SimX: An Integrated Sensor Network Simulation and Evaluation Environment

Xiaogang Yang Mingsen Xu Paul Stickney Wen-Zhan Song
Sensorweb Research Laboratory, Washington State University
Email: {xiaogang_yang, mingsen_xu, pstickne, songwz}@wsu.edu

***Abstract*—It is well known that designing, testing and debugging sensor networks are extremely hard. It is mainly due to limited resources and distributed natures of sensor networks. In a lab environment, it is difficult and tedious to try various network configurations and simulate various challenging environment factors. Thereafter, the sensor network community has longed for a good sensor network simulation tool allowing the philosophy of WYSIWYG: What You See Is What You Get. In this paper, we present the design and evaluation of a visualized network manipulation tool integrated with TOSSIM and TOSSIM Live, called SimX. The SimX includes the following features: (1) topology manipulation, which allows a simple mouse drag on the virtual network node to actually change the physical network topology and the link qualities; (2) timing control, which allows a user to control a simulation to run faster or slower, even to pause the simulation; (3) variable watch and conditional breakpoints, which allow a user to watch variable value changes of all network nodes concurrently and set conditional breakpoints; (4) sensor input control, which allows a user to simulate different sensor data inputs and sampling rates. Besides supporting those traditional evaluation criteria, such as throughput, energy efficiency and delivery ratio, the SimX also supports a holistic evaluation methodology. The holistic evaluation methodology evaluates how well the *holistic* high-level spatial and temporal environmental changes have been recorded, not only those low-level engineering criteria. We also designed a XML-based sensor network visualization tool, which allows the user to illustrate any sensor network data, including data imported from TOSSIM and SimX.**

keyword: Wireless Sensor Network, Simulation, SimX, TOSSIM, XMonitor

I. Introduction

Maintaining and repairing a sensor network in remote challenging environments are extremely hard. Thus a comprehensive lab test is necessary before field deployment. However, testing and debugging sensor networks in a lab environment are also very hard. A joke teases that developing a sensor network system requires at least PhD level students, because the learning curve of sensor network programming is very sharp and debugging a distributed wireless network is even harder. Enabling trace statements could in fact change the system behaviors, as a sensor node has very limited resources. Also, it is difficult and tedious to try various network configurations and simulate various challenging environment factors. Thereafter, the sensor network community has longed for a good sensor network simulation tools with the following features:

1) The simulator shall allow users to run sensor node software without any modifications, at least above device driver layer. The NS-2 and Qualnet have been widely used in network community for simulations, however, simulation code can not run in sensor node directly. In the practice of system design, this is not ideal: firstly, it takes much time to port simulated codes to sensor node, which significantly increases the project delays; secondly, it could bring many unintended human errors during the software migration. A verified protocol in NS-2 may be not correct in node software after migrations.
2) The simulator shall have a manageable GUI with the design philosophy of **WYSIWYG**: What You See Is What You Get. WYSIWYG was used in computing to describe a system in which content displayed during editing appears very similar to the final output. In a sensor network simulator, we aim to simulate a virtual network in single PC and WYSIWYG means that "Whatever we can change in simulator through a KVM (Keyboard, Visual display unit, Mouse) simulates a physical change in a physical testbed". For example, if a user moves nodes in the simulated network topology, the simulator shall actually alter network connectivities and link qualities.
3) The simulator shall allow users to set network-wide breakpoints to trace the system dynamics. In our sensor network design practice, we frequently find that some subtle system problems are very hard to catch, because it does not happen often and we do not know how to repeat the cause. It would be really great if the time could suddenly pause so that we can look into systems states in every node. Certainly, it is impossible to do in a real testbed, but it could be feasible in simulators.

TOSSIM [1] was developed to simulate entire TinyOS applications, and works by replacing components with simulation implementations. The level at which components are replaced is flexible. The *greatest* feature of TOSSIM is that same high-level source code can work in both real sensor network testbed and TOSSIM. This feature greatly relieves the development overhead in system research and meets the first preferred feature summarized above. TOSSIM Live is an extension of TOSSIM (of TinyOS 2.x) to enable virtual tiered networks. It has two main parts: a simulation throttle and a serial forwarder. The simulation throttle allows the simulation clock to emulate the wall clock, whereas the serial forwarder allows serial messages to be sent to and received from the virtual motes.

In this paper, we designed a set of visualized network manipulation and evaluation tools, building upon TOSSIM and TOSSIM Live. It will be called SimX in this paper. The SimX tool includes the following features:

1) Topology manipulation: it will allow a user to change the simulated network's physical connectivity. A simple mouse drag on the virtual sensor node will actually change the physical network topology and the link qualities. It will also allow a user to turn on or off the virtual sensor node. Moreover, if the node software changes its RF power or channel, then it can be reflected in the network connectivity and topology in SimX, which makes SimX a powerful tool to approximate and manipulate the real world network topology. Those features enable the simulation of network mobility, topology control and disruptive environments.
2) Timing control: it will allow a user to control a simulation to run faster or slower, and even to pause the simulation process. It enables simulation time to get synchronized with the real time. For instance, we set sampling rate as 1Hz; SimX can simulate each sensor node to execute the sampling exactly once per second in real time fashion.
3) Variable watch and conditional breakpoints: it will allow a user to watch variable value changes of all nodes concurrently during simulation running time. It also allows a user to set a conditional breakpoint. For example, pause the simulation when a variable $x > 100$. This friendly user interface can facilitate the software developing.
4) Sensor data input: it will allow a user to simulate a sensor's data input by various sources. For example, a sensor input can come from a local file or a math function (e.g., a sine function to generate sinewave as input). Moreover, it can also simulate different sampling rates in different nodes. For instance, it can tune down the sampling rate in an individual node, in which case the sensor may lose some fidelity of source data.

Besides supporting those traditional evaluation criteria, such as throughput, energy efficiency and delivery ratio, the SimX also supports a holistic evaluation methodology. It views a sensor network as a "camera" which records the environmental changes in space and time, and each sensor is like an image sensor in camera. The holistic evaluation methodology will evaluate how well the *holistic* high-level spatial and temporal environmental changes have been recorded, not just those low-level engineering criteria. To implement such an innovative evaluation methodology, we will inject a image stream to TOSSIM through SimX, such that each sensor records a portion of each scene and delivers to a gateway. The gateway then assembles received data to a image file. We will then evaluate the relative quality of the generated image, comparing to the original one. In addition, we designed an integrated XML-based sensor network visualization tool with SimX, which allows to illustrate data from any testbed or TOSSIM application without changing source code.

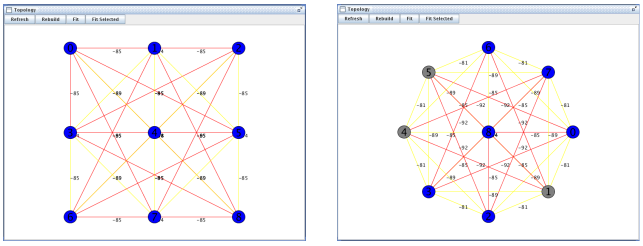
The rest of this paper is organized as follows. Section II describes the system model and formulates the problem. Section III introduces XMonitor. Then section IV illustrates how to use SimX and what benefits it brought to the TinyOS application development. Section V reviews related work on debugging tools for TOSSIM. Finally, Section VI concludes the paper and discusses future work.

II. SimX Features and System Design

SimX is an integrated simulation and evaluation environment. It bridges the gap between users and their simulators, and facilitates the designing, testing and debugging in distributed sensor networks. SimX provides developers with more flexibilities to manipulate simulation and evaluate applications. The design and implementation of SimX components will be presented in this section.

A. Topology Manipulation

The topology of sensor network varies over time, mainly due to the radio channel's variance. Other environmental factors contributing to the topology changes are: the radio interference, mobility of sensor nodes and sensor nodes' lifetime. Moreover, network topology can also be altered by intentional topology control to save power and increase the network lifetime. The specific behavior of the wireless link depends on two elements: the radio channel, and the environment where they are placed. Therefore, topology manipulation, which faithfully simulates both two elements, is essential for a simulator to approximate network behavior.



(a) Topology illustration (b) Topology manipulation

Fig. 1. Topology illustration and manipulation

SimX fulfills the above functionalities by illustrating and manipulating a network topology in run-time manner, shown in Figure 1. The topology manipulation component in SimX, visualizes the network based on the information of relative node positions and link quality from TOSSIM. Each node acts as an object in the simulation program, which has the desired attributes, including status of node indicating its aliveness, radio gain, node ID, etc. Virtual positions generated by algorithm reflect relative positions between every two nodes. The color of nodes and edges can indicate that the node activity and the link quality respectively.

Figure 2 shows the mechanism of topology manipulation. The core of topology manipulation is Dynamic Topology (DynTopo) and TxPower. DynTopo manages the topology information. And TxPower rebuilds the simulation model if

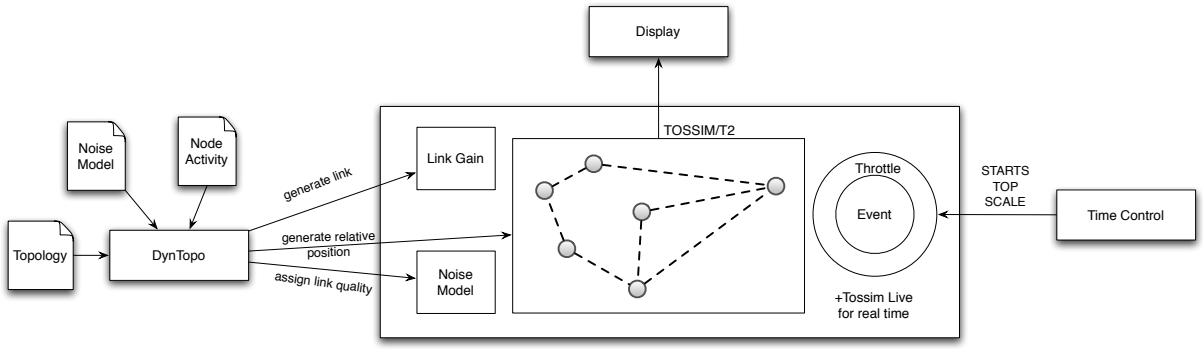


Fig. 2. Topology control and timing control

it detects a power change. In SimX, transmission power and distance between nodes determine the link quality. Topology manipulation in SimX hides complex low-level operations from users and interacts with users through friendly graphic interfaces. Users can interact with the simulation through this component. If the user drags the node with a mouse to a different position in the simulator, the link quality will be re-calculated according to the new distance. Meanwhile, in Figure 1, users can stop and start any node in TOSSIM through this component in the running time as well.

Besides that, the topology manipulation component can also assist developers in testing the performance of designed protocol stack in dynamic sensor network and comparing the performances of different protocols. For instance, if users want to evaluate the performance of CTP [2], the intuitive approach is to apply this topology manipulation to test its self-forming and self-healing performance besides other communication metrics. Users can conveniently simulate the disruptive scenario where nodes die or move out of range. With this support, many evaluations of networking protocol in simulator become realistic; for instance, how fast can one node find a new parent when it lost its old parent, what is the robustness of its self-forming algorithm and etc,.

B. Timing Control

Timing control in simulation includes two elements: synchronizing the simulation events to the real time, and controlling the execution pace of events while keeping them in order.

First, the TSync component of SimX enables simulation synchronized to the real time. TOSSIM keeps time at mote instruction clock cycle granularity: $4MHz$. And every TOSSIM event occurs at a virtual time and runs instantly. This allows a globally temporal ordering of events occurring, but it lacks of synchronization to the real time. TSync improves the time-throttle support provided by the TOSSIM Live extensions [3]. TSync actively examines the forward event queue, so that it has a higher time accuracy than the TOSSIM Live extension. Peeking forward results in a more accurate synchronization approach, especially as the frequency of events decreases. Second, it is significant that simulation speed and execution time can be controlled and tuned according to users' intention. Timing control component of SimX addresses this problem. In Figure 3, users could see the *stop*, *start* and *scale*, *run until* buttons. These buttons are used to stop the simulation, start it, change its speed or set the time limit to run it. Figure 2 illustrates the mechanism behind it. In TOSSIM, there exists a "Main Loop" to control the simulation, where TOSSIM events result in the series of TinyOS events and commands comprising an entire application. Therefore, the appropriate approach of controlling the simulation is to control "Main Loop". For Speed Control, SimX sets different delays before running the next step. If the simulation has been stopped, SimX will pause the "while loop" until *start* triggers the loop. If the user has specified *run until* parameter, SimX will run until the specified time expired.

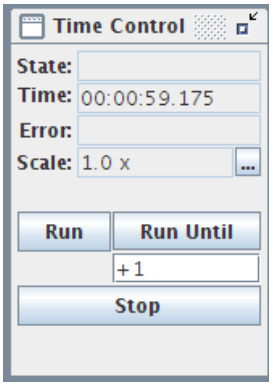


Fig. 3. Timing control

C. Variable Watch and Conditional Breakpoints

Variable watch and conditional breakpoint in those IDE (Integrated Development Environment) tools have greatly eased the application development. However, the distributed embedded system lacks integrated debugging environment, and suffers from the difficulty of monitoring the runtime details in real time. Therefore, users developing the application in distributed system long for an integrated test and debug tool. Motivated by this need, SimX supports a useful component, called variable watch and conditional breakpoints, by taking advantage of interfaces provided by TOSSIM. First, variable watch component displays any variable's value in watch list, and can synchronize them automatically. The variables can be either primitive type data or elements of array and structure. In order to provide users with more convenience, SimX also allows them to select variables from drop-down list of

variables, then displays selected variables for all the nodes concurrently, as shown in Figure 4. Variable watch monitors runtime details in an illustrative way so that developers can easily control the simulation.

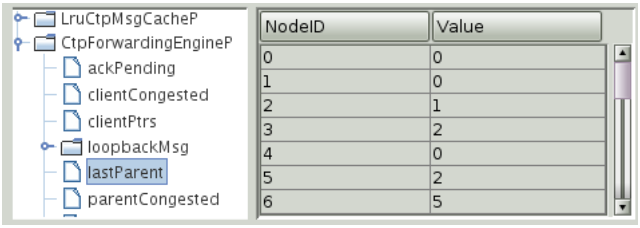


Fig. 4. Variable watch list

Second, conditional breakpoint enables interuption of running applications as the set conditions has been satisfied. The motivation is that instantaneous events in the networks are still hard to capture, although the variable watch enables users to monitor the variables of all the nodes concurrently. During the interruption, users inspect the test environment to find out whether application functions as expected. Users can set up multiple conditional breakpoints to pause the simulation as well. And, the conditional breakpoints help control the simulation efficiently by setting up one or multiple conditions. The conditions are defined by users' concerns, which can be variable sanity check, variable value comparison and etc,.

SimX improves TOSSIM memory-read interfaces to uphold the inspection of variables. The key component is PROBE, which serves as an alternative of TOSSIM Python getValue interface. As illustrated in Figure 5, PROBE can access raw data from nesC component without TOSSIM interfaces. Thus, PROBE can circumvent the complex data structure in TOSSIM filter.

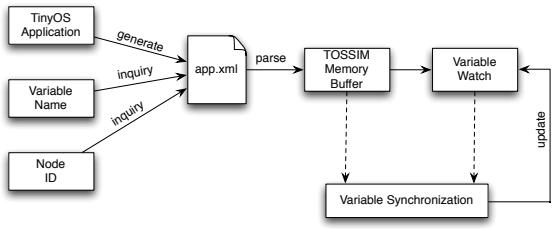


Fig. 5. Variable watch: PROBE

In addition to the variable polling as required by users, variable watch also has the functionality of synchronizing the watch list with the running simulator. Once any bit changed, disparity event can trigger PROBE to retrieve the data and update variable values.

D. Sensor Data Input

Researchers perceive and interpret the simulation behavior and performance based on the input and output of this "black box". And the sensor data as an input stream plays a fairly important role in creating different sensing scenarios for sensor networks. For instance, in the previous work of in-network data compression [4] [5] , different data patterns can affect the performance of compression and aggregation greatly. In order to make the comparison between simulation and testbed fair, it is significant to simulate the real data patterns to establish authentic input streams. However, the data streaming component in TOSSIM can only generate several simple data patterns such as sine wave and triangle wave. It is insufficient to simulate the real data by these simple patterns.

This motivates SimX to support any data input by creating a generic sensor component. In Figure 6, SimX sensors substitute the sensors in TOSSIM with its own sensor components to simulate the real data input. Sensor component can give users flexibilities in two aspects: first, to interpret various input data from the format of SAC (Seismic Analysis Code), binary format and text file source, so that this enables in-network processing evaluation by injecting archived data set. Second, users can also create their own stream reader to support any data format. Thus, SimX is able to generate miscellaneous data patterns by efficiently managing the sensor array and creating a generic sensing interface. Additionally, SimX can simulate multiple sampling rates for any data input. In the sensor network with sufficient capacity, sampling rate will determine the data quality. The more data is collected for a phenomenon during a time period, the more precise the image can be represented by fusion center.

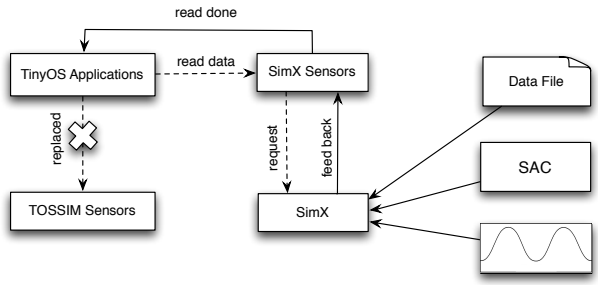


Fig. 6. SimX sensor component

Illustrated in Figure 6, sensor components substitute the TOSSIM sensors, and run independently to generate data. The sensor component of SimX establishes a generic interface and effective queue management for the array of sensors to read data from either data file or mathematical function output. In addition, SimX supports different sampling rates by controlling the streamer source.

III. XMonitor Design

Toward an integrated simulation and evaluation environment, we have presented the design and evaluation of a visualized network manipulation tool, SimX. The exhibited architectures and components support the effectiveness and efficiency of SimX in establishing and controlling simulation. And furthermore SimX also supports a holistic evaluation methodology. As a complete tool chain, SimX necessitates a compatible visualization tool to display and evaluate the raw data. Motivated by that, we design a XML-based sensor network visualization tool, namely XMonitor.

XMonitor is a sensor network visualization and evaluation tool designed to illustrate any sensor network's data from either testbed or simulator. XMonitor can evaluate and envision the engineering criteria, such as network throughput, energy efficiency and delivery ratio. More interesting is that it can also represent the data imaged in high-level spatial and temporal environmental fashion, which will be demonstrated in section IV-C. And it is comprised of three components: Topology Graph, Raw Data Display, and Waveform Sensing Data. Topology Graph exhibits the relative topology graph of the network. The connection edges among nodes reflect the relationship of child and parent pairwise. Besides network topology, individual node information, such as accumulated number of packets received, data rate, packet loss ratio and energy consumption, can be displayed as well. Raw Data Display lists raw messages in the original packet format faithfully. If the message can be recognized, then they could be parsed and labeled to different parts based on fields of message format. Waveform Sensing Data plots the data from different nodes and channels into real time wave form. XMonitor is designed to be user centric; for instance, it applies the packet filter to render the data users concern, and it can keep the topology layouts for different exhibitions.

The essential feature of XMonitor is to run independently from TinyOS. Any type of message could be decoded only based on the XML file, provided by users, without doing the code-level changes to XMonitor. The XML file has three parts: Message Hierarchy, Message Format and Data Format. Message Hierarchy is related to hierarchical structure of received message. And XMonitor needs first to identify the received message based on the structure description. Message Format includes message field, size and offset information, which are used to help XMonitor decide where to get correct information. After identification and localization of the incoming messages, XMonitor decodes the data payload according to the Data Format. With this generic XML based decoding mechanism, XMonitor can provide an integrated environment for users to visualize and evaluate different types of sensor data from TinyOS application.

IV. SimX Usages and Case Studies

In this section, we demonstrate how to use the four key features of SimX to control the sensor network simulation: runtime control, topology manipulation, sensor input control, variable watch and conditional breakpoint control. In the following case study of SimX, we used Octopus [6], an example application in the TinyOS-2.x. Octopus samples the sensors periodically; nodes packetize the data and send them back to the gateway along the routing path. The underneath routing protocol is CTP. And gateway collects all the data from network and illustrate them in XMonitor.

A. Runtime Control

SimX provides users with the timing control component, described in section II-B, to control the simulation speed. Two cases are examined for runtime control: setting countdown clock and tuning the running speed of simulation.

First, countdown clock can help users run the simulation subject to a time epoch. For example, users could click the $run\ until$ button to set the running time period of simulation. If $run\ until\ 5s$ is set, the simulation will stop after 5 seconds. Otherwise, the simulation will run until next stop operation. This countdown clock can provide users with a flexible running time management.

Second, users desire to be able to tune the running speed of simulation. With the help of the timing control component, shown in Figure 3, users can decelerate the simulation to take a deeper inspection of the sensor network. And also they can accelerate the simulation to skip some tedious repeat steps. When the $stop$ button is clicked, data rate displayed in XMonitor can be observed to become lower and lower, until it reaches 0. And then all the nodes in XMonitor Topology Graph are frozen. In this way, users can temporarily pause the simulation and pull out the intermediate results they want. $Scale$ Button is used to control the speed of simulation. If the value of Scale is set as 1.0x, the simulation run in normal speed. If it is set to 4.0x, data rate becomes 4 times higher, indicating that the simulation accelerates.

B. Topology Manipulation

As a case study for topology manipulation, we examine the CTP routing protocol. We will know whether CTP works under different circumstances, and how well it performs. In Topology Panel, default nodes' positions are in grids roughly, the $Sink\ Node\ 0$ is in the left corner of the grid as shown in Figure 7. Then we could see that other nodes reach the $Sink\ Node$ within one hop or more. We simulate several scenarios. Firstly, we turn off $node\ 2$ which is the current parent of $node\ 9$. Then we can observe that $node\ 9$ looses its parent after a few seconds, and switches to a new parent, which has the best link quality among its neighbors. This proves that CTP works and changes topology as designed. Secondly, we relaunch $node\ 2$, and move it close to $node\ 9$, and find that $node\ 9$ connects to $node\ 2$ again, due to the better link quality. Thirdly, we test the topology under different deployment topologies. We deploy the nodes in a ring topology with the $Sink\ node\ 0$ in the center. In this way, most of the nodes connected to it within one hop, which is also validated by XMonitor. Finally, we simulate the poor link quality of $node\ 9$ by moving it away from the window. Due to the dependence of CTP on the estimated link quality, $node\ 9$ eventually loses its connection with the network. Through the studied cases, we can get an intuitive validation of CTP functionality. And then it is safe to further examine the performance of CTP routing protocol under more comprehensive tests.

C. Sensor Data Input

To study the scenario of sensor data input, we adopt the Octopus application to collect data from sensor networks. And in the following test, original sensors in TOSSIM are replaced by SimX sensor component which supports reading data from

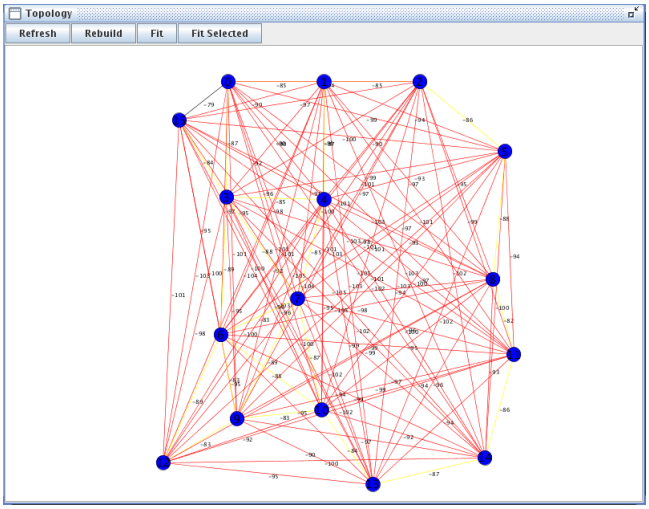


Fig. 7. Topology Control Component

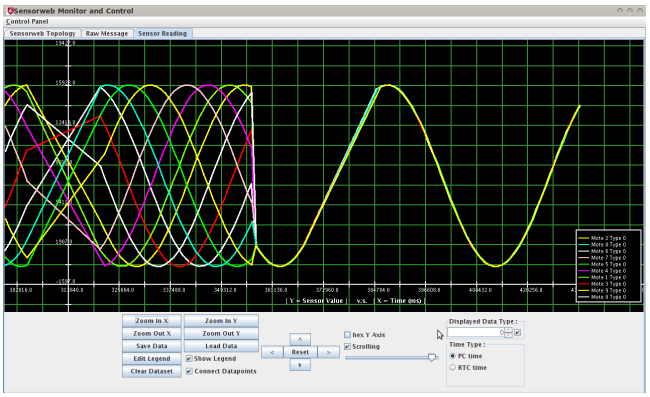


Fig. 8. Data input of synchronized sinewave

SimX stream source in the fidelity of 8 bit, 16 bit and 32 bit respectively. This experiment is carried out along three axes: first, different stream sources alter in any node at any time; second, holistic methodology, a new evaluation methodology for sensing data; third, different sampling rates are examined to explore the tradeoff between traffic load and data quality.

1) Different Sources of Data Input: SimX allows the users to alter sensor data input anytime in any node without affecting other nodes. First, we inject sine waves with different offsets to each node, and we remove all these offsets and make them run with same phase in each node. Based on the collecting application of Octopus, we can find that the waveforms promptly respond to this phase change as indicated in Figure 8. This feature can guarantee that the input data is synchronized with each other. And then data can be further exploited to scrutinize the network time synchronization. For instance, we inject the waveforms without offset, and let the nodes run based on their local clocks. All the local clocks are synchronized to a global network clock by the designed time synchronization algorithm. If the algorithm works well, there should be no offset among different output data streams.

Moreover, the ability to alter the sensor input at will also facilitates many experiments, requiring tedious repeat tests. For example, we want to find out the best compression coefficients for different data patterns. We can inject diverse data patterns to each sensor, and retrieve the different results accordingly. Bundling up the similar simulations alleviates the efforts involved and promotes the productivity.

2) Holistic Evaluation Methodology: Traditionally, the low-level engineering evaluation criteria, such as throughput, energy efficiency and delivery ratio, is used to compare different approaches. However, those are not really what an end user want. An end user is only interested with the holistic data quality, in other words, how faithfully a sensor network captures the environmental changes. For example, if approach A has higher delivery ratio than approach B, however, but approach B captures more global changes than approach A, then B is still preferred than A. A holistic evaluation methodology views sensor network as a "camera" which records environmental changes in space and time, and each sensor is like an image sensor in camera. In this case, it can evaluate how well the *holistic* high-level spatial and temporal environmental changes have been recorded, not just those low-level engineering criteria.

SimX supports such an innovative evaluation methodology. For example, a user can customize a stream reader, which reads and splits a gray scale image to N sensor nodes in TOSSIM, such that each sensor records and delivers a portion of the image to a gateway. Once the gateway receives the data, it starts to reassemble the image. In our XMonitor tool, the reconstruction of the original image is witnessed as continuous plotting pixel by pixel. After the delivery of whole image is done, we evaluate the relative quality of represented images.

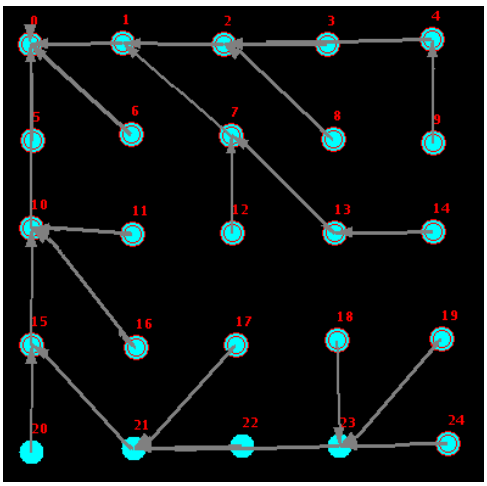
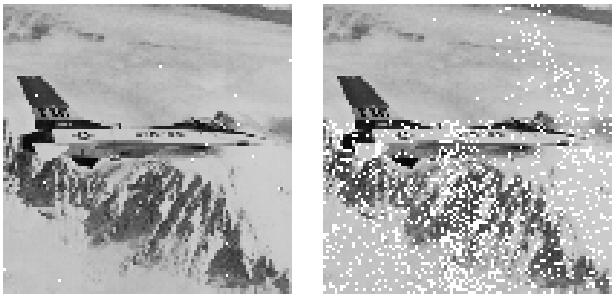


Fig. 9. Data collection tree of the 5×5 grid layout. Node 0 on top-left corner is the sink

In Figure 10, we carry out different tests under two different topologies. First one is a grid topology, in which $Sink\ Node$ is placed in the left upper corner of the grid. The grid is formed by 5×5 array, in which many nodes reach $Sink\ Node$ in multi-hop, whose topology is shown in Figure 9. The second



(a) Ring topology with 25 nodes (b) Grid topology with 25 nodes

Fig. 10. Image representative quality based on different topologies

one is a ring layout with $Sink\ Node$ in the center. Other 24 nodes are evenly placed on a ring around the $Sink\ Node$, in which most connections are within one hop. Each node senses its own portion of the image and transmits the sampled data.

We can find that the image delivered through ring layout is better than that of grid layout. The "white spots" which degrades the image quality are observed in images represented under both ring and grid topology. Compared with the grid topology, "white spots" observed under ring topology are much less, and spread randomly in the image. Whereas, there are more white spots observed under the grid topology. And the white spots area becomes denser as the distance between sensor node and sink node increases. Holistically, these white spots denote that packets have been lost during multi-hop transmission. And the more hops the node requires to deliver back its sensing data, the higher packets lost it will experience.

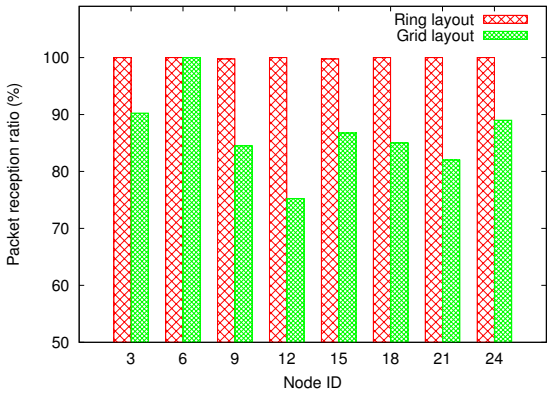


Fig. 11. Packet Reception Ratio comparison between different topologies

In addition, we also generate low-level performance analysis graphs to validate this holistic evaluation. In Figure 11, generally the nodes in Grid topology have lower Packet Reception Ratio (PRR) than those of Ring topology. Referring to the Figure 9, we can find some interesting results in Figure 11. Centered at node 0, we draw a series of concentric circles are drawn with their radius increased by unit length. The node inside the smaller circle has a better PRR than the ones outside that circle. For instance, node 6, which is within the smaller circle than the node 3, experience 100% PRR, while node 3 only has 90% of PRR. And node 9 has even bigger radius than the node 3, resulting in its 85% of PRR. There are also some exceptions to this observation, such as node 12 and 24. It is because that they take some other nodes with much smaller radius as their routing parents.

3) Different Sampling Rates: As stated in section II-D, SimX can also simulate the sampling rate differences. In this case study, we vary the sampling rate in the range of $1Hz - 10Hz$. Our goal is to examine how the sampling rate affects the data quality, and also to find out the minimum sampling rate enabling users to recognize the image without difficulty. For different sampling rates, we show the results in Figure 12. Under the same ring topology with no packet loss, $10Hz$ represents the image faithfully. And $3Hz$ sampling rate can depict the image roughly while $1Hz$ can hardly capture the image's outline.

(a) Sample rate 10 Hz (b) Sample rate 3 Hz (b) Sample rate 1 Hz

Fig. 12. Holistic evaluation methodology with different sampling rates

D. Variable Watch and Conditional Breakpoint

As an integrated simulation tool, SimX is entitled to debug a running application. Besides the cases studied in the section IV-A, variable watch also plays an important role in retrieving the running information from simulator and conduct the debugging. For instance, one node in Octopus application stops delivering data packets to gateway leaving no obvious clues. We first doubt whether its timer still works or is inadvertently stopped by some exceptions. To clear our doubts, we put the counter (incremented by 1 every time timer fired) into watch list. The automatic update of watch list can enable us to check if the counter keeps increasing. Our doubt of timer issues can be safely removed if the counter value was not frozen.

And if we suspect the sending queue is stuck, we can put the variables of queue size and header pointer into the watch list. The same principle applies when we suspect the problem is due to the transmission failure. And similarly, we can put the current parent of this node along with its hop count to the variable watch list.

Moreover, conditional breakpoint can be helpful when users can not predict the network behavior. The conditional breakpoint will help localize the phenomenon when it happens. For instance, when users simulate a disruptive network with unpredictable network congestion or nodes failure, they have no idea when to get the critical information from simulator. Due to the unpredictable behavior, users can not control them by mapping the pause command to a specific time slot. So, if users miss the critical moment to retrieve the important information from simulation, it will take them much more time to repeat the experiments. What makes it worse is that

some simulations take a long time to run until the desired behavior happens. Therefore, conditional breakpoints can be leveraged to control the simulation at the crucial point, such as connection disruption.

V. Related Work

TinyOS is an operating system designed for wireless embedded sensor networks [7]. TinyOS presents all hardware platforms with different components. All the communication among components, including commands and events, are realized by specifying interfaces. TinyOS tries to maximize its flexibility, ignoring platform support, application construction, and reliability. TinyOS-2.x is a second generation sensor network operating system written in the nesC language. It increases its reliability, decreases its overhead, simplifies application construction, and makes porting to new platforms easier. While TinyOS and nesC are about creating reusable components, TinyOS-2.x is about composing components and building reliable applications. Due to the unified hardware abstraction, the high-level application can be ported to any platform without major changes. Thereafter, researchers can circumvent the hardness of debugging distributed sensor network, by examining them first in simulator, such as TOSSIM.

As one of the prevailing simulators [8] for TinyOS, TOSSIM has the same mechanism as TinyOS, only translating the platform related components into simulation components. TOSSIM can support thousands of TinyOS motes running with handy maintenance. Hardware interruptions are translated into discrete simulator events. TOSSIM could add link quality and topology into the nodes, which makes simulation more realistic. The simulator engine enables reading and writing global values in the motes, and collecting data by the serial forwarder tool. Due to TOSSIM's excellent features of controlling and monitoring, it becomes a prevailing tool to simulate sensor network. However, the event driven mechanism of TOSSIM does not always reflect real situation, such as the time synchronization issue. Interaction between TOSSIM and user needs to be improved for more convenient operation.

YETI [9] is a NesC editor plugin for Eclipse. It could help the developer to fix syntax error, highlight keyword, validate code in real-time fashion and complement codes automatically. It could support GDB and generate wiring graph of tinyOS program [10]. This is an important step for visualizing TinyOS program developing. There is another toolkit for visualizing the Runtime Behavior of TinyOS Applications [11], which is similar to YETI and also moves one step further in visualizing programming. However, it does not enable the manipulation of application and interaction between simulation and developers. It also needs more user friendly interfaces, which can get information and control the simulation runtime.

TinyViz [1] is a visualizing extension tool to TOSSIM/T1. It is a Java-based GUI for TOSSIM/T1, allowing simulation to be visualized, controlled, and analyzed. The core engine of TinyViz, by itself, generally manages the event/command interfaces to TOSSIM. It provides plugins: Debug messages which could show the debug information generated by simulation. Setting breakpoint allows simulation to be paused when some conditions satisfy. Radio allows people to change the connectivity through location GUI. However, it is only designed for TOSSIM in TinyOS-1.x, and still needs more plugins to support TinyOS-2.x.

VI. Conclusion and Future work

In this paper, we have presented SimX, an integrated simulation and evaluation environment for sensor networks, following the principle of **WYSIWYG**. It bridges the gap between complicated TOSSIM interfaces and developers. In our design, this visualized simulation tool manipulates and interacts with the simulator by four components: topology manipulation, timing control, variable watch and sensor input control. The unified graph interfaces of SimX provide users flexible control on simulation. We also showed that SimX can evaluate not only the low-level network performance, but also the high-level holistic evaluation. We believe SimX can greatly benefit sensor network research and TinyOS community, as it gives users visualized flexible control on sensor network test and evaluation.

In the future, we plan to extend SimX to control a real sensor network testbed based on WYSIWYG philosophy. Basically, user can use a single integrated development environment to turn on/off or move sensor nodes, change the sensor input, or pause a network as same time for debugging purpose.

References

[1] P. Levis, N. Lee, M. Welsh, D. Culler, Tossim: Accurate and scalable simulation of entire tinyos applications, in: Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys)., 2003.

[2] O. Gnawali, R. Fonseca, K. Jamieson, P. Levis, CTP: Robust and efficient collection through control and data plane integration, 2008.

[3] C. S. Metcalf, Tossim live:towards a testbed in a thread, Master's thesis.

[4] A. Kiely, M. Xu, W.-Z. Song, R. Huang, B. Shirazi, Adaptive linear filtering compression on realtime sensor networks, in: The 7th Annual IEEE International Conference on Pervasive Computing and Communications (PerCom), 2009.

[5] S. Pattem, G. Shen, Y. Chen, B. Krishnamachari, A. Ortega, Senzip: An architecture for distributed en-route compression in wireless sensor networks, in: ESSA, 2009.

[6] A. Ruzzelli, G. OHare, V. Roy, A. Barbirato, S. Boivineau, M. Dragone, R. Jurdak, C. Muldoon, Octopus: A dashboard for sensor networks visual control, 2009.

[7] P. Levis, S. Madden, D. Gay, J. Polastre, R. Szewczyk, A. Woo, E. Brewer, D. Culler, [duplicate] the emergence of networking abstractions and techniques in tinyos, in: Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation, USENIX Association, Berkeley, CA, USA, 2004, p. 1.

[8] J. Pan, A survey of network simulation tools: Current status and future developments, Tech. rep. (November 2008).

[9] N. Burri, R. Schuler, R. Wattenhofer, Yeti: A tinyos plug-in for eclipse, in: REALWSN, 2006.

[10] P. Levis, TinyOS Programming (June 2006).

[11] A. R. Dalton, J. O. Hallstrom, A toolkit for visualizing the runtime behavior of tinyos applications, in: The 16th IEEE International Conference on Program Comprehension, 2008, 2008, pp. 43–52.