages like *EventCmdMsg,* generated by users, must be disseminated across all the nodes (destined by users) within the network. In the system, I use flooding routing scheme for command message dissemination. The flooding routing scheme is the most simple and easy to implement. To avoid infinite message broadcasting, each command message includes sequence number and processing command identification number in its header field. Upon receiving command message, the network module will send a signal to main control module, called *EventDetectionM*, in order to process command as specified by users. The issue of how the system handles and processes command message will be discussed in Section 3.3 .

3.2 Time Synchronization

Time synchronization is essential due to the event ordering. Assume that an event E1 happened earlier than an event E2, but E2 arrived at the base station earlier than E1 caused by network delay. Without time synchronization, a user may conclude that an event E2 happened earlier than E1. It causes an error if what a user wants is time crucial events.

It is easy to figure out which event happened first by comparing one event with another event in a single node. However, it becomes difficult if there exist multiple nodes in the network. This is because each node in the network has its own hardware clock. Since the purpose of this project is to provide accurate and semantic data, it is needed to provide a plausible way to order each event with its time when it actually happened. To address this concern, all the nodes in the network needs to be time synchronized. There are several ways to implement time synchronization, particularly, in the Wireless Sensor Networks. The problem is to find the algorithm which fits the current multi-hop routing scheme within the system. The TPSN [6] algorithm that developed at UCLA would be a perfect match for the current system. The model of TPSN is exactly same as the routing scheme of the system: The TPSN constructs hierarchical tree network first and then a pair wise synchronization is performed along the edges of its tree structure.

When the system begins by the routing algorithm described in Section 3.1, every node in the network forms hierarchical tree network. Initially, each node's state is set to NOT_SYNCED except the gateway node whose local time will be used to synchronize every node in the network. Once a child node decides its parent in its neighbor table, the node is ready to synchronize its time to parent's time.

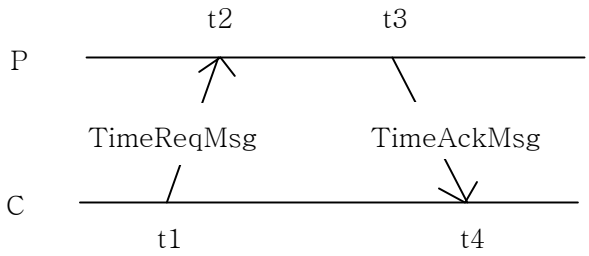


Figure 3.2: **Simple Pairwise Time Synchronize Model**

Once a node C decides its parent node to be a node P, it is ready for time synchronization. As depicted in Figure 3.2, a node C will broadcast its *TimeReqMsg*, including the timestamps when the message sent (t1), destined to a node P and waits for the *TimeAckMsg* from a node P. A node P receives the *TimeReqMsg* and at the same time it records the reception time (t2). After the internal processes, a node P will broadcast *TimeAckMsg* including the time t1, t2, and the time a node P broadcast acknowledge message (t3) to a node C. The node C received *TimeAckMsg* from a node P. At the same time, a node C will record the reception time (t4).

$$\Delta = \frac{(t2 - t1) - (t4 - t3)}{2}$$

Now, a node C has all necessary information to calculate the drift between parent (node P) and child node (node C) using above equation from [6]. The two nodes will be synchronized by adding

Δ to node C's local time. Since a parent node can have multiple nodes as its children, the system should be able to synchronize multiple nodes efficiently. The child node selects its parent among the neighbors, meaning that they share the radio range. If a parent node sends the *TimeAckMsg* to a specific node which requests time synchronization and such a node can only perform time synchronization process, then it wastes precious resource of the sensor node. While on the other hand, in the system the nodes are allowed to snoop the *TimeAckMsg* and synchronize their local time although they do not issue the *TimeReqMsg* to their parent. However, skews will occur when time passes by. Therefore, the system performs periodical time synchronize message exchanges with its parent node to maintain the system synchronized.

3.3 Event Detection

From previous discussion, it should be clear to users how the routing and time synchronization work in the current system. As I have described my event specification language and examples in Chapter 2, I will describe how the system attempts to perform event detection processes. The initial system implementation is a centralized version. It means that all the events are collected to the base station and then, the base station notifies only the data that a user has interests.

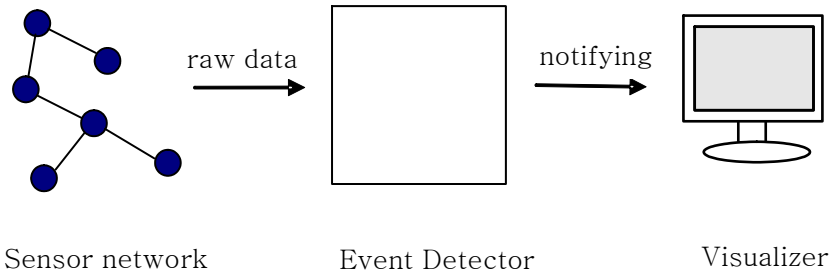


Figure 3.3: **Event Detection Process Abstrat Model**

When a sensor node in the network receives the event specification, the node will execute the proper routine based on the event specification. With a simple example below, I describe how the system detects an event of a user's interest.

Event	lights_on
Scheme	id, t, R2, light
Where	light>200 and region = R2

This event specification is made by user and disseminated to only the sensor nodes that belong to the region R2. Since the raw data keep coming to the base station, the base station will apply its

predicates to receiving events. After applying the event detection operator to every event, only the events with light greater than 200 from region R2 (which is pre-defined above) will be notified to user.

The result of primitive event specifications will be stored and used for complex event specification. Assume that there are the result of two primitive event specifications, E1 and E2. The user defines his interest in E1∧E2. Only the event satisfying the complex event specification (i.e., E1∧E2) will be notified to a user. So far, the system supports a couple of operators such as ∨ and ∧. The extension for more operators and detailed complex event specification syntax remains for future work.

3.4 Visualizer

The visualizer allows users to interact with and interpret the processes of the sensor networks. The resulting hierarchical network tree structure and the data flow to base station help users to understand the system behaviors. The current visualizer includes many features from previous project, DorothyViz, by Bryant Ng [7]. Figure 3.4 illustrates how the network tree is constructed as well as how the data flow in the network. The node attached to the base station is the gateway node. It is also the sink node that collects every data from the nodes in the network. In figure, the green line indicates that node 1's parent is node 2 and that node 2's parent is node 0. The data generated from each node will flow along the edge of the green line. When the sensor broadcasts a message, the sensor node is outlined with red circle. The user can construct a command message controlling the behavior of each node. Pressing the send command message button will bring up an additional frame to show user the resulting events from network. It enables network to have multiple event specifications running simultaneously.

However, users should be aware that the displayed result in the visualizer is shown only when the base station received the messages forwarded by the gateway node. It means that the displayed result in the visualizer actually happened at the node before it appears. Since the system supports the time synchronization, users can estimate when the events actually happened and order such events.

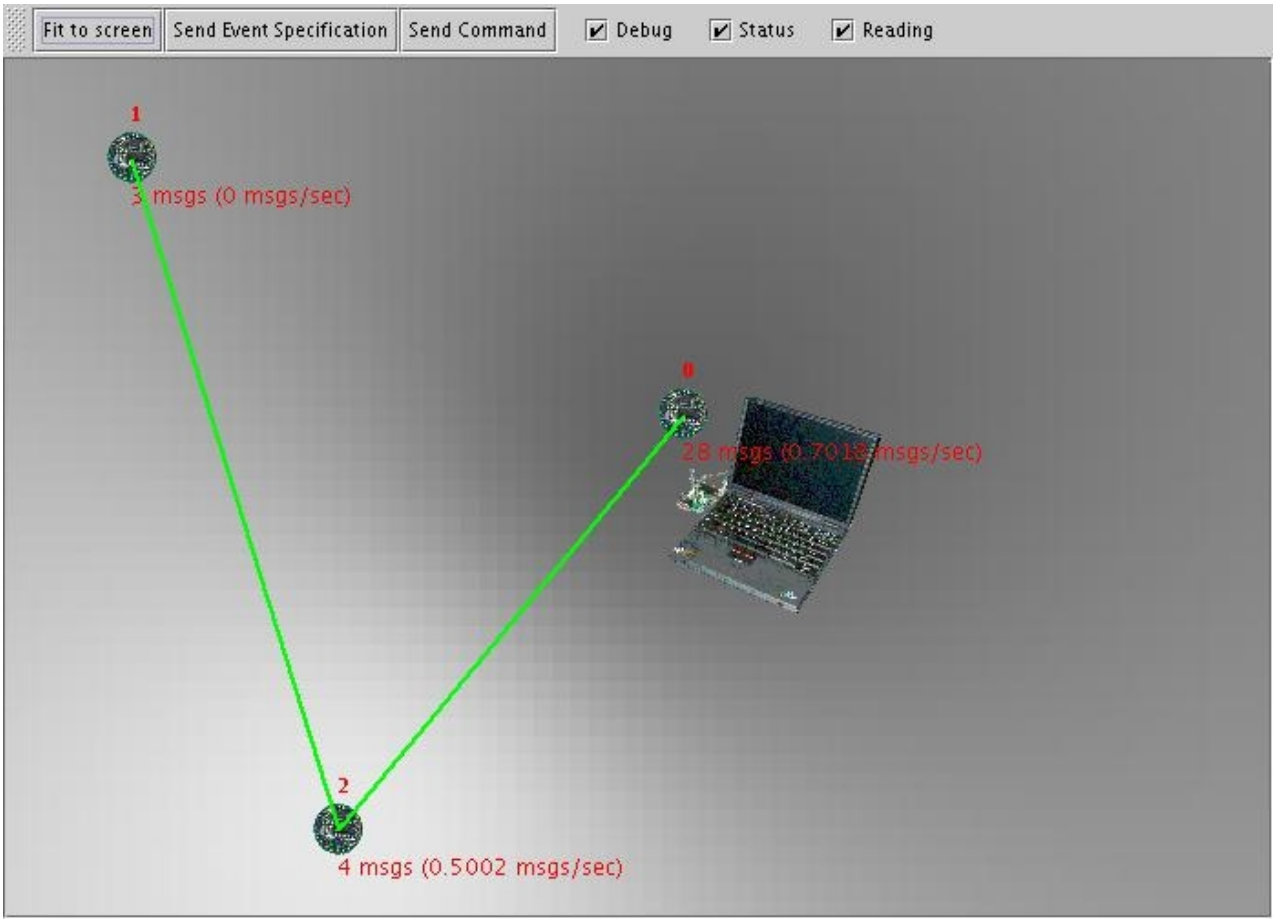


Figure 3.4: **Visualizer**: A simple multi-hop network with three nodes: Node 0 is the gateway node that forwards messages to the base station and broadcasts command messages from the base station to network

Chapter 4

User Guide

Before I explain how the users install the system, I shall mention here what operating system is used and which type of sensor platform is used for this project. The system uses the Berkeley Motes (in Figure 4.1) and specifically, the mica mote platforms. The size of mica sensors is approximately 2cm x 4cm x 1cm. The mica motes are equipped with a 917MHz RFM radio running at 40 KB per second, a 4MHz Atmel microprocessor, 4KB RAM and 128 KB code space, 512 KB EEPROM, "micasb" sensor board that has sensors such as light, temperature, acceleration and sound, two AA battery as power source.

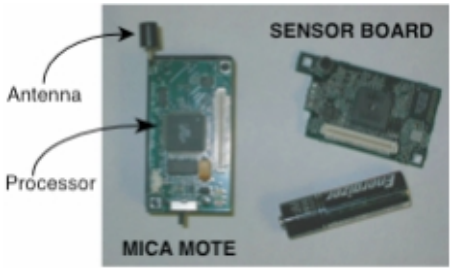


Figure 4.1: **Berkeley Mica Mote and micasb Sensor Board**

The operating system that runs on the mica mote platform is TinyOS [8, 9]. It is a real-time component-based event-driven embedded operating system. The TinyOS is structured into modules (components) and applications programmed by "wiring" several relevant modules. Since concurrency is one of the most important factors in the Wireless Sensor Networks, TinyOS achieves concurrency by using split-phase fashion.

4.1 System Installation

To use the system, TinyOS should be installed to users' computers either Linux or Windows. The detailed installation information is available on the TinyOS website [9].

The system that runs on sensor nodes are located at */apps/EventDetction/src/eventDetction* directory. The header file that requires to compile and run the system is located at */apps/EventDetection /src/common* directory. As I mentioned before, the system that runs on sensor is a combination of several layers. The routing module files are located at */tos/lib/MintRoute* directory. All files and directories start off from the TinyOS root directory that is *tinyos-1.x*. Make sure all the files are in proper location.

To install the program to sensor nodes, go to the directory *tinyos-1.x/apps/EventDetection/src/ eventDetection* and type *make mica*. This will compile the program for Mica platform. Connect you Mica mote to program board and power cable to program board as well. Once you connect correctly everything, you are ready to install software to sensor nodes. Type *make mica reinstall.#*, where # is the unique identifier of the node. Node number 0 is assigned as the gateway node. Since the gateway node should communicate with base station, users should be careful and assign proper node identification numbers which is always zero. I do not place any restriction on the number of nodes in the network. However, more nodes will make network busy and packets may be lost during system running. I tested the system with mica motes, but there should not be a problem to install the system onto mica2 or mica2dot platforms.

The visualizer for the system is placed under */tinyos-1.x/tools/java/net/tinyos/EventDetectionViz* directory. To compile the visualizer, users just need to type *make*. The visualizer can communicate with mica motes with the message types such as *EventDataMsg* and *EventCmdMsg* defined in the sensor node software directory. When users type *make* for visualizer compilation, those message type definitions are processed and converted to Java format by a program called *mig,* which is a tool generating a code that processes TinyOS messages.

4.2 System Usage

Once the programs for sensor nodes and visualizer are correctly installed, users are ready to deploy and retrieve data. First, turn on the sensor nodes after installing the program for sensor nodes. Second, open a new terminal and go to TinyOS Java directory by typing *cd tinyos-1.x/tools/java*. Then, type *java net.tinyos.EventDetectionViz.MainClass*. This will start up the visualizer. Initially, in the visualizer users can see one sensor node connected to the base station. The user should give a couple of minutes for the network to construct hierarchical tree among sensor nodes in the network. This is because MintRoute needs to gather neighbor link information by exchanging messages.

After one minute or more than one minute, users should be able to see the tree network as in Figure 3.4.

The LED lights of sensor show the current behaviors occurring at the sensor node. The red LED light toggling means that sensor reading value is ready and sensor broadcasts the *EventDataMsg* packet. The green LED light toggling means that sensor node received an *EventCmdMsg* and need to process the command message. The yellow LED light toggling means that the sensor node received *EventCmdMsg* and the sensor node needs to log the sensor reading value along with the timestamp to its memory space.

Initially, the visualizer shows every packet arrival to the base station since DEBUG check box is checked on. Turning off the DEBUG check box stops showing the packets to the users. The user can construct an *EventCmdMsg* that controls the behaviors of sensor nodes by clicking *Send Command* button. When the button is clicked, a dialog box brings up like Figure 4.2. Clicking *Send Command!!* button will show the result from the network. Figure 4.3 is the example result after the user sent the constructed command message.

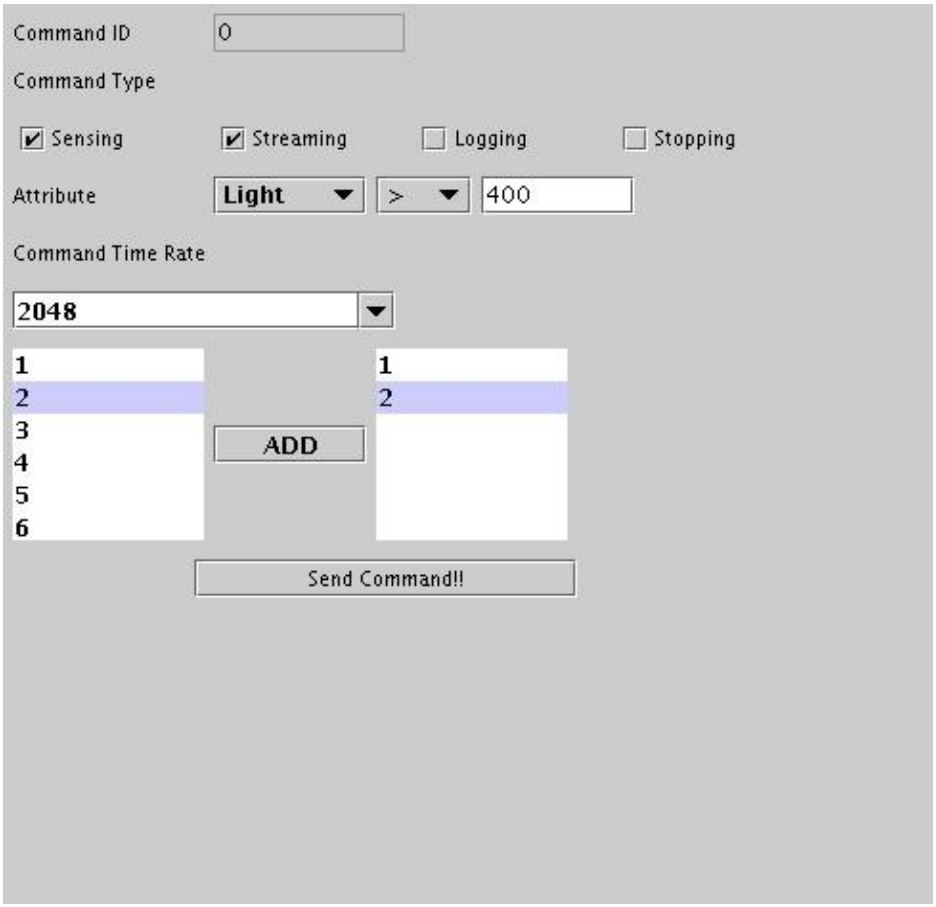
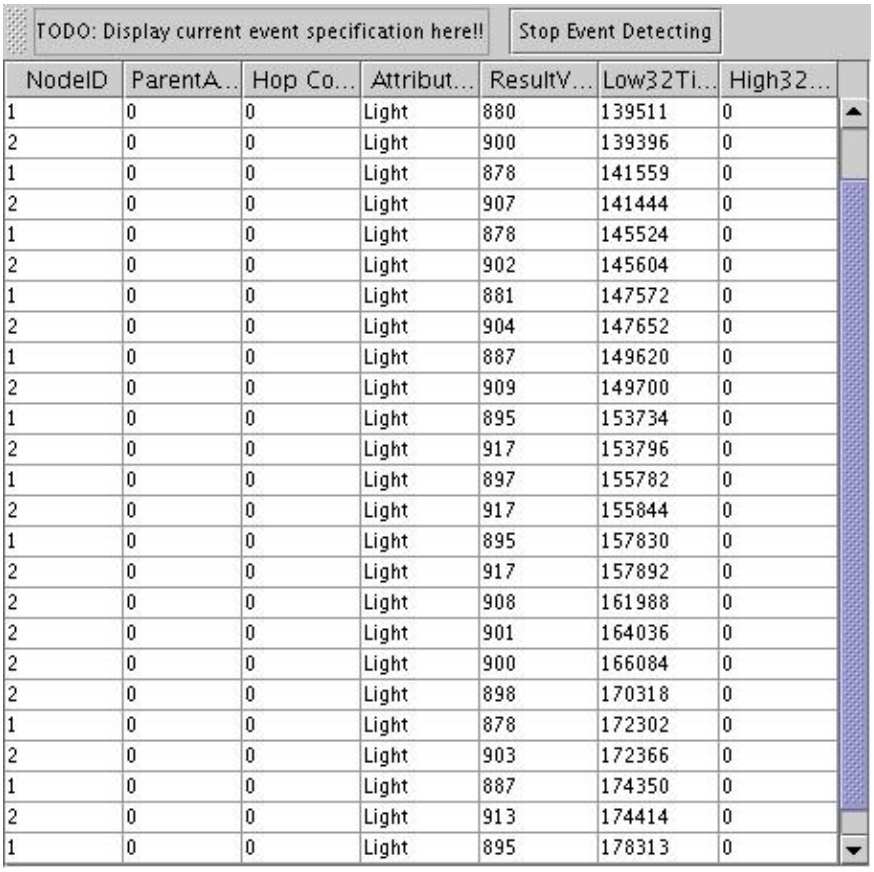


Figure 4.2: **Dialog Box to Construct EventCmdMsg**: Command ID is unique and tells how many command messages have been sent over the network; Command type will set the flag that controls the behavior of sensor node; The user can set attribute field and predicate; Further, users can set the destination where the command message should take place



NodeID	ParentA..	Hop Co...	Attribut...	ResultV...	Low32Ti...	High32...
1	0	0	Light	880	139511	0
2	0	0	Light	900	139396	0
1	0	0	Light	878	141559	0
2	0	0	Light	907	141444	0
1	0	0	Light	878	145524	0
2	0	0	Light	902	145604	0
1	0	0	Light	881	147572	0
2	0	0	Light	904	147652	0
1	0	0	Light	887	149620	0
2	0	0	Light	909	149700	0
1	0	0	Light	895	153734	0
2	0	0	Light	917	153796	0
1	0	0	Light	897	155782	0
2	0	0	Light	917	155844	0
1	0	0	Light	895	157830	0
2	0	0	Light	917	157892	0
2	0	0	Light	908	161988	0
2	0	0	Light	901	164036	0
2	0	0	Light	900	166084	0
2	0	0	Light	898	170318	0
1	0	0	Light	878	172302	0
2	0	0	Light	903	172366	0
1	0	0	Light	887	174350	0
2	0	0	Light	913	174414	0
1	0	0	Light	895	178313	0

Figure 4.3: **Displaying Resulting Data Arrival to the Base Station**: The last two columns are timestamp values that show when the message has been generated and network is time synchronized

Chapter 5

Open Issues

I have described the initial version of a simple event detection system. Users would understand how the sensor nodes communicate each other, how the nodes are time synchronized, and how the sensor networks detect events. However, there are a few limitations to the current system and future work remains.

5.1 Limitations and Future Work

As of the current version of the system, users should wait for a couple of minutes until nodes in the network construct a hierarchical tree network. It is due to the algorithm of routing layer to gather enough information to decide its level of the tree network. Users might be able to reduce the waiting time by changing the threshold for neighbor link state information. Another limitation is from the visualizer. The link connections between nodes are displayed only when there are messages from network. When the user wants to log the data to nodes' memory, there is no message delivery to the base station. It is the case where the links in the visualizer disappears. So, the user may analyze incorrectly that the current network links are broken. In fact, those links are not broken. They will be displayed if there is any message delivery initiated by a new event specification.

The current version of the system supports primitive event specification and a couple of complex event specification operators. The future work for the system therefore remains to define the fully functional complex event specification. Once it becomes available to the future system, the system should be able to collect the result of each primitive event specification and report the final events after applying proper operators. To provide better scalability and to avoid limitations of most centralized system, the current system has to be extended to support decentralized architecture as well. Distributing efficiently event detectors to networks would provide better performance and consume much less resources. However, it is difficult to decide how to distribute event detectors. An interesting extension for the system would be to have 'cost calculation mechanism' to detect primitive events based on the amount of energy consumption. Intuition for this idea is following.

Suppose that a user is interested in detecting the events of E1∧E2 and also interested in the events of E2∨E3. In this situation, detecting event of E2 specification first would cost less than otherwise.

Chapter 6

Conclusion

I built the initial system using the Berkeley Mica Mote platform running on TinyOS. Currently, the Mica mote is out of production. However, the system can be used to operate on Mica2 or Mica2dot platforms without any problem. This is because they are the mica family and they share the same code and runs on same TinyOS.

In the system, the simple event detection routines processes in a robust fashion. Further, the system provides a good start using sensor networks to detect events. Although the initial version has not fully incorporated with complex event specification language, the system is groundwork for future development to embody complex event specification language in the system.

Appendix

Message Structures used in this report

```
struct EventDataMsg {
    uint8_t     type;          // attribute type
    uint16_t    parentaddr;    // parent node address
    uint16_t    resultdata;    // sensor reading value
    GTime       timestamp;     // time of the event happened
}

struct EventCmdMsg {
    uint16_t    seqno;                          // sequence number
    uint8_t     commandtype;                    // type of command message
    uint8_t     attributename;                  // attribute type
    uint32_t    startinginterval;
    uint32_t    endinginterval;
    uint32_t    newrate;                        // sampling rate
    uint16_t    nodelist[MAX_NODE_NUM];
}


struct TimeReqMsg {
    uint16_t    sourceaddr;    // requestor's address
    uint16_t    parentaddr;    // parent node address
    GTime       sentTime;      // time of TimeReqMsg sent
}
```

```
struct   TimeAckMsg   {
         uint16_t     sourceaddr;   // parent node address
         uint16_t     reqfrom;      // child node address
         GTime        sentTime;     // original TimeReqMsg sentTime
         GTime        recvTime;     // time of reception TimeReqMsg
         GTime        ackTime;      // time of TimeAckMsg sent
}
```

Bibliography

[1] A. Mainwaring, J. Polastre, R. Szewcyzk, D. Culler, and J. Anderson, Wireless sensor networks for habitat monitoring, *ACM International Workshop on Wireless Sensor Networks and Applications*, September 2002.

[2] D. Malan, T. Fulford-Jones, M. Welsh, and S. Moulton, CodeBlue: An Ad Hoc Sensor Network Infrastructure for Emergency Medical Care, *International Workshop on Wearable and Implantable Body Sensor Networks*, April 2004.

[3] A. Woo, T. Tong, and D. Culler, Taming the underlying challenges of reliable multihop routing in sensor networks, *Proc. the First ACM Conference on Embedded Networked Sensor Systems (SenSys 2003)*, November 2003.

[4] V. Shnayder and other, Simulating the Power Consumption of Large Scale Sensor Network Applications, *Proc. of SenSys 2004*, to be published

[5] D. C. Luckham, *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*, Addison-Wesly Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[6] S. Ganeriwal, R. Kumar, and M. B. Srivastava, Timing-Sync Protocol for Sensor Networks, *The First ACM Conference on Embedded Networked Sensor System (SenSys)*, p. 138-149, November 2003.

[7] B. Ng, Data Flow Processing in Sensor Networks, Master thesis, May 2004.

[8] J. Hill, R Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, System architecture directions for networked sensors, *ASPLOS*, November 2000.

[9] TinyOS, A component-based OS for networked sensor regime, http://www.tinyos.net/