Visualization in Sensor Network Simulator, SENSE and Its Use in Protocol Verification

Christopher Morrell, Thomas Babbitt, and Boleslaw K. Szymanski
Department of Computer Science and
Center for Pervasive Computing and Networking
Rensselaer Polytechnic Institute
110 8th Street
Troy, NY 12180
morrec, babbit, szymansk@cs.rpi.edu

ABSTRACT

People are always looking for a way to visualize a problem. Whether that be by creating a 3-dimensional model, building a graph, or simply drawing on a board. The issue of visualizing a problem also finds its way into networking, and more specifically wireless sensor networking. Many networking simulation tools exist, and while some are very good at performing simulation, they may not be good at providing a visualization tool to examine simulation results. SENSE is an example of a simulation tool specifically design to easily build a sensor network model and simulate it efficiently, yet, it lacks the ability to robustly display the results of the simulation in a visual manner. This paper addresses that problem, and presents a clean solution for connecting the SENSE wireless sensor network simulator with iNSpect, a visualization tool intended to work with ns2. This paper also presents two examples of the use of this tool combination to identify and solve issues that arose in the development of wireless sensor network routing protocols.

Keywords

network visualization, wireless sensor networks, network simulator, SENSE, iNSpect

1. INTRODUCTION

As has been said many times before, a picture is worth a thousand words. In the case of network simulation, a picture is worth a thousand pages of simulation output. Using simulation to debug protocols and algorithms for sensor networks often results in the designer confronted with pages upon pages of simulator output, that without serious effort to understand, is mostly meaningless. The author in [10] also agrees that we have to find a way to deal with large amounts of information.

> The amount of information available to scientists from large-scale simulations, experiments, and data collection is unprecedented. In many instances, the abundance and variety of information can be overwhelming.

This author also discusses the importance to science to be able to abstract data in such a way that it is meaningful to the human brain. Certainly the network setup and statistical outputs are meaningful, but the line by line textual output of which packets went through which nodes in the network don't mean much without a picture. In addition to understanding output, there are some problems that may arise while developing protocols and algorithms for networks, that would not be identifiable without a way to really see what is happening in a network. Without the aid of visualization, the designer would regularly need to analyze pages of simulator print-outs, to draw a picture of what happened in the simulation, and where something went wrong. Protocols for wireless networks are particularly sensitive to environment perturbation and transient failures. As a result, any, even small, change in such protocol design may results in huge change in behavior. Understanding and controlling the ramifications of such changes are greatly aided by the visualization of the simulated execution. Hence, we fully agree with the views of the authors of the nam tool [2] supplementing ns2 who succinctly stated their motivation for developing a visualization tool as follows:

> Protocol design requires understanding state distributed across many nodes, complex message exchanges, and with competing trac. Traditional analysis tools (such as packet traces) too often hide protocol dynamics in a mass of extraneous detail...nam [is] a network animator that provides packet-level animation and protocol-specific graphs to aid the design and debugging of new network protocols...Nam now integrates traditional time-event plots of protocol actions and scenario editing capabilities.

Equally important is the aid that visualization can provide in understanding and teaching how the sensor network operates and how protocols and algorithms designed for sensor networks really works. Specially in wireless networks, routing and distributed computing is often a complex and dynamic process which is difficult to conceptualize or grasp intuitively and therefore difficult to teach. Visualization can be effectively used to build intuition about the network protocol modus operandi which then can create a foundation for deeper understanding of the protocol. The use of visualization in teaching networking has been discussed in [8]. The above mentioned debugging and educational needs motivated the research presented in this paper.

SENSE is our simulator of choice when working on problems involving wireless sensor networks. The recent problems we have been working on are those that involve the development of routing protocols within these networks. While SENSE is an extremely robust, easy to use simulation tool, the visualization tools provided with it are lacking. In the search for a solution, many iterations of tools have evolved, until a pre-built tool, intended for use with ns2 [6], was discovered. This tool, entitled iNSpect [12], was written by a group of researchers at Colorado School of Mines. When used in conjunction with SENSE, iNSpect provides the ability to easily create an animated playback from the output of a wireless sensor network simulation. This animation provides researchers the ability to step through a simulation, and gain a visual image as to how packets traverse the network, how nodes work together, and how network flows relate to each other.

This coordination between SENSE and iNSpect, has been a breakthrough in our use of the SENSE simulator, and is therefore the subject of this paper. We will first discuss the SENSE simulator, providing some background on its development, and the functionality that it provides. This will be followed by a discussion about the visualization tools that were built into SENSE as it proceeded in development, ending with a description of how SENSE and iNSpect are used together. Lastly, we will present the wireless sensor network routing protocols that encompass our work, and two examples of problems that were discovered, solved, and tested using the combination of SENSE and iNSpect. This combination of tools has proven invaluable in the advancement of our research.

2. SENSE SIMULATOR

SENSE is a C++ based simulator which was created to specialize in the simulation of wireless sensor networks. Originally presented in [3], SENSE was created to provide extensibility, reusability, and scalability to wireless network simulation. SENSE uses a component-port model to represent parts of a network, and is built on top of COST [4], which is a general purpose discrete event simulator. COST is built on top of a component based C++ extension entitled CompC++ [14].

By using a component-port model as the basis of design, SENSE achieves its goal of being extensible, as new protocol stacks are easily implemented. This model allows a user to abstract many of the details of the simulator, and simply create new network components as needed. In the case of a wireless sensor network, the sensor is a composite component, that consists of several lower level components. These low level components include the layers of the protocol stack, mobility, and power management.

In SENSE, components are connected by two types of ports, inports and outports. Inports are generally functional in nature, in that they implement a certain function. Outports on the other hand can be described as an abstraction of a function pointer; they define what functionality they expect of others. More specifically, the ports are used to connect components. For example, each layer of the protocol stack, which is defined as a component, gets an outport pointing up to the next higher layer, and an outport pointing down to the next lower layer. In addition to these, each layer also has an inport coming from each layer around it. Layers that are required to communicate directly with the mobility or power management components have additional ports to allow data to flow from those components into the protocol stack. Inports and outports are simply connected to one another, providing a connection between components for data to flow through and information to be exchanged.

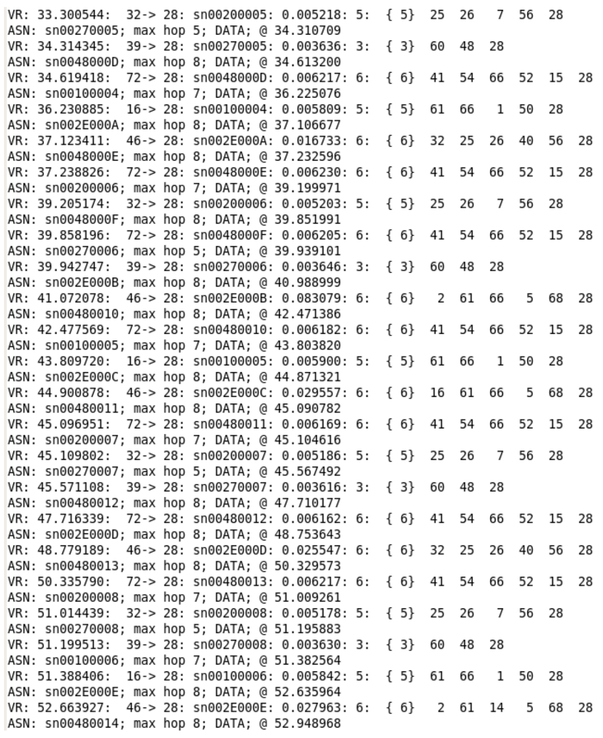
In addition to providing the ability to easily expand the simulator to support newly developed protocols, SENSE was written to include support for many popular protocols. These included protocols range all the way from the physical layer to the application layer, and include IEEE 802.11, AODV, several radio models, several power management models, and others. Originally SENSE was created as a wireless sensor network specific response to the wildly popular network simulator ns2 [6].

As popular as ns2 is, without many add-on packages, it is not well suited to simulate wireless sensor networks. The fact that ns2 was originally written as a wired network simulator, the layers required to simulate a wireless network in this environment increases its complexity. As SENSE was written specifically as a simulator for wireless sensor networks, it is fairly popular in that community, and has been used by [16], [13], and [9], in addition to many others.

3. VISUALIZATION WITHIN SENSE

SENSE was originally created without the ability to provide visual feedback from a simulation, and although SENSE has been a commonly used simulator, the lack of visualization has been one of the largest complaints about it [7]. Normal simulation feedback looked much like figure 1. Visual Route (VR) lines provide simulation time, source, destination, packet number, end-to-end packet transmission time, and a node by node path through the network. Assign Sequence Number (ASN) lines are output upon the source identifying a packet to be sent. Data includes packet number, maximum hops to destination, packet type, and simulation time. All of this information is very useful to a researcher who is testing a new network protocol, but the effort required to truly understand what each line means is enormous. One would have to manually plot each node location and overlay each packet's path in order to see how each flow, packet, or node related to each other.

As SENSE was improved through several versions, some minimal visualization capabilities were added to it. These new capabilities allowed it to extract a plot that included the locations of all of the nodes as shown in figure 2(a). In addition to this, SENSE would provide a graphical representation of the path taken by any single packet, shown in figure 2(b).



```
VR: 33.300544:  32-> 28: sn00200005: 0.005218: 5:  { 5}  25  26   7  56  28
ASN: sn00270005; max hop 5; DATA; @ 34.310709
VR: 34.314345:  39-> 28: sn00270005: 0.003636: 3:  { 3}  60  48  28
ASN: sn0048000D; max hop 8; DATA; @ 34.613200
VR: 34.619418:  72-> 28: sn0048000D: 0.006217: 6:  { 6}  41  54  66  52  15  28
ASN: sn00100004; max hop 7; DATA; @ 36.225076
VR: 36.230885:  16-> 28: sn00100004: 0.005809: 5:  { 5}  61  66   1  50  28
ASN: sn002E000A; max hop 8; DATA; @ 37.106677
VR: 37.123411:  46-> 28: sn002E000A: 0.016733: 6:  { 6}  32  25  26  40  56  28
ASN: sn0048000E; max hop 8; DATA; @ 37.232596
VR: 37.238826:  72-> 28: sn0048000E: 0.006230: 6:  { 6}  41  54  66  52  15  28
ASN: sn00200006; max hop 7; DATA; @ 39.199971
VR: 39.205174:  32-> 28: sn00200006: 0.005203: 5:  { 5}  25  26   7  56  28
ASN: sn0048000F; max hop 8; DATA; @ 39.851991
VR: 39.858196:  72-> 28: sn0048000F: 0.006205: 6:  { 6}  41  54  66  52  15  28
ASN: sn00270006; max hop 5; DATA; @ 39.939101
VR: 39.942747:  39-> 28: sn00270006: 0.003646: 3:  { 3}  60  48  28
ASN: sn002E000B; max hop 8; DATA; @ 40.988999
VR: 41.072078:  46-> 28: sn002E000B: 0.083079: 6:  { 6}   2  61  66   5  68  28
ASN: sn00480010; max hop 8; DATA; @ 42.471386
VR: 42.477569:  72-> 28: sn00480010: 0.006182: 6:  { 6}  41  54  66  52  15  28
ASN: sn00100005; max hop 7; DATA; @ 43.803820
VR: 43.809720:  16-> 28: sn00100005: 0.005900: 5:  { 5}  61  66   1  50  28
ASN: sn002E000C; max hop 8; DATA; @ 44.871321
VR: 44.900878:  46-> 28: sn002E000C: 0.029557: 6:  { 6}  16  61  66   5  68  28
ASN: sn00480011; max hop 8; DATA; @ 45.090782
VR: 45.096951:  72-> 28: sn00480011: 0.006169: 6:  { 6}  41  54  66  52  15  28
ASN: sn00200007; max hop 7; DATA; @ 45.104616
VR: 45.109802:  32-> 28: sn00200007: 0.005186: 5:  { 5}  25  26   7  56  28
ASN: sn00270007; max hop 5; DATA; @ 45.567492
VR: 45.571108:  39-> 28: sn00270007: 0.003616: 3:  { 3}  60  48  28
ASN: sn00480012; max hop 8; DATA; @ 47.710177
VR: 47.716339:  72-> 28: sn00480012: 0.006162: 6:  { 6}  41  54  66  52  15  28
ASN: sn002E000D; max hop 8; DATA; @ 48.753643
VR: 48.779189:  46-> 28: sn002E000D: 0.025547: 6:  { 6}  32  25  26  40  56  28
ASN: sn00480013; max hop 8; DATA; @ 50.329573
VR: 50.335790:  72-> 28: sn00480013: 0.006217: 6:  { 6}  41  54  66  52  15  28
ASN: sn00200008; max hop 7; DATA; @ 51.009261
VR: 51.014439:  32-> 28: sn00200008: 0.005178: 5:  { 5}  25  26   7  56  28
ASN: sn00270008; max hop 5; DATA; @ 51.195883
VR: 51.199513:  39-> 28: sn00270008: 0.003630: 3:  { 3}  60  48  28
ASN: sn00100006; max hop 7; DATA; @ 51.382564
VR: 51.388406:  16-> 28: sn00100006: 0.005842: 5:  { 5}  61  66   1  50  28
ASN: sn002E000E; max hop 8; DATA; @ 52.635964
VR: 52.663927:  46-> 28: sn002E000E: 0.027963: 6:  { 6}   2  61  14   5  68  28
ASN: sn00480014; max hop 8; DATA; @ 52.948968
```

Figure 1: An example of the text only output from the SENSE simulator.

While this basic level of information is useful, more is still required to fully understand what is happening throughout the life of an entire network, or even throughout the lifetime of a given network flow or packet. Of course through some basic image manipulation, additional information was available, including the ability to overlay node identities, node locations with paths, multiple paths, and with some effort some animation of packets moving across the network. In order to create the animation, separate images were created of each packet's path on the network, which were then stacked together into an animation. While this animation could provide additional information, and give a more visually appealing view of the network, once created, it was static, and the time and effort required to build this product was excessive in relation to the information gained from it. This method of visualizing the network also failed in its inability to provide any useful statistics with the pictures and animations.

4. VISUALIZATION WITH SENSE AND INSPECT

Although several iterations of tools were created to provide some visualization functionality to SENSE, it was decided that work on the functionality of the simulator would be more important, and finding a visualization tool that already existed, and could be used with SENSE would be more appropriate. Since ns2 is probably the most popular network simulator in existence, of course the first choice of visualization tools was the Network Animator (NAM) that is packaged as part of ns2. NAM was made to work with SENSE, but as it was never really designed to visualize wireless networks [5], it is missing key capabilities required in a wireless

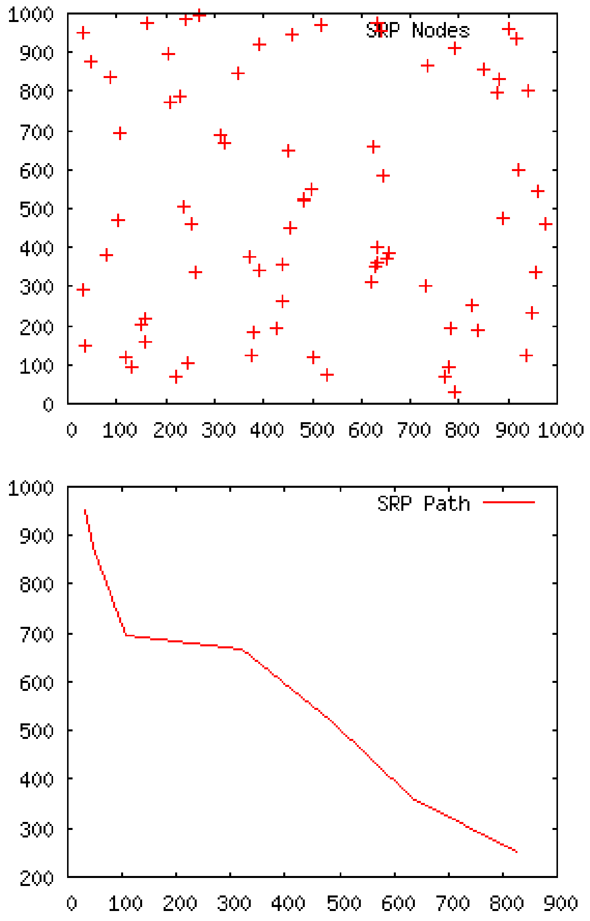


Figure 2: (a) An example of the original visualization capabilities built in to SENSE, this plot shows the location of 75 nodes on a 1000 x 1000 unit area. (b) The path followed by the first packet in a simulation of the SRP protocol.

simulation. Since broadcast is the primary medium of a wireless sensor network, and our protocols are self-selecting, there is no predefined path or link between two nodes. Nam requires a link to be present in order to track packets between nodes. NAM's depiction of a broadcast also did not meet the requirements of what we were looking for. In NAM, a broadcast is depicted by a simple circle around the broadcasting node that is the size of the wireless transmission range. Since we don't know the node within that circle that would be selected to forward a packet, our protocol required the link to be created after the fact, increasing the complexity of the model. Although NAM did not meet our specific requirements, it did lead us to the discovery of the visualization tool iNSpect.

Originally presented in [12], a group of researchers at Colorado School of Mines created a visualization tool called iNSpect that is specifically tailored to visualize wireless network simulations. Although intended for use with ns2, and in the future is planned to work well with ns3, we've chosen to use iNSpect as the visualization tool for SENSE. The shortcomings in NAM that have been discussed were also identified by other researchers, and led to the development of iNSpect. iNSpect is a C++ OpenGL based visualization tool that takes ns2 mobility and simulation files as input, and presents a Cartesian coordinate based display of the simulated network as shown in figure 3. iNSpect provides the ability to see packets flow across the wireless network, as well as collecting various statistics. One feature of iNSpect that can easily be compared to NAM, is its ability to depict a broadcast. While NAM displays a circle as described earlier, iNSpect draws an arrow from the broadcasting node to all nodes within transmission range. Once we have a node that is selected to forward the given packet, it changes its color to green to depict its win. All other nodes change their colors to black, showing that they are not the winners of that election.

Since SENSE was not originally created to produce ns2 type output, and iNSpect was created to use ns2 output, the main effort in making SENSE work well with iNSpect was to modify the output from a SENSE simulation to match the output of ns2. Once broken down to its most basic levels, this is a relatively simple process. iNSpect expects three separate input files to produce a visualization of a simulation. Those files are the configuration file, the mobility file, and the simulation file.

The configuration file is a user created file based on a template provided with the iNSpect tool. The main items that must be modified in the configuration file are the physical dimensions of the simulated field, the number of nodes within the given simulation, the start and end times of the portion of the simulation to be visualized, and the colors of the nodes. The color definition also includes the ability to separate events into different categories for statistic gathering. Within the configuration file, it is also possible to set the amount of simulation time that an event is visualized for before it reverts to its default state.

The mobility file is the file that iNSpect uses to present the physical locations of the nodes. In our work with SENSE and iNSpect, we've not provided the ability for nodes to move, so SENSE simply provides output that establishes the initial positions of the nodes. In SENSE, node locations are established at network setup, so a simple textual output in the format expected by iNSpect gave us all of the data required for the mobility file. This format is simply three lines for each node, which identify the X, Y, and Z coordinates of the given node.

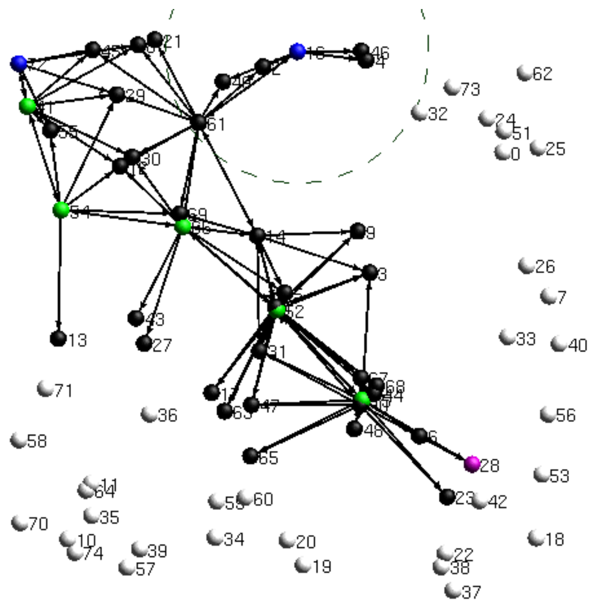
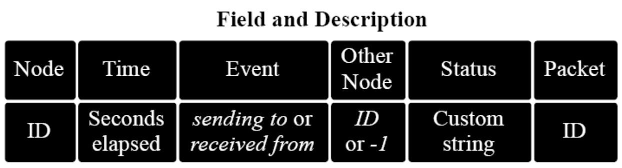


Figure 3: An example of the visualization window in iNSpect. In our protocols, we use black to identify nodes that intentionally drop packets, green are forwarding nodes, blue are sources, and purple are destinations.

Lastly, the simulation file is the most important part, and provides all of the information to the visualizer that concerns data transmissions. These include sent packets, received packets, acknowledged packets and dropped packets. This is of course the most complex of the three files required by iNSpect. In order to provide the output required by iNSpect, we used the basic template provided in [11] which is also include here as figure 4. In order to create events for iNSpect to display, we simply had SENSE output a line of text that filled the fields presented. The fields include the ID or address of the node, the simulation time of the event to be displayed, a description of the event (either sending to or received from), the node on the other end of the event, or -1 for a broadcast, a status string, and the packet ID. The status string provides a great deal of flexibility in the use of the visualizer, as that string is used to provide the coloring in the simulation replay in iNSpect. In our implementation, we've used the status string to separate network initialization packets from data packets. In addition to this separation, we've used the status string to identify source nodes, destination nodes, forwarding nodes, nodes that intentionally drop packets, and several other types of events.

As mentioned previously, the implementation of providing iNSpect compatible output from SENSE, while effective, is


Field and Description

Node	Time	Event	Other Node	Status	Packet
ID	Seconds elapsed	*sending to* or *received from*	*ID* or *-1*	Custom string	ID

Figure 4: The table that gives the requirements for an event in iNSpect.

also very simple. The extent of the work required involves identifying when events in the protocol need to be displayed in the visualization. For our protocols, those events were the transmission of a packet, the reception of a packet, the winning of a forwarding election, and the intentional dropping of a packet due to the loss of an election. Each of these events simply required an output from the simulator into the simulation file that followed the format presented in the table in figure 4. As protocols change, or different events within the simulator want to be focused on, a simple change of where text is output from the simulator is all that is required. It is also possible to change the iNSpect configuration file to no longer recognize specific status strings if the data represented by those status strings is not desired.

In addition to the basic visualization capabilities that iNSpect brings to SENSE, there are several other features that increase is value immensely. First is the ability to change the speed of the animation on the fly, including the ability to skip forward and backward in 5 second increments. If all that is required is the statistics that are presented at the end of a simulation, the speed may be set to extremely fast. In addition, it is possible to rewind the animation, thus reviewing specific events within the simulation repeatedly. iNSpect also provides the ability to view node coordinates in the animation. Another extremely useful tool within iNSpect is the ability to view a connectivity graph or conduct a partition check of the network. These tools give more insight into the physical layout of the nodes, and can help to identify connectivity issues within the network. While iNSpect is running an animation, each time it changes colors due to an event, statistics are gathered to represent that event as well. At any time in the simulation, a user may print the currently collected statistics both for the network as a whole, or for individual nodes. Lastly, iNSpect provides the ability capture both still images and animated movies for later replay. All of these features not only make iNSpect a great tool for providing wireless network visualization, but since it now functions well with SENSE, also provides excellent visualization capabilities for simulating wireless sensor networks.

5. USING VISUALIZATION FOR PROTOCOL ENHANCEMENT

Although visualization is a wonderful tool to have, it does not provide any added value if it is not put to good use. To that end, I will first describe the family of protocols that we have worked on, and will follow that with two specific examples of how SENSE with iNSpect has been useful in this work.

5.1 The SSR family of protocols

As described in [1] the Self-Selective Routing (SSR) family of protocols was developed as a possible solution to the problem of Wireless Sensor Network Routing. The SSR family consists of several different protocols which began with SSR, followed by Self-Healing Routing (SHR), and Self-Selecting Reliable Path Protocol (SRP). The common theme between all of these protocols is that nodes are allowed to make routing decisions autonomously. These decisions are based simply on the number of hops required to reach the destination of a given flow.

At network establishment, each source sends a Destination Request (DREQ) packet to their flow's destination. This DREQ packet is flooded across the network, until every node has received it and forwarded it. Upon receipt of the DREQ, each destination node replies with a Destination Reply (DREP) packet, which is also flooded across the network. As each node receives and forwards the DREP packet, they record either the hopcount of the current packet or their current hopcount, if one exists, whichever is lower. At the completion of the DREQ/DREP phase of the network, each node in the network knows how many hops they are away from the destination for any active flow in the network.

Once DREQ/DREP is complete, Data can be sent across the network by simply selecting a random delay between 0 and λ, where λ is a delay factor used to tune the network. Once a packet whose source is further from the destination for the given flow is received by a node, it selects its random delay value and sets a timer for that delay. Once a node's timer expires, it considers itself the winner of the election, and will forward the packet unless it was halted due to transmission of the packet by some other node whose timer expired first.

Originally, SSR provided no ability to repair damaged routes. Therefore, if a path were to be interrupted by some kind of failure, the packets would simply be lost. This brought about the development of SHR, which provides some route repair facilities. Although seemingly simplistic, the route repair algorithm used in SHR is the same that is still used in SRP and the latest version of the protocol, Reliable Path Self-Selection Protocol (RPSP). If a node does not see the packet that it sent forwarded by a closer node, it will broadcast the packet again. If it fails a second time, it will simply increase its hopcount to the destination by two and forward the packet again. This hopcount increase will make it appear that the forwarding node is further away from the destination, thus allowing other nodes to compete for the ability to forward the packet. This will continue until such time as some other node forwards the packet or the TTL is reached and the packet dies.

While SHR functioned quite well, an improvement was found that works extremely well to increase reliability and throughput. Once a node wins the election for a given flow twice in a row, it will artificially decrease its delay to just over the radio transition time. This almost guarantees that the node will continue to win elections for subsequent packets in the flow. It was proven in [15] that if at least one reliable path exists, it can be guaranteed that the network will eventually converge to the shortest reliable path, hence the name Self-Selecting Reliable Path Protocol (SRP).

5.2 Visualization to modify DREQ/DREP

After enhancing SENSE with an interface to iNSpect, it was observed that the network establishment, or DREQ/DREP phase of the SRP protocol was not working correctly. As designed, each source node would only send a single DREQ packet for any given flow. It was observed that when implemented in SENSE, an error occurred that allowed many nodes to repeatedly send DREQ packets. This only occurred when the random time selected between packet transmissions at the source was less than the round trip time for the DREQ/DREP. When this time was small enough, a source would continually send DREQ packets until such time as the DREP returned. In some cases, this resulted in hundreds of DREQ packets being flooded across the network before the destination could respond with a DREP. Until visualization with iNSpect was implemented into SENSE, the only way to identify this problem would have been by searching line by line through simulator output to find repeated packets. With visualization, it was simply a matter of watching a DREQ/DREP cycle, and seeing that nodes were flashing with the same received DREQ color multiple times. Once the problem was identified, a simple fix was implemented, in which nodes which wanted to resend a DREQ were forced to wait 10 packet transmission cycles. This still resulted in a few nodes sending multiple DREQ packets, but cut the number of DREQ/DREP packets drastically.

SENSE and iNSpect were also used to validate this modification to ensure that the protocol was now functioning as desired. In order to validate this change, the statistic printing function that was mentioned earlier was used. If the protocol is working at its absolutely optimum during DREQ/DREP, each node will see n-1 DREQ packets and n-1 DREP packets, where n is the total number of nodes in the network. This is due to the fact that both the DREQ and DREP are flooded across the entire network, and ideally, each source only sends one DREQ and each destination only sends one DREP. Before the modification to the protocol, the total number of DREQ packets was approximately 3n, while the number of DREP packets was approximately 1.5n, depending on the random seed used. After modification, the total number of DREQ packets was within n/10 of n and the number of DREP packets was exactly n for every seed tested.

5.3 Visualization to create variable λ

In addition to modifying the DREQ/DREP stage of SRP, iNSpect was also used to validate a modification to the selection of λ that the protocol uses. In the original design of all of the SSR protocols, λ was a fixed value that was selected at run time. Generally, we've maintained λ to be equal to 100ms. This was a fine solution, except for the fact that individual nodes have different numbers of neighbors. Those nodes with more neighbors may require a random delay range larger than 0 to 100ms to avoid collision, while those nodes with few neighbors may allow a significantly smaller range to be used. Initially, iNSpect was used to get a general idea of the high and low values of numbers of neighbors. Figure 5 is an iNSpect generated plot with the connectivity graph displayed on a randomly generated network of 75 nodes in a 1000m by 1000m space with a 230m transmission range. It is easily observed that there is a large variation in the number of neighbors that nodes have. Some nodes, such as node 26, have as few as 5 neighbors, while other nodes, such as node 47, have as many as 17 neighbors. This large variation in density results in a protocol that is not completely efficient with a fixed value of λ. In the example of node 47, 17 neighbors means that on average, there will only be a 5.8ms gap between delay selections, which could lead to a higher probability of collision. On the other hand, node 26 has only 5 neighbors, which means there will be an average 20ms gap between delay selections, resulting in a low probability of collision, but a fairly inefficient node. Variable λ corrects this variation.

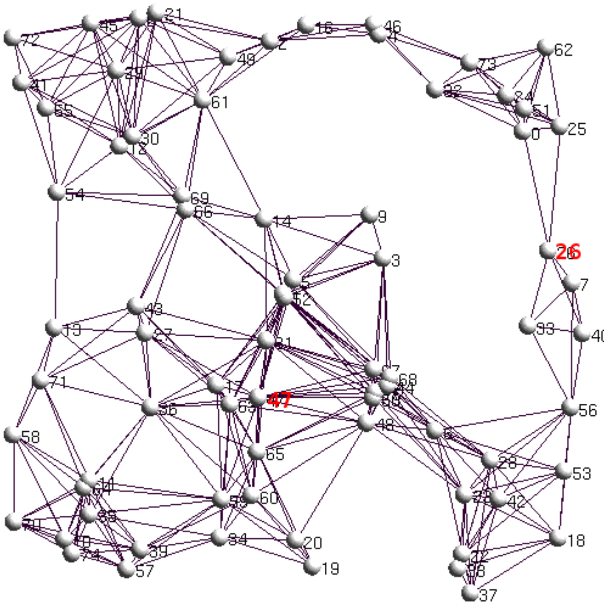
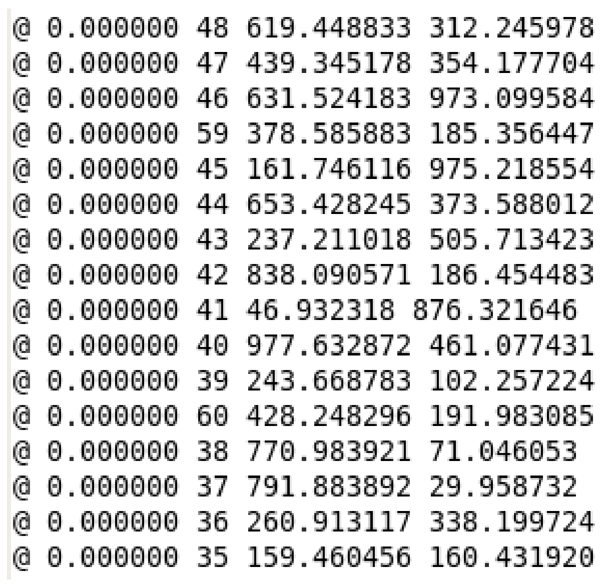


Figure 5: An example of the connectivity graph provided in iNSpect that was used to identify the variation in the number of neighbors throughout the network.

After validating that the number of neighbors varied as much as it does, a simple solution was devised in which each node calculates how many neighbors it has using the first DREQ sent across the network. Since every node will only forward a DREQ packet once, all that is required is to have each node count the number of times that it receives that packet. Upon completion of the initial DREQ packet's transmission, each node multiplies the number of neighbors by 15ms to generate its value for λ. Since the majority of networks that we work in have an average number of neighbors between 7 and 8, this puts the average value of λ between 105ms and 140ms. Although this value is greater than the original 100ms, because of the reduced number of collisions, it actually results in faster throughput with higher reliability.

While the varying number of neighbors is easily identifiable using iNSpect's Connectivity Graph feature, with SENSE alone, it would require a fair amount of output deciphering and math to calculate how many nodes are within the transmission range of each other. Figure 6 shows a sample of SENSE output that gives location information. Each line represents a different node at simulation time 0.0, and provides the node's identity, in addition to each node's location in the simulation field.



```
@ 0.000000 48 619.448833 312.245978
@ 0.000000 47 439.345178 354.177704
@ 0.000000 46 631.524183 973.099584
@ 0.000000 59 378.585883 185.356447
@ 0.000000 45 161.746116 975.218554
@ 0.000000 44 653.428245 373.588012
@ 0.000000 43 237.211018 505.713423
@ 0.000000 42 838.090571 186.454483
@ 0.000000 41 46.932318 876.321646
@ 0.000000 40 977.632872 461.077431
@ 0.000000 39 243.668783 102.257224
@ 0.000000 60 428.248296 191.983085
@ 0.000000 38 770.983921 71.046053
@ 0.000000 37 791.883892 29.958732
@ 0.000000 36 260.913117 338.199724
@ 0.000000 35 159.460456 160.431920
```

Figure 6: An example of the location output from SENSE.

6. CONCLUSION AND FUTURE WORKS

As was stated in the introduction, a picture is worth a thousand words. The combination of SENSE and iNSpect has provided us the ability to convert the pages of simulation output, into a clear and concise view of our networks. The animations presented through this combination have been invaluable in our continued development of the family of Self-Selective Routing protocols. They allow us to present the textual output of our simulations in such a manner that problems are easily identifiable, while still maintaining the basic ability to gather statistics and keep things simple. The ability to find a visualization tool that would work easily with our simulator allows us to focus our research efforts on improving the simulator and our protocols, while always having access to the state of the art in wireless network simulator visualization tools. We believe that the combination of these tools will continue to be very useful to us and other SENSE users all over the world in support of the research on wireless sensor networks. At the same time, the separation of the simulation tool from the visualization tools enables us and others (both tools are open source) to constantly improve each tool independently from each other.

In the future, we plan to expand our SSR protocols to include the capability to handle mobility and provide energy savings. Since iNSpect is created to handle mobile nodes, it will be an easy step to make to visualize a mobility protocol within SENSE. All that will be required in SENSE will be additional output into the mobility file which will provide node locations at times other than the beginning of the simulation. In addition, the output of SENSE is easily modified to allow a user to identify which nodes may be in a sleeping state, which are active and listening, and which are transmitting at any given time by changing node colors within iNSpect. The statistical collection capability within iNSpect will also provide the ability to easily gather power management information. These simple changes to the SENSE simulator will help to leverage the power within iNSpect and help to more easily move forward in the improvement of our protocols.

7. ACKNOWLEDGEMENT

The authors wish to acknowledge contributions of their colleagues at RPI, Gang Chen, Mark Lisee and Joel Branch (currently at the IBM Watson Research Laboratory) for their contributions to the early versions of SENSE implementations.

Research was sponsored by US Army Research laboratory and the UK Ministry of Defence and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the authors, and should not be interpreted as representing the official policies, either expressed or implied, of the US Army Research Laboratory, the U.S. Government, the UK Ministry of Defense, or the UK Government. The US and UK Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

8. REFERENCES

[1] T. Babbitt, C. Morrell, B.K. Szymanski, and J. Branch, *Self-selecting reliable path for wireless sensor network routing*, Computer Communication Journal **31(16)** (October, 2008), 3799–3809.

[2] L. Breslau, D. Estrin, K. Fall, S. Floyd, J. Heidemann, A. Halmy, P. Huang, S. McCanne, K. Varadhan, Y. Xu, and H. Yu, *Advances in network simulation*, IEEE Computer **33(4)** (2000), 59–67.

[3] G. Chen, J. Branch, M.J. Pflug, L. Zhu, and B. Szymanski, *Sense: A sensor network simulator*, Advances in Pervasive Computing and Networking (2004), 249–267.

[4] G. Chen and B.K. Szymanski, *Cost: A component-oriented discrete event simulator*, Proc. Winter Simulation Conference, WSC02 (San Diego, CA), vol. I, December 2002, pp. 776–7809.

[5] D. Estrin, Mark H, J. Heidemann, S. Mccanne, Y. Xu, and H. Yu, *Network visualization with the vint network animator nam*, Tech. report, University of Southern California, 1999.

[6] K. Fall and K. Varadhan (eds.), *The ns manual (formerly ns notes and documentation)*, The VINT Project, 2008, `http://nsnam.isi.edu/nsnam/index.php`.

[7] J. Glaser, D. Weber, S.A. Madani, and S. Mahlknecht, *Power aware simulation framework for wireless sensor networks and nodes*, EURASIP Journal on Embedded Systems **2008** (2008).

[8] C. Goldstein, S. Leisten, K. Stark, and A. Tickle, *Using a network simulation tool to engage students in active learning enhances their understanding of complex data communications concepts*, 7th Australasian conference on Computing education, Australian Computer Society, Darlinghurst, Australia, 2005, pp. 223–228.

[9] T.T. Huynh and C.S. Hong, *A novel hierarchical routing protocol for wireless sensor networks*, Mobile

Communications Workshop, LNCS, Springer, New York, NY, 2005, pp. 339–347.

[10] Chris Johnson, *Visualization viewpoints*, IEEE Computer Graphics and Applications (2004), 13–17.

[11] S. Kurkowski, T. Camp, and M. Colagrosso, *A visualization and analysis tool for wireless simulations: inspect*, ACM's Mobile Computing and Communications Review, to appear (2008).

[12] S. Kurkowski, T. Camp, N. Mushell, and M. Colagrosso, *A visualization and analysis tool for ns-2 wireless simulations: inspect*, Proceedings of the IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS) (2005), 503–506.

[13] H.K. Ryu, Y.Z. Cho D.H. Kim, K.W. Lee, and H.D. Park, *Improved handoff scheme for supporting network mobility in nested mobile networks*, Computational Science and Its Applications, LNCS, Springer, New York, NY, 2005, pp. 344–347.

[14] B.K. Szymanski and G.G. Chen, *Sensor network component based simulator*, Handbook of Dynamic System Modeling (Paul Fishwick, ed.), CRC/Taylor and Francis Publishing, 2007, pp. 35–1–35–16.

[15] B.K. Szymanski, C. Morrell, S.C. Geyik, and T. Babbitt, *Biologically inspired self selective routing with preferred path selection*, Bio-Inspired Computing and Communication, LNCS, to appear, Springer, New York, NY, 2008.

[16] H.L. Xuan and S. Lee, *Two energy-efficient routing algorithms for wireless sensor networks*, Networking, LNCS, Springer, New York, NY, 2005, pp. 698–705.