Multihop Routing Simulation of TinyOS-Based Wireless Sensor Networks in Viptos

Heather Taylor
Electrical Computer Engineering
University of Vermont
hltaylor@uvm.edu

Graduate Mentor: **Elaine Cheong**
Research Supervisor: **Dr. Jonathan Sprinkle**
Faculty Mentor: **Prof. S. Shankar Sastry**

August 4, 2006

Summer Undergraduate Program in
Engineering Research at Berkeley (SUPERB) 2006

Department of Electrical Engineering and Computer Sciences
College of Engineering
University of California, Berkeley

Multihop Routing Simulation of TinyOS-Based Wireless Sensor Networks in Viptos

Heather Taylor

Abstract

Wireless Sensor Networks are a burgeoning area of research and applications in embedded systems. The purpose of this project is to understand and further develop Viptos (Visual Ptolemy and TinyOS), an integrated graphical development and simulation environment for TinyOS-based wireless sensor networks. TinyOS is the operating system for the Berkeley Motes, which are small embedded systems capable of collecting audio, temperature and other kinds of sensor data and transmitting it via a radio. A TinyOS simulator called TOSSIM is used, a key piece of which is the ability to simulate a network topology once it is functioning. Viptos extends the capabilities of the TinyOS simulator to allow simulation of heterogeneous networks. The final goal of this research is to create a graphical representation of the communication between motes in a multihop network simulation in Viptos. This will be done by adding an entity to Viptos which will collect transmitted information and display a graphical representation of that communication between nodes.

1 Background

1.1 Wireless Sensor Networks

Wireless Sensor Networks (WSNs) are a growing field in embedded systems applications and research. Wireless motes are circuits which are capable of collecting data from their various sensors. They are able to transmit this information in the form of a packet via an on board radio transmitter to other motes within the network. This packet is a collection of information often containing collected data as well as network specific information such as source and destination nodes, or a timestamp. A multihop network is formed when a packet transmitted by a mote can be received by a neighbor within the network that is not necessarily the base node. This neighbor then sends the packet to the next closest mote until it reaches the base station. WSNs that are set up to behave in a multihop fashion are more dynamic than single hop networks in that they can be dispersed over larger areas. In a single hop formation, each mote must be located within radio range of the base node because it must communicate directly, whereas multihop formations allow motes to be more physically separated as they only need to be in contact with neighbor mote, as opposed to the base node.

1.2 Software

1.2.1 TinyOS

TinyOS is an open-source operating system which takes into account the severe memory constraints present in wireless embedded sensor networks. Its component-based architecture provides a library which allows the user to easily implement or build his/her own custom application while minimizing code size. TinyOS also allows fine-grained power management as part of its event-driven execution model, as well as the scheduling flexibility necessary for the unpredictability of wireless communication in the physical environment. [2]

1.2.2 Ptolemy II

Ptolemy II is a Java-based component assembly software framework created as part of the Ptolemy Project. It includes a graphical user interface (GUI) called Vergil, which allows users to interact with the simulation on a graphical level. The Ptolemy Project studies modeling, simulation and design of concurrent, real-time, embedded systems. It focuses on the assembly of concurrent components, with an emphasis on the use of well-defined models of computation. These models control the interaction between components, which allows users to address the problem of heterogeneous mixtures of models of computation.

Ptolemy II contains a growing library of domains, where each domain realizes a different model of computation. The components within the library are mostly domain polymorphic, in that they can operate in several domains. In addition many components are data polymorphic, and are able to operate on several data types. [1]

1.2.3 TOSSIM

TOSSIM is an interrupt-level simulator for TinyOS wireless sensor networks. It replicates the behavior of a WSN with fine-grained accuracy for thousands of nodes by using a probabilistic bit error model. It runs actual TinyOS code, using software replacements for the hardware and models the network interaction. TOSSIM is simple and efficient, yet can capture a wide range of interactions within a simulated network.

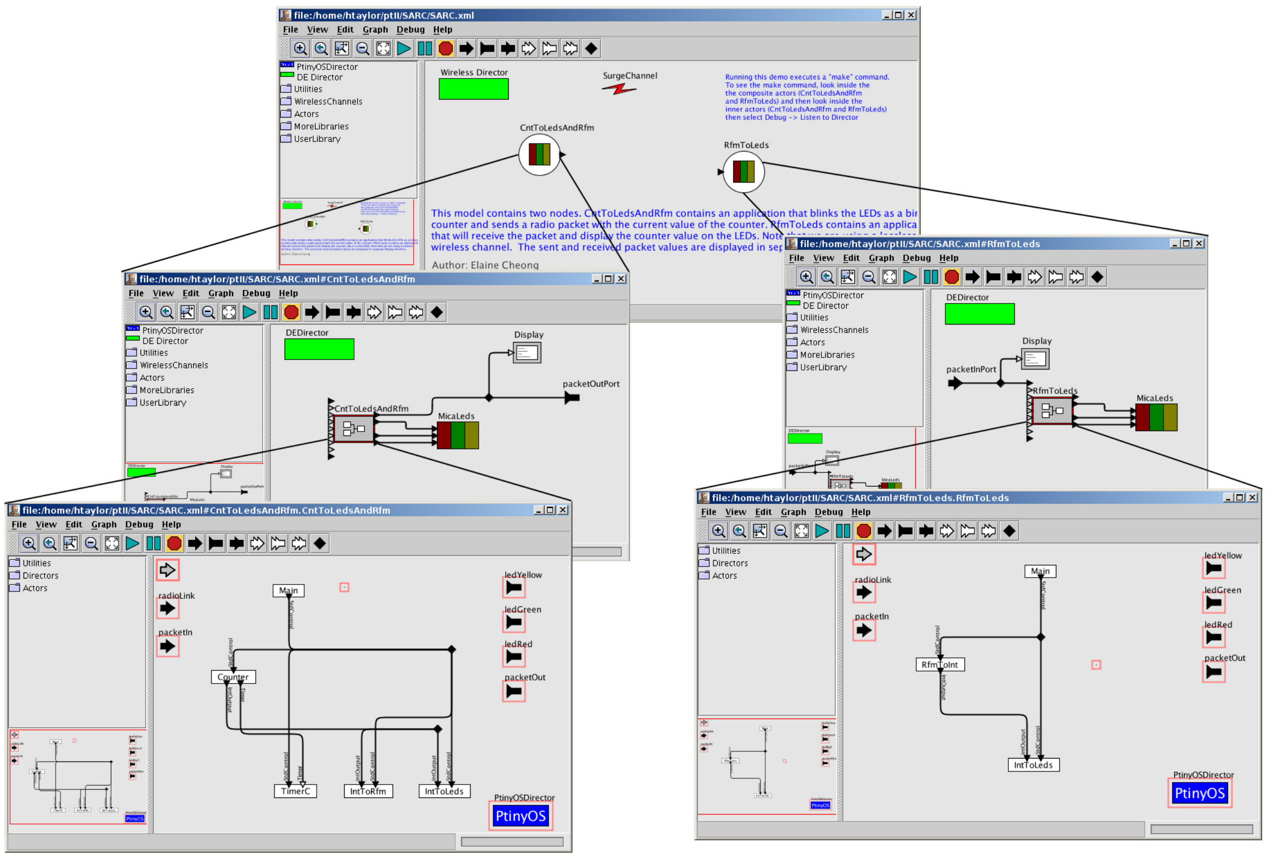
TOSSIM is often used in conjunction with a GUI called TinyViz which allows the user to view and interact with a graphical interpretation of the simulation. TinyViz includes different tools which the user can utilize to view different aspects of the simulation such as debug messages, packets sent, and radio links among motes (see Fig. 5).[6]

1.2.4 Viptos

Viptos (Visual Ptolemy and TinyOS) is an integrated graphical development and simulation environment for TinyOS-based WSNs. The Viptos development environment allows users to create TinyOS programs from block and arrow diagrams of nesC/TinyOS components (see Fig. 1). This diagram can then be converted to a nesC program and compiled and downloaded onto TinyOS-supported hardware from within Viptos.[4] Viptos also contains the capabilities of VisualSense, allowing it to model communication channels, networks and non-TinyOS hardware.

Viptos bridges the gap of VisualSense and TOSSIM by providing interrupt-level simulation of real TinyOS programs, including packet-level simulation of a network, while at the same time granting the use of models of computation only available in Ptolemy II, allowing users to model various parts of a system. TOSSIM itself only allows the simulation of homogeneous networks, whereas Viptos can simulate non-TinyOS, heterogeneous networks. It also allows users to switch channel models and easily change other parts of the environment.

The actor-oriented modeling environment of Viptos is inherited from Ptolemy II, and it allows the different models of computation at each level of simulation to be used by developers. At its lowest level, the discrete-event scheduler of TOSSIM is used by Viptos to model the interaction between a simulated mote CPU and the TinyOS code. At the level above this, the discrete-event scheduler of Ptolemy II is used to model inputs to mote hardware, or the mote-hardware interface. This is then embedded in VisualSense so that the user can


Running this demo executes a "make" command. To see the make command, look inside the the composite actors (CntToLedsAndRfm and RfmToLeds) and then look inside the inner actors (CntToLedsAndRfm and RfmToLeds) then select Debug -> Listen to Director

This model contains two nodes. CntToLedsAndRfm contains an application that blinks the LEDs as a bir counter and sends a radio packet with the current value of the counter. RfmToLeds contains an applica that will receive the packet and display the counter value on the LEDs. Note wireless channel. The sent and received packet values are displayed in sep

Author: Elaine Cheong

Figure 1. Different levels of Viptos

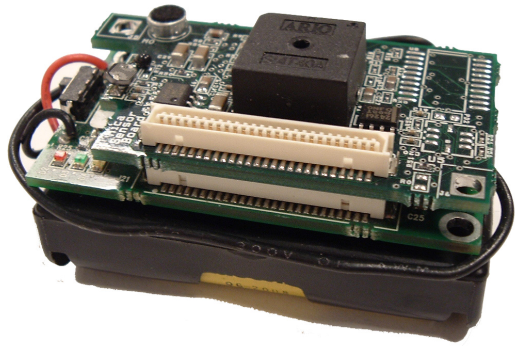


Figure 2. Mica2 Mote

model wireless channels which will allow simulation of elements inherent in wireless communications such as packet loss, corruption, and delay as well as other aspects of the physical environment such as those detected by the mote's sensors.

As Viptos is based upon Ptolemy's models of computation it includes many of the same entities including *actors*, and *directors*. From VisualSense it inherits a library of *channels* for communication and a wireless director. Actors are entities which perform a defined operation. Examples of actors include general-use objects like clocks and ramps, as well as more complex objects such as *MicaCompositeActor* which is used to represent the Mica2[1] mote(see Fig. 2). Directors are software components which act as schedulers for the simulation. In the case of of wireless communications all information that is passed without a direct connection from one actor to another is passed through a channel. The passing of this information varies for different channels. For example the *LimitedRangeChannel* only passes on packets to actors that are calculated to be within a certain range of the transmitting mote, and *AtomicWirelessChannel* passes on all packets regardless of distance.[3]

The ability to test routing algorithms is essential in the research and development of WSNs. Prior to the completion of this project, Viptos did not offer the option of viewing communication between wireless actors in any way other than reading the debug messages. The purpose of the additions made was to alleviate this discrepancy by providing a tool to developers that would allow the visualization of communication in a easy to interpret and graphical manner.

2 Process

Several approaches were used in the creation of this simulation. Existing software was analyzed for hints to produce the desired results in Viptos. Creating this code also required analysis of similar existing objects within the Viptos code base. Discussed below are the analyzed software and the changes made.

[1]Mica2 motes are wireless sensors developed at University of California, Berkeley and manufactured by Crossbow. These are the type of motes used in the live demos.

2.1 Surge: A Sensor Network Application

A program called `Surge` was a focus during this research, as it was a TinyOS program created for multihop networks. `Surge` itself is a simple program that is used to perform simple operations such as collecting photosensor data and sending it back to a base station. The program allows motes to organize themselves within the network by maintaining a parent node and their own depth in the tree. Initially, a mote is assigned an ID number, with the base station being 0, and then picks its parent by listening to messages sent by its neighbors and selecting the one with the smallest depth. After this setup, motes transmit their depth as part of their packet, and the base station periodically broadcasts a depth of 0. This information allows motes to change their selected parent node to a neighbor, if the link quality falls below a set value, on the basis of link quality and depth. This ability to automatically change routing paths in a wireless network is especially important in deployments where power is limited, and there is little to no human interaction [5].

The packet that is transmitted through the network is defined in `Surge`. Every packet, known as *TOS_Msg* takes the form of:

- **address**: 2 bytes
- **type**: 1 byte
- **group**: 1 byte
- **length**: 1 byte
- **data**: 29 - 1 byte each

The packet also contains several other fields which were mostly disregarded for this research. The **address** field is used to store the address of the receiving mote in the network. The **data** field is different based on the type of message being sent. The two main message styles are: *MultiHopMsg* and *SurgeMsg*. *MultiHopMsg* contains information on packet source and origin node addresses as well as packet sequence and node hop count values, which is primarily used to setup the network, and is periodically sent out so motes can reevaluate their connections within the network. *SurgeMsg* is used much more often for regular packets within the network. The message style that is being used is stored in *TOS_Msg*'s **type** field with each format having a unique **type** ID. The **group** field contains an ID for the group, which can be used to exclude communication with motes that are not part of the group. The **length** field defines how much relevant data is being stored in the **data** portion of the packet. When the *SurgeMsg* is used, the first byte of the **data** field contains the source address of the packet.

2.2 Analysis of Existing Live Demos

Mica2 motes (or any TinyOS based mote) can be programmed to run `Surge` by assigning a mote ID number to each, with the base node being set to zero. After the motes have been programmed, the base mote is connected to the computer via the programming board (see Fig. 4) which is then connected to the serial port. A serial forwarder program included with TinyOS, reads packet data from the serial port and forwards it to a Java tool to be displayed on a Java interface (see Fig. 3). The concept of connecting communicating motes with a graphical link is a common theme among wireless networks as it aids in their development by showing their interconnectivity in an easy to interpret format.

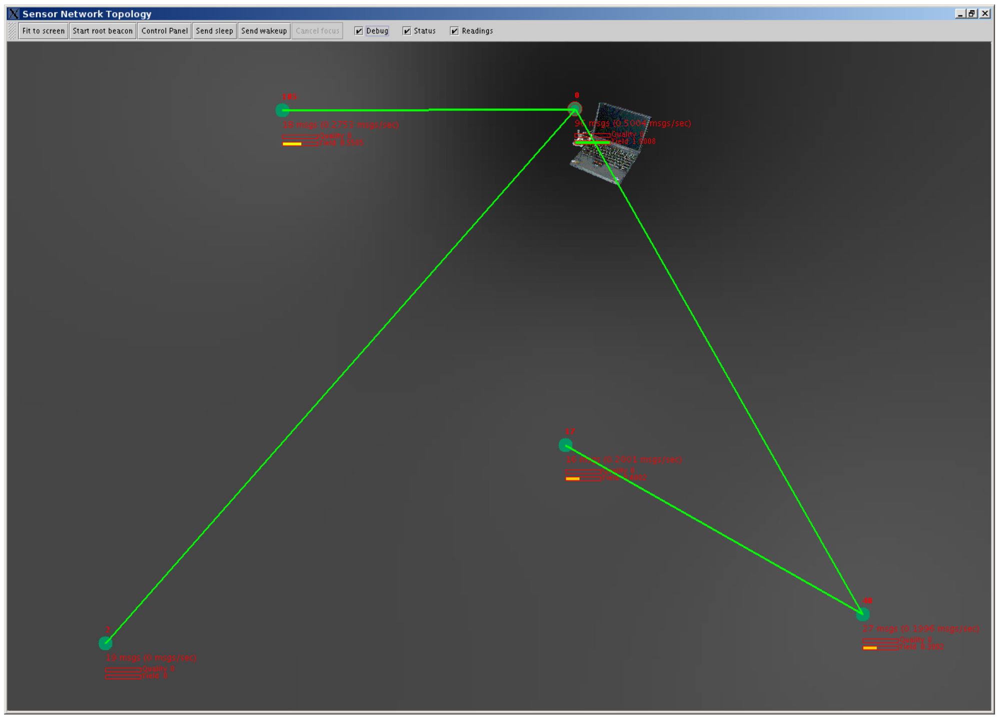


Figure 3. Live Surge Demo Screen Shot

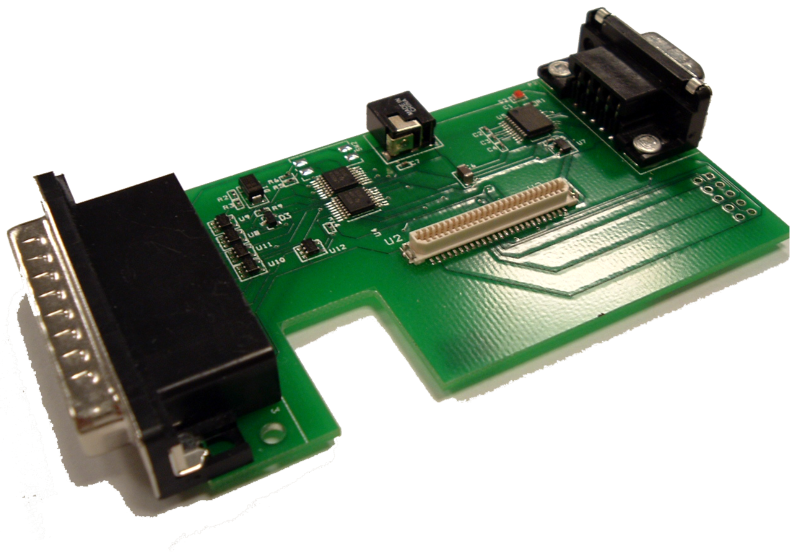
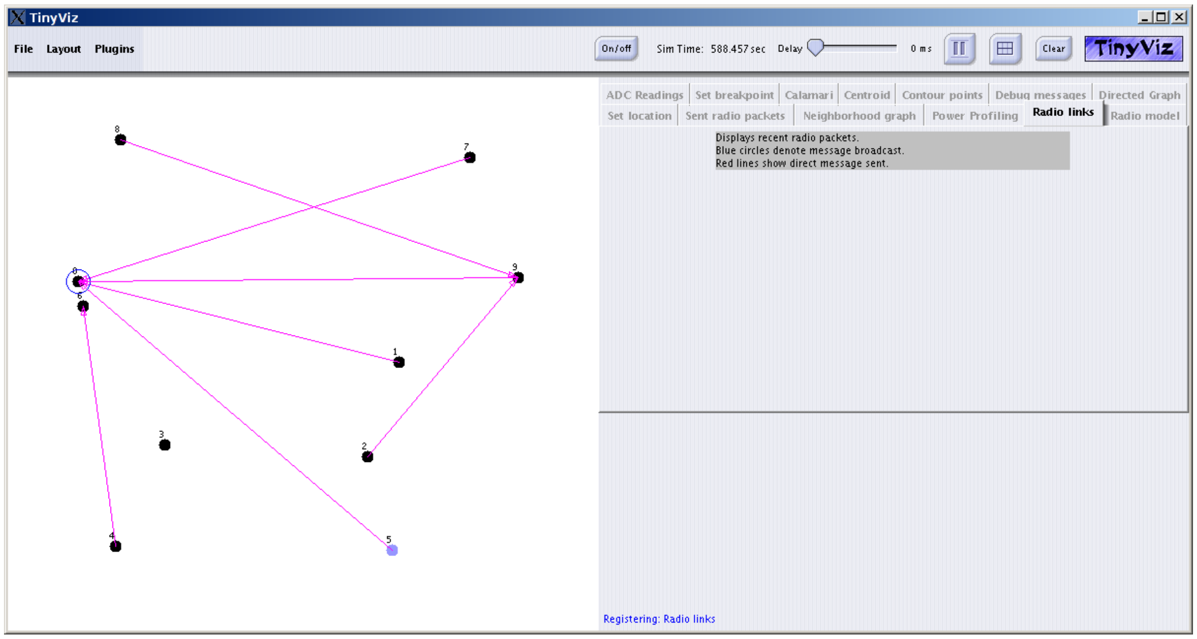


Figure 4. Mica2 Programming Board



Sim Time: 588.457 sec

0 ms

Displays recent radio packets.
Blue circles denote message broadcast.
Red lines show direct message sent.

Registering: Radio links

Figure 5. Simulation of Surge in TinyViz

2.3 Analysis of Existing Simulations

Since the goal of this research was to develop a tool in a simulation package another existing simulation was explored. A simulation provided by a combination of TOSSIM & TinyViz allows users to view the connections between motes as the network is simulated. To run the example simulation TOSSIM is started, and the number of nodes to be simulated is specified. After TOSSIM has started, TinyViz, a simulation GUI, is invoked to display the network as the simulation runs. Plugins which display radio links within the network, as well as sent packets, are several of many options within TinyViz to allow the user to manipulate the network. The radio links allow users to see what path a packet took to reach the base node of the network, and the sent packets plugin allows the viewing of packet data broken down into the specified fields. The focal point of this simulation is the depiction of communication with lines connecting the nodes. The TOSSIM/TinyViz simulation is similar to the capabilities of Viptos, however, Viptos itself is capable of simulating heterogeneous networks, which the example combination is not capable of.

2.4 Editing & Adding to Viptos Code

2.4.1 Creating LinkVisualizer

LinkVisualizer was created as part of this research and is based on the *ptolemy.domains.wireless.lib.TerrainProperty* entity already in use in the Viptos environment. *TerrainProperty* is an object which, when used in conjunction with *TerrainChannel*, prevents the sending of packets between motes that will cross the entity's icon in the simulation. *LinkVisualizer* is similar in this aspect, except it simply uses the received information to determine the location of the two motes. Once this information is attained, a line element created for just this purpose is used to connect the communicating objects (see Fig. 6). The *LinkVisualizer* entity is currently only implemented for use with *AtomicWirelessChannel*, however it could easily be included in other wireless channels. *LinkVisualizer* uses *TokenProcessor*, which is an interface that registers token processors with the channel in use so that it can

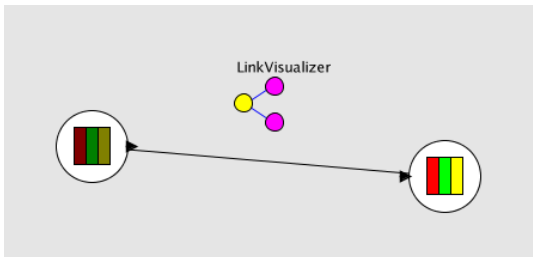


Figure 6. *LinkVisualizer*

inspect the packets as they are transmitted.

```
public void processTokens(RecordToken properties,
        Token token, WirelessIOPort sender, WirelessIOPort destination)
        throws IllegalActionException {

   if(_isOff) {
        Location senderLocation = (Location)sender.getContainer().getAttribute("_location");
        Location destinationLocation = (Location)destination.getContainer().getAttribute("_location");
        double x = (destinationLocation.getLocation())[0] - (senderLocation.getLocation())[0];
        double y = (destinationLocation.getLocation())[1] - (senderLocation.getLocation())[1];
        String moml = "<property name=\"_senderDestLine\" class=\"ptolemy.vergil.kernel.attributes.LineAttribute\">"
            + senderLocation.exportMoML()
            + "<property name=\"x\" value=\""
            + x
            + "\"/>"
            + "<property name=\"y\" value=\""
            + y
            + "\"/>"
            + "</property>";
        ChangeRequest request = new MoMLChangeRequest(this, getContainer(), moml) {
            protected void _execute() throws Exception {
                super._execute();
                LineAttribute line = (LineAttribute)getContainer().getAttribute("_senderDestLine");
                line.moveToFirst();
                line.setPersistent(false);
            }
        };
        requestChange(request);
        _isOff = false;
   } else {
        if (getContainer().getAttribute("_senderDestLine") != null) {
            String moml = "<deleteProperty name=\"_senderDestLine\"/>";
            ChangeRequest request = new MoMLChangeRequest(this, getContainer(), moml);
            requestChange(request);
            _isOff = true;
        }
    }
}
```

Figure 7. *ProcessToken* method in *LinkVisualizer*

LinkVisualizer required the writing of the method *processTokens(...)* to find the location of the container, an entity which contains different actors, and which the sender's *WirelessIOPort* resides within the simulation GUI. *LinkVisualizer* then uses the containers location in combination with the location of the destination mote's to calculate the distance in terms of rise and run between the two motes(see Fig. 7). A *MoMLChangeRequest* method is then called to create a line in the simulation between the two actors. The decision was made to remove the line shortly after it is made so as not to constantly depict communication, just that communication has recently occurred.

2.4.2 Adding to Existing Code

Outside of the creation of the *LinkVisualizer* class, changes were made to the *WirelessChannel* class to extend *TokenProcessor* which is the interface for the *processTokens(...)* method. *TokenProcessor* registers with a channel so that it may receive packet information as the simulation is running. Changes were also made to *AtomicWirelessChannel* to track instances of *TokenProcessor*s and call *processTokens(. . .)* when transmitting a packet. These changes included:

- adding *registerTokenProcessor(. . .)* to register a token processor for transmissions from specified ports.
- adding a *processTokens(. . .)* method to the channel to call the *processTokens(. . .)* method in the visualizer for each instance of a *TokenProcessor*.
- adding *unregisterTokenProcessor(. . .)* to unregister a token processor for transmissions from specified ports.
- changing the private *transmitTo(. . .)* method to call the *processTokens(. . .)* method within *AtomicWirelessChannel*
- creating a private set *tokenProcessors* to contain the token processors without a specified port
- creating a private HashMap *tokenProcessorsByPort* to contain token processors with a specified port

2.5 Creation of Surge Demo in Viptos

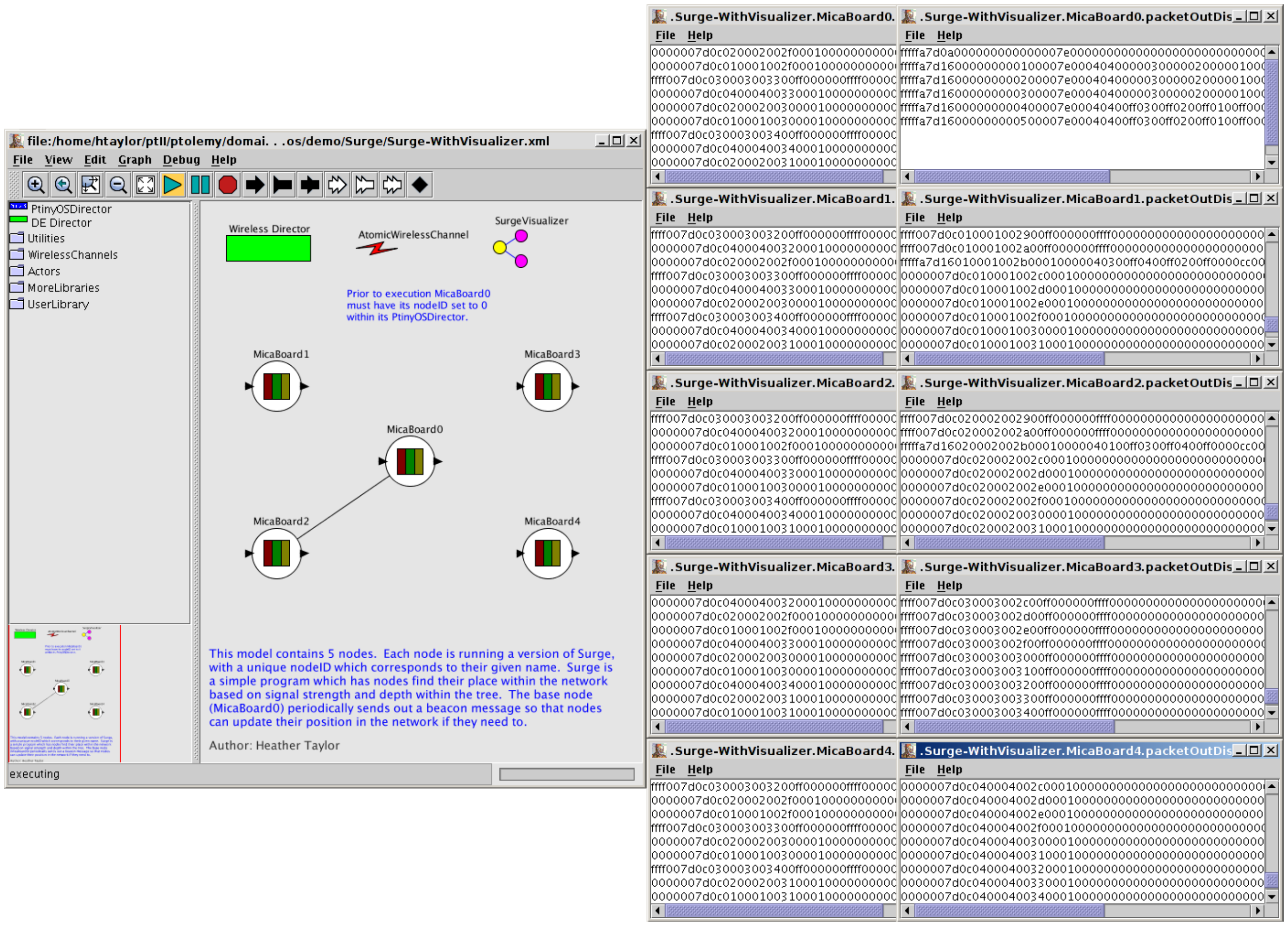
To create a demo which utilized the capability of viewing graphical links between motes during a simulation, an automatically generated model of the existing TinyOS software for `Surge` was modified. Multiple versions were made of motes running the `Surge` software. Each mote was then given a unique **nodeID** with a base node being set to 0. When run, this demo produces displays of the contents of their in- and out-ports. This result can be seen in Fig. 8. In addition to this setup the *SurgeVisualizer* class was created to extend *LinkVisualizer* so that it only creates links between motes when they meet set conditions. When a packet is sent through the network, *SurgeVisualizer* intercepts it, and examines the packet's **type** and **address** fields. First the packet **type** is checked for the *SurgeMsg* **type**. This is done to ensure visual links are not created for beacon messages from the base node. In addition the **address** field is compared to the destination **nodeID** in the simulation. If both the **type** and **address** conditions are met, a line is drawn between the sender and destination motes.

3 Contributions

3.1 Tool for Visualization

With the addition of *LinkVisualizer* and *SurgeVisualizer* entities, users of Viptos can now view the communication that transpires between wireless actors within a demo (see Fig. 6). Previously, only users of other simulation programs such as TOSSIM & TinyViz could utilize the ability to view communication in terms of a graphical link. Now this viewing capability has been extended to Viptos, where users can benefit from the ability to simulate heterogeneous networks.

When users create a Viptos simulation they can simply drag the *LinkVisualizer* entity from a component menu into their simulation and together with the *AtomicWirelessChannel*, the method to create and remove the line will be called automatically. Also the *SurgeVisualizer* can be used in conjunction with *AtomicWirelessChannel* to observe the routing tree used in a multihop network that uses the TinyOS `Surge` routing protocol.



Prior to execution MicaBoard0 must have its nodeID set to 0 within its PtinyOSDirector.

This model contains 5 nodes. Each node is running a version of Surge, with a unique nodeID which corresponds to their given name. Surge is a simple program which has nodes find their place within the network based on signal strength and depth within the tree. The base node (MicaBoard0) periodically sends out a beacon message so that nodes can update their position in the network if they need to.

Author: Heather Taylor

.Surge-WithVisualizer.MicaBoard0.

```
0000007d0c020002002f000100000000000
0000007d0c010001002f000100000000000
ffff007d0c030003003300ff000000ffff0000
0000007d0c040004003300010000000000
0000007d0c020002003000010000000000
0000007d0c010001003000010000000000
ffff007d0c030003003400ff000000ffff0000
0000007d0c040004003400010000000000
0000007d0c020002003100010000000000
```

.Surge-WithVisualizer.MicaBoard0.packetOutDis

```
fffffa7d0a000000000000007e000000000000000000000000000
fffffa7d160000000000100007e0004040000030000020000010
fffffa7d160000000000200007e0004040000030000020000010
fffffa7d160000000000300007e0004040000030000020000010
fffffa7d160000000000400007e00040400ff0300ff0200ff0100ff00
fffffa7d160000000000500007e00040400ff0300ff0200ff0100ff00
```

.Surge-WithVisualizer.MicaBoard1.

```
ffff007d0c030003003200ff000000ffff0000
0000007d0c040004003200010000000000
0000007d0c020002002f000100000000000
ffff007d0c030003003300ff000000ffff0000
0000007d0c040004003300010000000000
0000007d0c020002003000010000000000
ffff007d0c030003003400ff000000ffff0000
0000007d0c040004003400010000000000
0000007d0c020002003100010000000000
```

.Surge-WithVisualizer.MicaBoard1.packetOutDis

```
ffff007d0c010001002900ff000000ffff000000000000000000000
ffff007d0c010001002a00ff000000ffff000000000000000000000
fffffa7d160100010002b000100000040300ff0400ff0200ff0000cc00
0000007d0c010001002c0001000000000000000000000000000
0000007d0c010001002d0001000000000000000000000000000
0000007d0c010001002e0001000000000000000000000000000
0000007d0c010001002f0001000000000000000000000000000
0000007d0c01000100300001000000000000000000000000000
0000007d0c01000100310001000000000000000000000000000
```

.Surge-WithVisualizer.MicaBoard2.

```
ffff007d0c030003003200ff000000ffff0000
0000007d0c040004003200010000000000
0000007d0c010001002f000100000000000
ffff007d0c030003003300ff000000ffff0000
0000007d0c040004003300010000000000
0000007d0c010001003000010000000000
ffff007d0c030003003400ff000000ffff0000
0000007d0c040004003400010000000000
0000007d0c010001003100010000000000
```

.Surge-WithVisualizer.MicaBoard2.packetOutDis

```
ffff007d0c020002002900ff000000ffff000000000000000000000
ffff007d0c020002002a00ff000000ffff000000000000000000000
fffffa7d160200020002b000100000040100ff0300ff0400ff0000cc00
0000007d0c020002002c0001000000000000000000000000000
0000007d0c020002002d0001000000000000000000000000000
0000007d0c020002002e0001000000000000000000000000000
0000007d0c020002002f0001000000000000000000000000000
0000007d0c02000200300001000000000000000000000000000
0000007d0c02000200310001000000000000000000000000000
```

.Surge-WithVisualizer.MicaBoard3.

```
0000007d0c040004003200010000000000
0000007d0c020002002f000100000000000
0000007d0c010001002f000100000000000
0000007d0c040004003300010000000000
0000007d0c020002003000010000000000
0000007d0c010001003000010000000000
0000007d0c040004003400010000000000
0000007d0c020002003100010000000000
0000007d0c010001003100010000000000
```

.Surge-WithVisualizer.MicaBoard3.packetOutDis

```
ffff007d0c030003002c00ff000000ffff000000000000000000000
ffff007d0c030003002d00ff000000ffff000000000000000000000
ffff007d0c030003002e00ff000000ffff000000000000000000000
ffff007d0c030003002f00ff000000ffff000000000000000000000
ffff007d0c030003003000ff000000ffff000000000000000000000
ffff007d0c030003003100ff000000ffff000000000000000000000
ffff007d0c030003003200ff000000ffff000000000000000000000
ffff007d0c030003003300ff000000ffff000000000000000000000
ffff007d0c030003003400ff000000ffff000000000000000000000
```

.Surge-WithVisualizer.MicaBoard4.

```
ffff007d0c030003003200ff000000ffff0000
0000007d0c020002002f000100000000000
0000007d0c010001002f000100000000000
ffff007d0c030003003300ff000000ffff0000
0000007d0c020002003000010000000000
0000007d0c010001003000010000000000
ffff007d0c030003003400ff000000ffff0000
0000007d0c020002003100010000000000
0000007d0c010001003100010000000000
```

.Surge-WithVisualizer.MicaBoard4.packetOutDis

```
0000007d0c040004002c0001000000000000000000000000000
0000007d0c040004002d0001000000000000000000000000000
0000007d0c040004002e0001000000000000000000000000000
0000007d0c040004002f0001000000000000000000000000000
0000007d0c04000400300001000000000000000000000000000
0000007d0c04000400310001000000000000000000000000000
0000007d0c04000400320001000000000000000000000000000
0000007d0c04000400330001000000000000000000000000000
0000007d0c04000400340001000000000000000000000000000
```

Figure 8. Contents of each mote's ports

3.2 Surge Demo in Viptos

The creation of a demo in Viptos that utilizes the TinyOS software for `Surge` in conjunction with *SurgeVisualizer* will allow users to view and explore the use of a visualization tool within an existing simulation. This new `Surge` simulation expands on existing simulations by allowing users to use the Viptos environment, which includes functionality previously unavailable in the TinyViz simulation software used.

4 Conclusion

Viptos is a growing development tool, with uses in development and research. As WSNs are being further explored, the tools available in the Viptos environment will allow developers to better create and interpret the functionality of software used. The development of these easily usable tools will enhance the level to which these improvements can be made as analysis of routing algorithms within a network can be used to further refine software to a more efficient level. As WSNs are being deployed in more environments, this efficiency becomes more important.

Acknowledgments

Thank you to SUPERB for allowing me this opportunity. Thank you to my graduate student mentor Elaine Cheong for helping me through my project and keeping me on track. Thank you to Dr. Jonathan Sprinkle for serving as a research supervisor and keeping us all on our toes while helping solve various technical problems. Thank you to Prof. S. Shankar Sastry for acting as my faculty mentor and enlightening us all on the topics of CHESS & TRUST. Thank you to Prof. E. A. Lee for his help in writing the code needed to create a line.

References

[1] Ptolemy ii website. http://ptolemy.berkeley.edu/ptolemyII/, UC Berkeley.

[2] Tinyos community forum. http://www.tinyos.net/.

[3] Viptos website. http://ptolemy.berkeley.edu/viptos/, UC Berkeley.

[4] E. Cheong, E. A. Lee, and Y. Zhao. Viptos: A graphical development and simulation environment for tinyos-based wireless sensor networks. Technical Report UCB/EECS-2006-15, University of California, Berkeley, Feb. 2006.

[5] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler. The nesc language: A holistic approach to networked embedded systems. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2003.

[6] In Proceedings of the First ACM Conference on Embedded Networked Sensor Systems. *TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications*. SenSys, 2003.