

# A simple visualization and programming framework for wireless sensor networks: PROVIZ<sup>☆</sup>



Shruthi Ravichandran<sup>b</sup>, Ramalingam K. Chandrasekar<sup>b</sup>, A. Selcuk Uluagac<sup>a,\*</sup>,  
Raheem Beyah<sup>b</sup>

<sup>a</sup>Electrical and Computer Engineering Department, Florida International University, 10555 West Flagler St, Miami, Florida, 33174, USA

<sup>b</sup>Communications Assurance and Performance Group, School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, Georgia, 30332, USA

## ARTICLE INFO

### Article history:

Received 5 January 2016

Revised 19 May 2016

Accepted 27 June 2016

Available online 24 August 2016

### Keywords:

Wireless sensor networks

Visualization

Programming

Simulation

PROVIZ

## ABSTRACT

Wireless Sensor Networks (WSNs) are rapidly gaining popularity in various critical domains like health care, critical infrastructure, and climate monitoring, where application builders have diversified development needs for programming, visualization, and simulation tools. However, these tools are designed as separate stand-alone applications. To avoid the complexity of using multiple tools, we have designed a new extensible, multi-platform, scalable, and open-source framework called PROVIZ. PROVIZ is an integrated visualization and programming framework with the following features: PROVIZ includes (1) a visualization tool that can visualize heterogeneous WSN traffic (with different packet payload formats) by parsing the data received either from a packet sniffer (e.g., a sensor-based sniffer or a commercial TI SmartRF 802.15.4 packet sniffer) or from a simulator (e.g., OMNeT); (2) a scripting language based on the TinyOS sensor network platform that aims at reducing code size and improving programming efficacy; (3) an over-the-air programming tool to securely program sensor nodes; (4) a visual programming tool with basic sensor drag-and-drop modules for generating simple WSN programs; and (5) a visual network comparison tool that analyzes packet traces of two networks to generate a juxtaposed visual comparison of contrasting network characteristics. PROVIZ also includes built-in extensible visual demo deployment capabilities that allow users to quickly craft network scenarios and share them with other users. In this work, we introduce the various features of PROVIZ's visualization and programming framework, analyze test scenarios, and discuss how all the tools can be used in sync with each other to create an all-encompassing development and test environment.

© 2016 Elsevier B.V. All rights reserved.

## 1. Introduction

Recent advances in sensors and wireless technology have led to the proliferation of wireless sensor networks in health, structural, and environmental control and monitoring systems. Such diverse use cases result in a broad spectrum of WSN users, from advanced WSN application developers, who develop scalable and reliable WSN algorithms and applications, to intermediate developers, who use WSNs for simple environment data gathering, to novices and young students, who have just started experimenting with sensors. This diverse set of usage scenarios necessitate a WSN software programming framework with both advanced and simple easy-to-use tools.

The PROVIZ software framework introduced in this work, is one such open-source, platform-independent, and scalable framework that aims to serve the needs of the diverse user base with the following features. PROVIZ supports:

- visualization of live and pre-captured traffic from a variety of real sensors (e.g., MicaZ, Iris, TelosB) including the ability to work with multiple sniffers simultaneously in a distributed setup to visualize a large WSN deployment. It can also visualize packet trace data generated from a WSN simulator (e.g. OMNeT [1]).
- a set of built-in tools for developing sensor applications, programming sensor nodes over-the-air [2], and simulating in OMNeT.
- built-in extensible visual demo deployment capabilities, which allow users to quickly craft network scenarios and export them as XML files to easily modify or share with others.

<sup>☆</sup> This work is supported by the National Science Foundation, under grant NSF-ACI-1339781.

\* Corresponding author.

E-mail address: [selcuk@gatech.edu](mailto:selcuk@gatech.edu) (A. Selcuk Uluagac).

- a network discovery module to identify the sensors available in a WSN and an optional thin-client in sensor nodes to send sensor state information periodically. With the help of these periodic reports, users of PROVIZ can easily identify a network anomaly and can take necessary corrective measures for avoiding failures in the WSN.

One of PROVIZ's main features is the visualization of live, pre-captured or simulated WSN data. There are many visualization tools for WSNs that are either real-time visualizers [3,4] or visualizers built with WSN simulators [5,6]. These tools, albeit very useful, do not offer offline visualization and lack the ability to deliver useful network statistics to the user such as packet load in the network, average delay between nodes, etc. The visualization and network comparison tools introduced within PROVIZ aim at closing this gap by visualizing real-time, offline or simulated WSNs (all from one platform) and allowing users to compare two WSNs and visually verify differences in the performance of the two networks in terms of link characteristics.

PROVIZ also aims to make TinyOS [7] programming easier for novices interested in implementing simple sensor programming. TinyOS is one of the most widely used sensor network platforms to develop and program a myriad of applications in many different sectors [8,9]. Despite its large user base, novice application developers face a very steep learning curve in writing TinyOS applications in nesC [10] to perform basic data collection and distribution. A lot of systems that provide application-specific modules as add-ons to the main WSN program have been developed [11,12]. These modules implement the most commonly used sensor network applications like data collection, link estimation, etc., that can be added as single line function calls to the main program. However, they lack the flexibility of a simple scripting language that allows users (even novices) to experiment with their code and develop something beyond these add-on modules. Hence, the scripting tool developed in PROVIZ concentrates on this feature while still being relatively easy to use even for novices.

Moreover, efforts to introduce programming to middle or high schoolers [13,14] also strengthen our motivation for PROVIZ to provide a similar user-friendly graphical user interface (GUI) for sensor network application development. The afore-mentioned tools are extremely user friendly that they can be used to teach children programming concepts such as loops and if-else statements in a very simple graphical user environment. Nevertheless, there are not many tools that attempt to do the same with wireless sensors. Indeed, sensors are very attractive to young minds with their multi-color LEDs, light and temperature sensors, and radio interface. With these tangible interfaces, one can teach the students similar programming concepts, but with readily observable outputs in the LEDs or through RF packets that can be visualized from packet traces. Hence, the visual programming tool for WSNs aims to offer a simple GUI for K-12 students to interact easily with sensors and progress rapidly into programming sensors with the scripting language.

In summary, in this work we introduce the PROVIZ framework; its main advantage is that it is a complete integrated system where visualization, programming, and simulation can be done from the same software architecture as opposed to other similar useful works in the literature. The remainder of this paper proceeds as follows. We first discuss the related work in Section 2. In Section 3, we discuss the PROVIZ architecture and each of its modules. We elaborate on the visualization tool features in Section 4. Section 5 provides an overview of the network comparison tool. In Section 6, we describe the TinyOS scripting language and provide examples for the same. We analyze the visual programming tool and its supported operations in Section 7 and then discuss a use-

age scenario for PROVIZ in Section 8. We conclude the paper and discuss future work in Section 10.

## 2. Related work

The relevant literature to our work can be subdivided into works on WSN visualization and analysis tools, scripting languages for sensor network programming, and graphical programming of WSNs.

### 2.1. WSN visualization and analysis

When deploying multiple sensor networks, it might be imperative to debug the deployments, obtain network characteristics, and compare them for differences and irregularities. For instance, this debugging capability will help a developer to compare two WSN deployments based on link characteristics obtained from packet traces.

There are a lot of useful tools that visualize and monitor the WSN and obtain network properties such as topology, packet loss, etc. The following are some of the relevant tools developed in this area. MOTE-VIEW [15] is a user interface software developed by MEMSIC [16] for monitoring and visualizing a WSN deployment. It acts as a health monitoring tool that queries a database server (populated by packets from nodes) to retrieve node status and configuration of all nodes in the network. It is also used for visualizing node qualities such as throughput, bandwidth, link quality, and congestion (obtained from server) in color codes in a topology chart. Livenet [17] is a tool developed to analyze a sensor network reconstructed from packet traces. The tool uses multiple packet sniffers and merges all the packet traces to get a more detailed account of the WSN behavior. From this merged packet trace, network characteristics such as topology, bandwidth usage and packet loss are analyzed and plotted. SNIF [18] and Sympathy [19] are tools developed for network debugging and fault finding within the WSN. While Sympathy debugs control information sent by sensors along with regular traffic to a sink node, SNIF uses a deployment support network [11], which is a network of secondary sensors, used specifically for debugging to find faults in the network. NetViewer [3] is a real-time visualization and monitoring tool. It collects data from a sink node and based on packet format specified by user, parses the packets, visualizes the packets, displays payload information, and demonstrates topology of the network. Other than the topology, this work does not provide any other important network detail to the user. Octopus [4] is also a WSN monitoring, visualization, and a control tool. Besides visualizing the WSN in real-time, Octopus can also reconfigure the sensor network by sending out short control messages. Code for this tool has to be installed in the sensor along with the sensor network program to avail its functions or, the program and packet structure have to be developed according to Octopus specifications. NetTopo [5] is an integrated simulation and visualization framework which is designed for validating algorithms used in WSNs. NetTopo is essentially a simulator which is integrated with a real testbed to run the WSN algorithms and validate them. It is able to visualize the topology used in the simulation environment and it updates the information about connections, sensed data of each node, etc. However, NetTopo is specifically designed for simulating and validating algorithms and hence it cannot do live or off-line WSN visualization and also cannot program a sensor node. TOSSIM [6] is a WSN simulator, which can simulate a homogeneous wireless sensor network with sensors running a common application. It provides a visualization tool called TinyViz [6], which can be used to visualize, control, and analyze TOSSIM simulations. TinyViz can visualize the sensor nodes and the network traffic in a WSN. It provides

information about the data transferred between nodes, serial communication packets, LED states, and provides a control window to change the simulation parameters. However, TOSSIM is a simulator and cannot do off-line visualization or interact with real sensor nodes. Although, the tools discussed above are efficient monitoring and/or fault detection tools for WSNs, they are essentially different from the PROVIZ software framework that they do not provide specific network characteristics to the user such as average size of packet, average packet delay, etc. nor have the capacity to compare two networks.

## 2.2. WSN scripting languages

TinyOS [7] is a widely used operating system for sensors. It uses nesC [10], a language based on C for developing WSN applications. Although nesC is very well established and can be used to write many complex applications, the process is still very tedious involving writing multiple files (configuration and implementation files) and many lines of code. Hence, the following works have developed various scripting languages to reduce the complexity of developing applications in nesC.

SNACK [12] is a kit consisting of a configuration language, a service library and a compiler for programming WSNs. With SNACK, WSN users spend minimal time to put together an application by using its in-built service library that consists of commonly used functions such as routing, sensing, etc. The compiler will compile this application written in the configuration language to nesC. The syntax of the configuration language is still hard for novices and SNACK does not give users the flexibility to add user-defined functions to its library or reuse program blocks like the scripting tool in PROVIZ. Ceu [20] is another higher level programming language for C and nesC that reduces program size and the complexity of developing applications in TinyOS. However, despite reducing code size and programming time considerably, the scripting language developed is not very user-friendly and requires quite a bit of familiarizing. The language is not part of an integrated framework like PROVIZ where code can also be developed and simulated. Additionally, it does not implement TinyOS concepts such as modularity thereby, relinquishing the code reuse property of TinyOS modules and interfaces. DSN [21] is a declarative language and compiler for fast and efficient sensor network programming. Similar to SNACK, it also offers a library of add-on modules to be added to programs written in Snlog, a dialect of Datalog [22]. The DSN compiler compiles this program to configuration and module files in nesC. Since the DSN is based on a data querying language, novices again, face a steep learning curve. The scripting languages discussed above, despite having shorter code sizes compared to nesC require some time to comprehend. They also do not support the code reuse property (modularity) of TinyOS. The scripting tool in PROVIZ tries to keep the code size to a minimum and at the same time ensures flexibility for users to develop their own code and reuse them across different applications and files.

## 2.3. WSN visual development

The relevant visual development tools for WSNs can be broadly classified into application development environments and visual programming tools. Application development environments are GUIs for creating TinyOS configuration files by visually wiring component and module files from the TinyOS library, while visual programming tools use visual blocks and icons for creating sensor network programs.

Viptos [23] is a graphical application development tool for TinyOS-based WSNs. It allows users to develop TinyOS programs by constructing block and arrow diagrams of TinyOS components. This is to remove the programming complexity of locating and

wiring various components from the TinyOS library. TOSDev [24] is another application development tool for TinyOS. Like Viptos, this tool also offers graphical editing of wiring diagrams along with a source code editor similar to Eclipse [25]. RaPTEX [26] is an integrated code development, simulation, and emulation environment that offers TinyOS protocol stack development with wiring and component connection diagrams similar to Viptos. It allows creation of topology and simulation of network level performance in OMNeT and emulation of node-level behavior using Avrora [27] as the underlying-emulator. With this integrated environment, the tool allows easy customization of existing TinyOS protocol suites and acts as a platform for testing nesC applications before deploying to a WSN. PROVIZ is different from RaPTEX that PROVIZ offers additional tools for post-deployment analysis of a WSN such as packet trace visualization and comparison and a simpler scripting-based programming environment for novice users. WISDOM [28] is a platform-independent visual programming tool that uses a modular approach to programming. Users can create modules in C, independent of platforms, connect them graphically with connectors and the tool will generate platform-specific code from the modules. However, this work is not very well documented. It does not reveal how sensor-relevant code (e.g., timer, radio, and LED function calls) are declared in C. SEAL [29] offers a visual programming and standard programming language (as an alternative to nesC) for WSNs. The visual programming language abstracts the SEAL programming language using Google Blockly [30]. Although, the programming language is easy for novices, by steering away from TinyOS, it loses some advantages offered by TinyOS such as split-phase operations and code reuse. TinyInventor [31] is a visual programming tool that adopts the OpenBlocks [32] visual programming language to create a TinyOS relevant programming environment for WSNs. The programs are a collection of functional blocks created by users with drag and drop components. The blocks are compiled to generate nesC code for sensors. The tool is simple and easy to use and is similar to Scratch that also uses OpenBlocks. However, the main disadvantage is that the tool is not part of an integrated system like PROVIZ where visualization, programming, and simulation can be done from the same system. Moreover, the visual programming tools discussed above are simple to use. However, they lack the pictorially descriptive drag-and-drop modules that the visualization tool developed in our work offers and are not part of an integrated framework like PROVIZ where programs can be built, simulated or programmed and visualized, all from a single environment.

## 3. PROVIZ design overview

PROVIZ is an integrated multi-platform, modular, generic, extensible, and scalable programming and visualization framework [33] developed using the QT GUI Framework [34]. QT is a C++ based, open-source and a cross-platform application and user interface development framework supporting most of the popular desktop and mobile operating systems. Fig. 1 shows the PROVIZ framework design, where PROVIZ runs in a host machine and has a *PROVIZ Client*, which can run in a local or remote host and gather packet data using different sniffers. As shown in Fig. 1, PROVIZ has a modular architecture, which enables the user to replace or modify a module, without affecting the overall system behavior. PROVIZ provides two main functionalities. One is the functionality to visualize the data and the other is the functionality to develop applications and program sensor nodes.

### 3.1. Modules for programming functionality

This sub-section discusses the design of modules for the programming functionality.

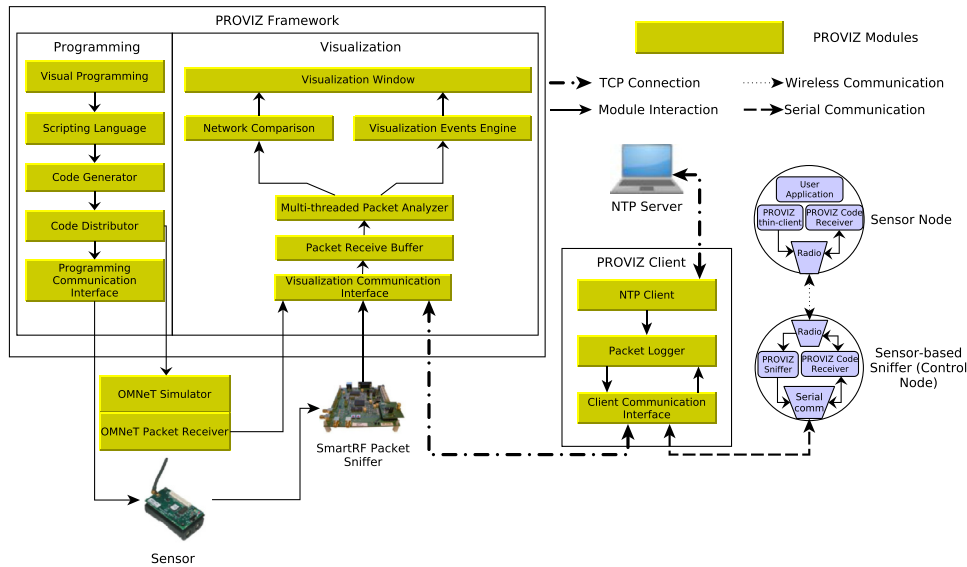


Fig. 1. PROVIZ architecture.

### 3.1.1. Visual programming

The visual programming tool has rows of actions or functions for each sensor. Basic input (light, temperature sensor), output (LEDS, Radio) and operation (On, Off) modules can be dragged and dropped onto these rows to create actions. Each sensor is given a node ID and once all actions are created, the user can drag and drop the sensor onto a canvas and connect to the other sensors via radio as required. Code is generated in the scripting language for each sensor.

### 3.1.2. Scripting language

The scripting language inherits the best of TinyOS concepts such as modularity and split-phase operations and at the same time is simple enough for a novice programmer to develop programs for data collection, monitoring and distribution. The scripting language uses simple C/C++ abstractions of TinyOS concepts. Programs are written in an editor window, which is integrated with the visualization window in such a way that the user can switch between the tools easily by a single mouse click. Details of this language are discussed in Section 6.

### 3.1.3. Code generator

After entering the script in the editor window, the user can use the *Generate Code* option in PROVIZ to generate the nesC code from the scripting language.

### 3.1.4. Code distributor

After generating the code, the user can modify or change the implementation details in the nesC code and use the *Program WSN* option to generate the WSN application binary. It uses the *Communication Interface* to remotely program wireless sensor nodes using wireless code dissemination. Or, in the case of resource unavailability, the framework interfaces with the OMNeT simulator [1] to simulate the WSN and users can verify program functionality from the output generated by OMNet.

### 3.1.5. Communication interface

After the Code Distributor generates the nesC code image, the Communication Interface is utilized to disseminate the code image.

### 3.1.6. Wireless code dissemination

For wireless code dissemination, PROVIZ uses the SIMAGE protocol [2]. SIMAGE uses the LQI value to assess the link quality between the nodes and then dynamically adapts the packet size. In

a WSN having nodes with poor link quality, the dynamic packet resizing technique reduces the number of retransmissions during code dissemination. Along with dynamic packet resizing, SIMAGE also provides energy efficient security services, confidentiality and integrity provided by the CC2420 [35] transceiver module in sensors (e.g., MicaZ, TelosB, etc.).

For programming a WSN, PROVIZ uses the Control Node (i.e., the node connected to the computer) as a base station for disseminating the code image. All the nodes in the WSN along with the user application should have a *Code Receiver* module, which can receive the disseminated code image to program the sensor node and transfer it to the neighboring nodes. The Control Node is also utilized in the visualization functionality as a sniffer as explained in the next sub-section.

## 3.2. Modules for visualization functionality

This sub-section discusses the design of modules for the visualization functionality.

### 3.2.1. Communication interface

This interface has a generic packet receive module for the visualization functionality, which can receive packets from different sources: 1) a PROVIZ Client with a sensor-based sniffer attached to either a remote or a local host, 2) WSN simulators such as OMNeT [1], and 3) a packet trace file, generated by commercial sniffers, e.g., the PSD [36] file format created by the TI SmartRF packet sniffer [36].

### 3.2.2. OMNeT Packet Receiver

While simulating the WSN application in the OMNeT simulator, the OMNeT radio receive driver outputs the time-stamped packet trace. The OMNeT Packet Receiver receives the network simulator trace and sends it to the Communication Interface.

### 3.2.3. Packet Receiver buffer

The Communication Interface receives the packet data and stores the data in the form of a byte-array in the Packet Receiver Buffer queue, which can be fetched later for processing.

### 3.2.4. Multi-Threaded packet analyzer

While capturing the packets, PROVIZ initiates a multi-threaded module, which parses the packet data stored in the Packet Receive



Buffer queue. It parses for the packet arrival time, IEEE 802.15.4 header information [37], and the packet payload and matches it with the user-defined packet formats. Since the Packet Analyzer has to parse all the incoming packets and translate them, it becomes a computationally intensive operation. In order to achieve faster processing and avoid any performance bottlenecks, the Packet Analyzer module is designed in a multi-threaded fashion. The number of threads running in the Packet Analyzer module is a configurable parameter, which can be modified by the users based on the incoming packet arrival rate. Whenever the application starts, based on the value in the thread count parameter, PROVIZ creates sufficient amount of worker threads and forms a thread-pool. So, whenever the user starts a visualization, the worker threads from the thread-pool is used to parse the packets, which eliminates the delay in thread creation.

### 3.2.5. PROVIZ visualization events engine

After parsing the packets, the Packet Analyzer module creates a packet transfer event and adds it to the *PROVIZ Visualization Events Engine*, a multi-map data structure with the time of event as the key. The Events Engine multi-map includes all the events in sequence, sorted based on the time of the event. In order to make the Visualization Engine extensible, PROVIZ defines an abstract class with event handler methods as pure virtual functions. So, if users want to extend PROVIZ with new features, they simply need to inherit the abstract class and define the event handler methods, which can implement features like node mobility, battery-level monitoring, etc. The new modules can add event objects in the Events Engine multi-map and these event handlers will be called by the Events Engine at the appropriate time. When the visualization starts, the time of the first event in the event queue is considered as the current visualization time and the Events Engine starts a periodic timer. Whenever the timer is triggered, the current visualization time is incremented and the events that are to be executed at this current visualization time are identified in the multi-map and they are triggered by calling their event handlers.

### 3.2.6. Network comparison

If the user would like to compare two traces, the network comparison tool gets the parsed packet data from the Packet Analyzer and computes the differences between two networks for pre-determined link characteristics (e.g. average packet delay, number of packets sent). On comparing the two traces, the tool informs the user if the networks match based on a threshold matching percentage. The tool also shows the two network topologies next to each other with missing and extra links (with respect to the first network) indicated and the link properties of each link are displayed when the link is right-clicked.

For visualization, PROVIZ either shows two networks in comparison or in the case of visualization of a single network, provides a graphical canvas (Fig. 4) for visualizing the sensor nodes and packet transfers. Also, it includes a *PROVIZ Packet Data Display Window* (Fig. 6), which shows the parsed packet information in a user readable format. The details of the visualization tool are articulated in Section 4.

### 3.3. PROVIZ client design

PROVIZ provides a *PROVIZ Client* application running in local or multiple remote machines for gathering WSN packets. The Control Node is connected to these remote/local machines running the PROVIZ Client. This node works as a sniffer for gathering WSN traffic while also serving as a base station for disseminating the code image over-the-air.

*Client Communication Interface*: The Client Communication Interface receives the sniffed packets from the Control Node and the

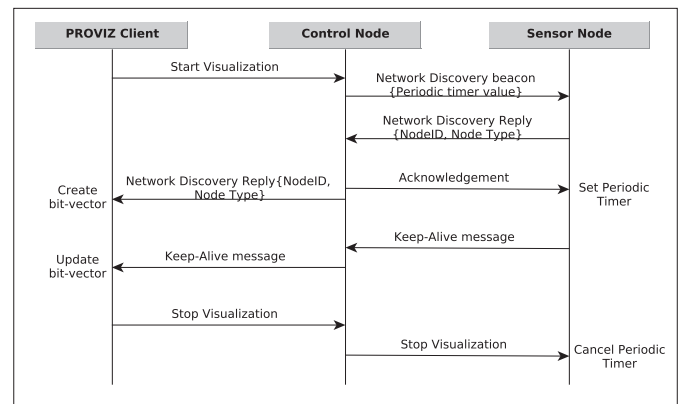


Fig. 2. PROVIZ network discovery module.

*PROVIZ Packet Logger* module adds a time-stamp to the packet data. The time-stamped packet data is sent to the visualization host over the network using a TCP socket.

*Network Time Protocol client*: To synchronize the data collection and visualization in a distributed environment, the PROVIZ Clients are time-synchronized with the help of the Network Time Protocol (NTP) [38]. The NTP Client requests the global time from the NTP server and then synchronizes the system time based on the server's reply.

#### 3.3.1. Thin-client and network discovery

PROVIZ provides an optional thin-client module that resides in sensor nodes along with user programs. PROVIZ also includes a Network Discovery process in the PROVIZ Client, which is used to find the number of sensor nodes available in a WSN and their types. Fig. 2 illustrates the procedure of network discovery in PROVIZ. When the PROVIZ Client starts, it triggers the Control Node to send a network discovery broadcast message. All the sensor nodes running the *PROVIZ thin-client*, on receiving this message, respond back to the Control Node with a specific message. This message includes the ID and the type (e.g., MicaZ, TelosB) of the sensor. After receiving the reply, the Control Node sends an acknowledgment and sends the received data to the PROVIZ Client. The PROVIZ Client identifies the unique Node IDs and creates a *Keep-Alive* bit-vector of appropriate size. Also, it reports the number of sensor nodes and their types to the PROVIZ visualization host. The thin-client, after getting an acknowledgment from the Control Node, starts sending a periodic (e.g., every 10 seconds) keep-alive message to the Control Node. The periodic interval in which the keep-alive message has to be transmitted, is sent with the network discovery broadcast message. The PROVIZ Client, on receiving a keep-alive message, updates the Keep-Alive bit-vector based on the source Node ID. At the end of the periodic interval, the PROVIZ Client checks the bit-vector for nodes which did not send a keep-alive message in that round and then clears the bit-vector before proceeding to the next round. If a node fails to send keep-alive messages continuously for a specified number of rounds, the PROVIZ Client communicates the node IDs to the visualization host, which in turn notifies the user. Then, PROVIZ decrements the number of live sensors that can be used for visualization. When the user closes the PROVIZ Client, it triggers the Control Node to send a broadcast message to stop sending the periodic keep-alive messages. The PROVIZ Client can do the network discovery for WSNs without thin-client, by analyzing the sniffed packets and creating a bit-vector based on this information. However, in this case PROVIZ cannot get the sensor state information. Thus, PROVIZ can still work with WSNs without thin-client, but with reduced functionalities.

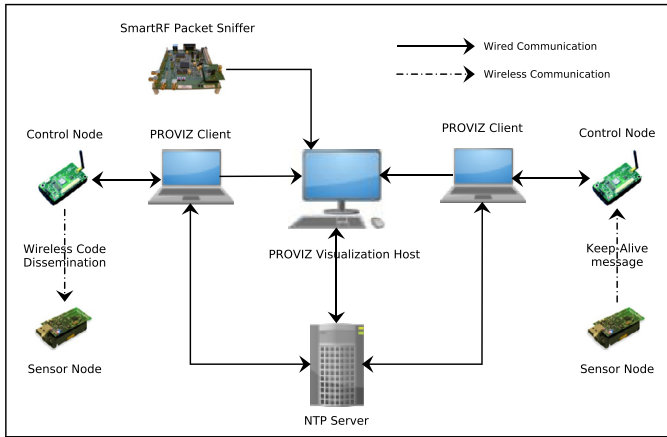


Fig. 3. PROVIZ distributed framework model.

### 3.4. PROVIZ distributed framework model

For visualizing a large WSN, PROVIZ uses a distributed approach that uses multiple sniffers as shown in Fig. 3. In the distributed setup, the PROVIZ visualization tool runs in a host machine and multiple sensor-based sniffers are placed in a distributed fashion so that a packet transmitted by a sensor node can be sniffed by at least one of the sniffers. The PROVIZ Client program runs in a time-synchronized remote or local host connected to a sniffer and sends the time-stamped packet data to the visualization host for visualization, utilizing the NTP protocol as explained in Section 3.3. In a distributed setup, some nodes will be in the sniffing range of multiple sniffers and hence, PROVIZ can receive duplicates of a same packet. However, since the PROVIZ Clients are time-synchronized, PROVIZ Events Engine can identify and remove the duplicates by validating the time-stamp and packet data.

## 4. Visualization tool features

This section discusses the various features of PROVIZ visualization functionality.

Fig. 4 shows the screen shot of the PROVIZ visualization tool with a group of infrastructure monitoring sensors sending structural health information [9] periodically to a cluster head. The PROVIZ Visualization User Interface window has a: 1) *Control toolbar*, which provides the control buttons for the visualization; 2) *Drag and Drop Window Holder*, which has sub-windows to hold the images of sensor nodes that are available in a WSN and to hold the demo application icons; and 3) *Graphical Work Area*, a canvas where the sensor node images are placed and the packet transfers are visualized.

### 4.1. Control toolbar

The Control toolbar provides control options to start, pause, and stop the visualization. Also, it has a support to zoom-in, zoom-out, and clear the nodes in the Graphical Work Area.

### 4.2. Drag and Drop Window Holder

The Drag and Drop Window Holder has multiple sub-windows and these sub-windows can hold either sensor node icons or icons associated with demo applications. The sensor icons displayed in the sub-window include an image of sensor node and a count associated with them. The count depicts the number of sensors of that type available in a WSN and it is determined by the PROVIZ Network Discovery module (Section 3.3.1) in the PROVIZ Client.

PROVIZ uses this count value associated with each sensor icon to control the number of sensors that can be visualized. In order to visualize a WSN, the sensor icons in the sub-windows can be dragged and dropped into the Graphical Work Area. Each node that is dropped needs to be associated with a unique Node ID, which is displayed along with the sensor icon. Also, a signal strength meter showing the signal strength of a node is displayed at the top of each sensor node icon. The signal level value is received from sensors in real time as explained below.

### 4.3. Packet visualization

The Control Node sniffs the packets complying with IEEE 802.15.4 and transmits the header and payload information of packets to the PROVIZ Client through serial communication. The Control Node, having CC2420 transceiver module [35], can estimate the wireless link quality and give a Link Quality Indicator (LQI) value. The sniffer program in the CC2420 transceiver fetches the LQI and RSSI value information from the CC2420 transceiver and appends it to the packet payload before sending the sniffed packet data to the PROVIZ Client. The PROVIZ Client sends this data to the visualization host and the multi-threaded Packet Analyzer (Section 3.2.4) in the visualization host parse the packets for LQI value and associate it with the packet transfer event, which is enqueued in the PROVIZ Visualization Events Engine (Section 3.2.5). When the packet transfer event handler is triggered, the signal strength meter of corresponding node is updated with the new value. On receiving the packets from the serial interface of Control Node, the PROVIZ Client prepends the time-stamp to the packet data and sends it to the visualization host. This packet information format starting with the time-stamp, followed by packet data, RSSI and LQI is also used by the popular TI SmartRF packet sniffer [36], which ensures that PROVIZ is generic and extensible. This feature is also utilized to visualize the packet transfers simulated by the OMNeT simulator, with the help of *OMNeT Packet Receiver*. For this, the TinyOS communication driver for the OMNeT simulator was modified to output the packet information whenever a node receives a packet and the OMNeT simulator creates a packet trace with this time-stamped packet information. The OMNeT Packet Receiver module reads the packet information and converts it to the SmartRF packet sniffer packet format, which can be parsed by the PROVIZ Packet Analyzer.

### 4.4. Heterogeneous WSN visualization

Most WSNs are heterogeneous with different types of sensor nodes sending packets with different packet payload formats. PROVIZ is capable of visualizing such a heterogeneous sensor network and it can understand different packet formats. In order to select multiple packet formats for visualization, PROVIZ provides a *Packet Format Selector* window, which lists the user-defined packet formats from which the users can select the packet formats and a feature to define the packet format graphically using the *PROVIZ Packet Format Specifier* as shown in Fig. 5. For defining a packet format, PROVIZ expects the users to start the packet format with a unique message ID, which is used by the PROVIZ Packet Analyzer to identify and match the packet data to a particular packet format. Then, the user can add multiple packet fields and specify the data-type for the field and the packet field name for each of the packet fields. After defining the packet format, PROVIZ expects the users to select a packet animation color using the color selection tool. The packets matching this format are visualized with the color specified by the user. The user can then save the color information along with the packet format details in an XML file, which can be exported or shared with other PROVIZ users. Also, users can define the packet format in the form of XML directly and give that

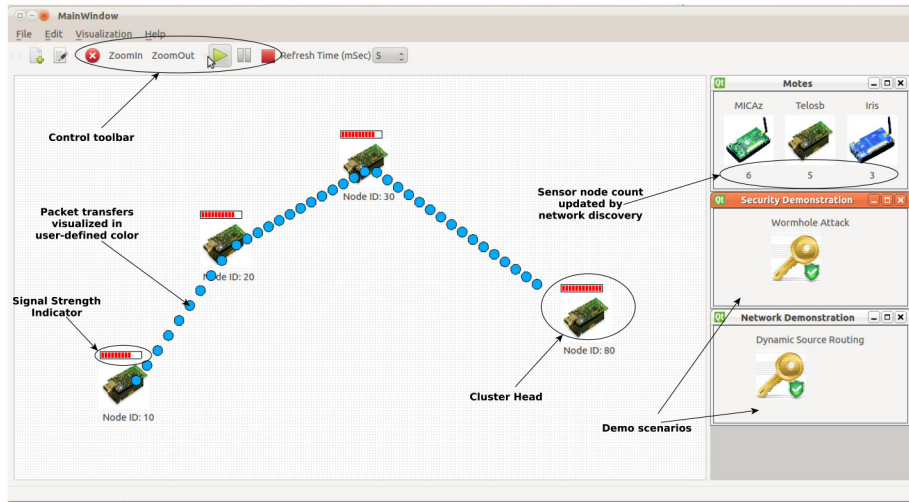


Fig. 4. Screen shot of PROVIZ visualizing a infrastructure monitoring WSN.

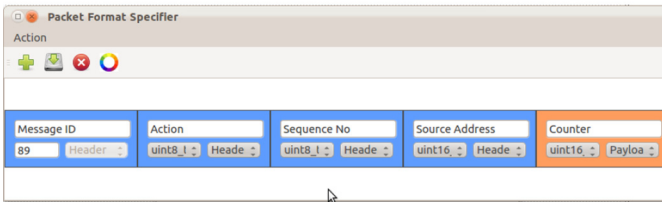
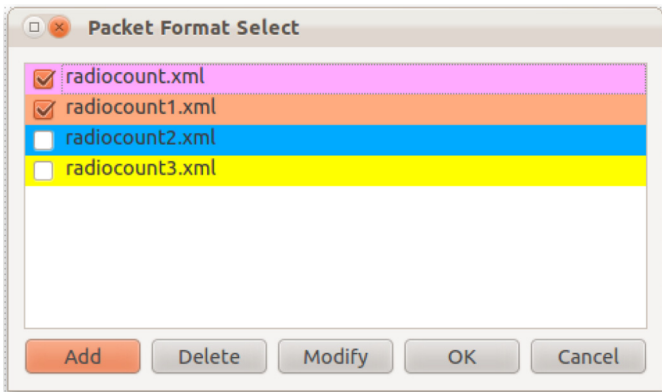


Fig. 5. PROVIZ packet format selector and specifier.

```

<Packet>
  <msgID>90</msgID>
  <Color>
    <r>0</r>
    <g>170</g>
    <b>0</b>
  </Color>
  <Header index="1">
    <Description>Action</Description>
    <Datatype>uint8_t</Datatype>
  </Header>
  <Header index="2">
    <Description>Sequence No</Description>
    <Datatype>uint8_t</Datatype>
  </Header>
</Packet>
    
```

Listing 1. Packet format XML.

as input to PROVIZ. A sample packet format XML snippet is shown in Listing 1.

By default, the Packet Analyzer can understand and translate the IEEE 802.15.4 header information [37] in the input packet

data. After interpreting the header information, the Packet Analyzer matches the raw packet payload data with a user-defined packet payload format based on the message ID value. After finding a match, the Packet Analyzer extracts the necessary bytes and type-casts it to the data-type mentioned in the packet payload format. Similarly, the rest of the packet data is type-casted to the appropriate data-types specified by the packet fields in the packet payload format and it is displayed along with the packet field description in the *PROVIZ Data Display* (Fig. 6). If the Packet Analyzer cannot find a match with a packet format, it displays the hex value of the payload in the Data Display window. Fig. 6 has the translated packet information of packets matching the packet payload format shown in Fig. 5 and also shows the hex value of

Time(mSec)	MSG Type	Data	Seq no	Destination PAN ID	Destination ID	Source ID	AM Type	MSG ID	Action	Sequence No
166	MSG Type: Data		103	8704	1000	254	6	89	5	
435	MSG Type: Data		109	8704	254	1000	6	89	5	0 5a 5 0 0 0 1b 6e eb ea
679	MSG Type: Data		110	8704	254	1000	6	89	5	0 5a 5 0 0 0 1b 6f eb eb
655	MSG Type: Data		105	8704	1000	254	6	89	5	
195	MSG Type: Data		108	8704	254	1000	6	89	5	0 5a 5 0 0 0 1b 6d eb e9
898	MSG Type: Data		106	8704	1000	254	6	89	5	
409	MSG Type: Data		104	8704	1000	254	6	89	5	
922	MSG Type: Data		111	8704	254	1000	6	89	5	0 5a 5 0 0 0 1b 70 eb eb
1142	MSG Type: Data		107	8704	1000	254	6	89	5	
1170	MSG Type: Data		112	8704	254	1000	6	89	5	0 5a 5 0 0 0 1b 71 eb ec
1379	MSG Type: Data		108	8704	1000	254	6	89	5	
1413	MSG Type: Data		113	8704	1000	254	6	89	5	0 5a 5 0 0 0 1b 72 ea eb
1625	MSG Type: Data		109	8704	1000	254	6	89	5	

Fig. 6. Screen shot of PROVIZ packet display window.



```

<Example_Scene>
  <Name>Packet Transfer Example</Name>
  <Node>
    <Node_Count>1</Node_Count>
    <Mote Node.ID="0">
      <Position>
        <x>-400</x>
        <y>0</y>
      </Position>
      <Description>Basestation</Description>
      <Node_Type>Telosb</Node_Type>
      <Mote_Image_Path>images/telosb.jpg
    </Mote_Image_Path>
    </Mote>
  </Node>
  <Simulate>
    <Frame time="100">
      <Start>0</Start>
      <End>10</End>
      <Color>
        <r>0</r>
        <g>0</g>
        <b>255</b>
      </Color>
      <SignalStrength>100</SignalStrength>
    </Frame>
  </Simulate>
</Example_Scene>

```

**Listing 2.** Demo scenario visualization XML.

the packet payload, whose format does not match with any of the user-defined packet payload format.

*Demo scenario visualization:* PROVIZ includes built-in extensible visual demo deployment scenarios, on which users can click and visualize easily as shown in Fig. 4. Users can use this feature to create a demo scenario to visualize a critical/complex WSN deployment and share it with other PROVIZ users. The demo visualization can be developed by creating an XML file, which specifies the node type to be used, number of nodes, node ID, node location, and the time, when the packet transfers should happen. To define a packet transfer, each transfer is considered as a frame, which has the information such as the node IDs between which the transfer should happen and the time when the transfer should be initiated. A sample demo visualization XML snippet is shown in Listing 2.

## 5. Network comparison tool

The network comparison tool is a tool designed to compare two networks based on their respective link characteristics. The tool can be used by researchers who would like to compare two current deployments or two past deployments. The tool can also be employed in the education of programming concepts in the middle or high school levels. The educators can assign homework to students to program a WSN according to specifications. These programming assignments can be created by expert programmers along with *solution* packet traces. This eliminates the involvement of educators who might not be well versed with programming and WSN concepts. The tool can then be used to compare the *solution* packet trace of the prescribed WSN and the trace from the WSN simulated or programmed by students. Students can be graded by percentage similarity to the *solution* network and can even themselves visually identify and learn the differences in the two WSNs.

### 5.1. Design

The network comparison tool extracts packet information (arrival time, the IEEE 802.15.4 header [37], and the packet payload) from PSD trace files generated by a packet sniffer (e.g., TI SmartRF Packer Sniffer). The tool first writes this packet information to XML

**Table 1**  
Example link score calculation for a graph.

Properties	Weights (%)	Property score
Number of packets	30	80
Average packet delay	0	0
Number of bytes	0	0
Average packet size	0	0
Periodicity	70	100
<b>Weighted average link score</b>		<b>94</b>

first for easier access and manipulation of packet characteristics. The parsed XML file is read and a graph structure with nodes and links is created. The tool uses a weighted-average approach to match the packet traces based on weights given by a user. Five typical link characteristics that a user tries to identify in a sensor network are utilized: number of packets sent, average packet delay, number of bytes sent, average packet size, and periodicity of packets sent. Depending on the user's preferred properties, weights (in percentages of the total score) for each of these link properties are obtained from the user before comparing the traces. These weights are considered for calculating individual scores for each packet trace. The trace file to be compared is referred to as the *problem* and the trace to be compared against is the *solution*.

A score for each of the properties is calculated based on the percentage accuracy of the value of the property with respect to the value in the corresponding link of the solution trace. Once this one-to-one mapping for all the properties is done, a weighted average of the link score is calculated. If the score passes a threshold 'N', the link gets a score of 1, else, it gets a 0. If N percent of the total number of links have a score of 1, trace matching/similarity is reported. Once this process of mapping the two traces is done, the two graphs are displayed along with the link properties to help visually identify the difference in properties, links and nodes. Table 1 shows the calculation of a link score as an example.

After initiating this comparison and calculating network similarity, the two networks are plotted using the Graphviz graph library [39] in Qt [40] and the percent network similarity is reported. The two networks are shown next to each other along with missing nodes/links color coded in red and extra nodes/links in blue. The features and capabilities of the network comparison tool allow one to successfully compare two traces. An example usage of the network comparison tool is shown in Fig. 9.

## 6. Scripting language for TinyOS

TinyOS [7] is an operating system for WSNs that uses nesC [10], a C-based language for developing sensor network applications. The system is widely used, but its main shortcomings for programmers are the complicated structure of the nesC language itself and the multiple files that have to be implemented for a single application. The scripting language in PROVIZ was developed to overcome these difficulties by having a simpler, user-friendly language based on C/C++ concepts with single-line commands for commonly used modules, and developing the entire application in a single file.

### 6.1. Application development with nesC

TinyOS requires the creation of two files for building an application - a configuration file and an implementation file. The configuration file contains wiring of modules (i.e., functions) and interfaces that are used and provided by this application. The implementation file is where the actual behavior is written. TinyOS operations are split-phase operations, meaning, function calls for operations like send packet, start timer, etc. that take too long



```

configuration RadioCountToLedsAppC {}
implementation {
  components MainC, RadioCountToLedsC as App, LedsC;
  components new AMSenderC(AMRADIO.COUNT.MSG);
  components new AMReceiverC(AMRADIO.COUNT.MSG);
  components new TimerMilliC();
  components ActiveMessageC;
  App.Boot -> MainC.Boot;
  App.Receive -> AMReceiverC;
  App.AMSend -> AMSenderC;
  App.AMControl -> ActiveMessageC;
  App.Leds -> LedsC;
  App.MilliTimer -> TimerMilliC;
  App.Packet -> AMSenderC;
}

```

**Listing 3.** Configuration file for the application.

```

module RadioCountToLedsC @safe() {
  uses {
    interface Leds;
    interface Boot;
    interface Receive;
    interface AMSend;
    interface Timer<TMilli> as MilliTimer;
    interface SplitControl as AMControl;
    interface Packet;
  }
}
implementation {
  message_t packet;
  bool locked;
  uint16_t counter = 0;
  event void Boot.booted() {
    call AMControl.start();
  }
  event void AMControl.startDone(error_t err) {
    if (err == SUCCESS) {
      call MilliTimer.startPeriodic(250);
    }
    else {
      call AMControl.start();
    }
  }
  event void AMControl.stopDone(error_t err) {
    // do nothing
  }
}

```

**Listing 4.** Part of the implementation file for the application.

to execute are not blocking calls. Instead, the calls are immediately returned and when the operation finishes execution, callbacks (events) are generated. For a novice developer, these concepts are hard to grasp.

Listings 3 and 4 are snapshots of the configuration and implementation files of a sample *RadioCountToLeds* application in TinyOS. The application starts a counter that updates itself every  $n$  milliseconds and sends out a packet containing the counter value. A node that receives this packet will read the counter value from the payload of the packet and displays the LSB on the LEDs. The snapshot of the implementation file is only about half the entire program. As is apparent, this is information overload for a novice user even if the user has basic programming knowledge.

## 6.2. A simple scripting language

In the previous sub-section, a sample TinyOS application was discussed and the difficulty in programming with nesC was demonstrated. Considering this, the design of the scripting language in PROVIZ was influenced by the following three factors: 1)

eliminating the writing of a configuration file and the wiring involved, 2) abstracting interfaces and making it simpler to create and reuse code, and 3) developing a language that can be scaled to include more abstracted functions and complex applications.

With these factors in mind, a C-based scripting language was chosen as C's basic principles are simple, intuitive, and easy to learn. While there are some scripting language specific function calls and include headers, the rest of the program written with this language follows only C. Therefore, users can port any C program to this script by simply modifying only those functions that are sensor-specific. Listing 5 shows the *RadioCountToLeds* application written in the new scripting language.

In the TinyOS application, a boot-up sequence of actions specified by the user is the starting point of the program. This boot-up sequence is called when the sensor boots, but this function call is not visible to users. To avoid this confusion, a `main()` function is added prior to the script that has a single function call to `INITIAL()`. The users will, then, have to define actions in `INITIAL()` as the boot-up sequence. The scripting language tries to expose the split-phase operation of function calls in TinyOS as much as possible while still trying to remain tractable. Apart from calling the timer and radio interfaces to start the timer or send packets, the user will have to define what has to be done in the callbacks (events) that are generated. This TinyOS concept was retained in the scripting language to offer more flexibility to the users in developing their application.

Listings 6 and 7 describe the TinyOS property of providing and using interfaces as a C++ abstraction of static classes. Creating an interface (a sequence of TinyOS commands and events) involves an interface name and the name of the module that implements the interface. This is the only way TinyOS allows the user to develop code in one file and use it across many files. This complexity can be abstracted to the concept of a static C++ class (interface) and the include file in which the static class functions are defined (module). Functions to be exported are defined as static class functions and the program that uses these functions, calls them like regular static class functions (without a class object) and includes the name of the program that implemented them. By abstracting this TinyOS concept to C++ classes, novice users can save time writing nesC code and understand nesC concepts better.

## 6.3. OMNeT simulation

Our script generates nesC code that can be either programmed onto a WSN or simulated in the OMNeT simulator [1], which is a C++ based, modular network simulator. The advantages in using OMNeT is that it allows heterogeneous network simulations where each node in the network can run their own application unlike TOSSIM [6] that can simulate only homogeneous networks. TinyOS applications can be simulated in OMNeT using NesCT [41]. NesCT is a language translator that takes nesC programs as input and generates OMNeT code that can be simulated in the OMNeT environment. However, NesCT only takes version 1 of TinyOS as input while the latest release of TinyOS is version 2. The scripting language in PROVIZ generates TinyOS-2.x code as well. To convert TinyOS-2.x code to TinyOS-1.x code, a custom parser was developed in Python. There are a few differences in TinyOS-2.x and TinyOS-1.x code and the parser simply parses these variations in code and generates TinyOS-1.x code that can be fed into the NesCT language translator. After simulating in OMNeT, the packets from the simulator output are sent to PROVIZ for visualization.

## 7. Visual programming tool

The visual programming tool can be used to introduce youngsters to programming concepts with the help of sensors. The tool

```

//Packet payloads are defined in structs for easy access of payload
//information
struct test_counter_msg {
    int count;
};
int counter = 0;
void INITIAL() {
    startPeriodicTimer_1(1000);
}
//Callback functions are defined with event return type
event timerTimeout_1() {
    counter = counter + 1;
    createPacket(packet1, test_counter_msg, counter);
    sendPacket(packet1, 20);
}
event receivePacket(test_counter_msg* msg) {
    if(msg->count & 0x1)
        led0On();
    else
        led0Off();
    if(msg->count & 0x2)
        led1On();
    else
        led1Off();
    if(msg->count & 0x4)
        led2On();
    else
        led2Off();
}
void main() {
    INITIAL();
}

```

**Listing 5.** RadioCountToLeds in the new scripting language.

```

void INITIAL() {
    //implementation
}
void Routing::getStateTable() {
    //implementation
}
void Routing::getHopCount() {
    //implementation
}
void main() {
    INITIAL();
}

```

**Listing 6.** Sample script that implements an interface in the form of static class functions.

```

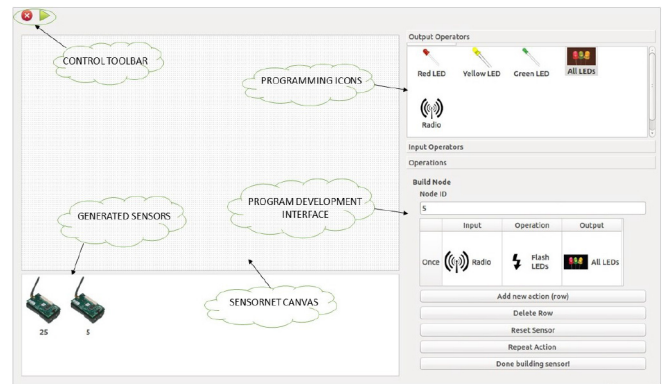
#include "Route"

void INITIAL() {
    //implementation
    Routing::getStateTable();
}
void main() {
    INITIAL();
}

```

**Listing 7.** Sample script that uses routing functions implemented in the previous script.

aids in creating a heterogeneous WSN with multiple sensors running custom programs. It has simple, visually descriptive drag-and-drop icons to create a sensor network program and connect sensors via radio. With this tool, reading temperature or light sensor, playing with the LEDs, sending messages over the air and setting up timed actions in sensors become much simpler. This tool gener-















**Fig. 7.** User interface of the visual programming tool.

ates code in the scripting language described in the previous section. The tool can easily be a stepping stone for young minds in learning a new language such as C and sensor network programming. They can visually see how their drag-and-drop icons translate to the scripting language and learn there onwards.

### 7.1. Design

The visual programming tool was developed in Qt [34] on top of the PROVIZ framework. Qt is an open source, multi-platform, C++ based framework for developing graphical user interfaces (GUIs). The user interface (UI) of the tool has five components as shown in Fig. 7. They are: 1) *Control toolbar* - for generating code and resetting the UI, 2) *Programming icons* - drag-and-drop icons to create sensor programs, 3) *Program development interface* - for creating action sequence, setting timer values, node ID, etc.,

**Table 2**  
Operator list.

Icon	Type of function	Name	Function
	Input operators	Light sensor	Reads the light sensor value
	Input operators	Temperature sensor	Reads the temperature sensor value
	Input operators	Radio	Reads payload value sent via radio
	Main operators	Turn off	Turns off the LED
	Main operators	Turn on	Turns on the LED
	Main operators	Send to	Sends input value over to radio
	Main operators	Flash LEDs	Flashes LSB to output LEDs
	Output operators	All LEDs	Operation routed to all LEDs
	Output operators	Green LED	Operation routed to Green LED
	Output operators	Red LED	Operation routed to Red LED
	Output operators	Yellow LED	Operation routed to yellow LED
	Output operators	Radio	Input value sent over radio

4) *Generated Sensors* - list of sensors developed in the program development interface and, 5) *Sensor network canvas* - canvas for creating a sensor network environment with sensors generated by the program development interface. These components are discussed below.

#### 7.1.1. Control toolbar

The control toolbar has options to reset the whole interface and start fresh and to generate code once the WSN is created in the *Sensor Network Canvas*.

#### 7.1.2. Programming icons

This interface has three sets of draggable icons for programming - input operators, output operators and operations. These icons have to be dragged and dropped onto the table to create an action sequence. Table 2 describes the function of each operator icon.

#### 7.1.3. Program development interface

This interface has a dynamically sized table where each row represents an action sequence in a sensor. For example, a sequence can be reading a sensor value and flashing the value using LEDs or another action sequence such as turning on the red LED. An action sequence can be repeated with a timer by selecting the row to be repeated and specifying the timer value in milliseconds. Action rows can be deleted or added dynamically. Once the sensor is built, a node ID is given to the sensor and the sensor is generated. That is, the program for this sensor is generated in the scripting language in the background and visually, the sensor is added to the *Generated Sensors* list.

#### 7.1.4. Generated sensors

Once a sensor is built in the *Program Development Interface*, the sensor is added as an icon to this list along with node ID. As the user continues building programs for other sensors, sensors are accumulated in this list. These sensors can be dragged and dropped onto the *Sensor Network Canvas*.

#### 7.1.5. Sensor network canvas

Sensors can be dragged and dropped onto this canvas to create a heterogeneous WSN. Nodes can be removed from the canvas by right-clicking a node and selecting the delete option. Nodes can be connected to each other via radio by right-clicking and connecting to a list of nodes in the canvas. By selecting this option, if the source node has an output *Radio* operator, packets are sent to the destination node ID specified.

## 8. Usage scenarios

In this section, we first demonstrate a scenario in which, PROVIZ is used to visualize and debug a security attack. Then, we demonstrate the usage of the packet trace comparison tool. Finally, we show the visual programming capability of PROVIZ.

### 8.1. Visualization

We use PROVIZ to visualize a WSN and show how the tool enables visual debugging to identify a security breach such as a black-hole attack [42]. To demonstrate the black-hole attack, first a compromised node advertises a route message to other nodes, claiming to have a shortest route to the cluster head. Then, the compromised node starts receiving the packets from other nodes, whose packets it eventually drops.

Fig. 8 shows the visualization of a WSN with three environment monitoring sensor nodes, with Node IDs 10, 20 and 30 and a cluster head, with Node ID 80. The sensor nodes in the WSN gather the environment information (e.g., temperature, ambient light) periodically and sends it to the cluster head. The sensor nodes use the Dynamic Source Routing (DSR) [43] protocol for determining the route to the cluster head and uses source-routing to route the packets to the destination. In DSR, to determine a route to a particular node ID, the sender generates a Route Request (RREQ) message for the destination ID and then broadcasts it. The intermediate nodes, on receiving this message, will append their node ID and re-broadcasts the RREQ message, until the destination node receives the RREQ and replies with a Route Reply (RREP) message. The RREP message has the route information that the RREQ message took and it is source-routed to the sender. The sender, on receiving the first RREP message, gets the route information and starts sending the packets.

Node 10 in the WSN sends the data to the cluster head, in a multi-hop route through nodes 20 and 30 as shown in Fig. 8. In the same network, we assume that node 40 is compromised by an attacker to demonstrate the black-hole [42] attack scenario. Although, node 40 is not a single hop neighbor of the cluster head, it sends a route advertisement claiming that it is one hop away from the cluster head. After receiving the route advertisement, node 10 starts sending the data packets through node 40 and the compromised node starts dropping the packets, thus initiating a black-hole attack. When an attack of this type is launched, it may not be triv-

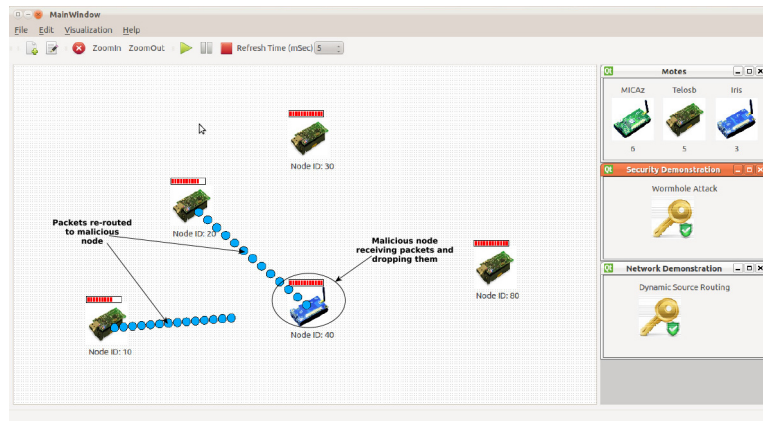


Fig. 8. PROVIZ demonstrating the visualization of a black-hole attack.

ial for a user to understand this attack and to identify the malicious node. In the mean time, the compromised node might have dropped critical time-sensitive data. The user needs to analyze the packet logs continuously to detect an anomaly in the WSN, which is a time consuming process. Also, if the black-hole attack is initiated intermittently, then it will be hard to detect the attack by packet log analysis. However, with the help of PROVIZ visualization tool, the researcher can monitor or replay the packet transfers from the point when the cluster head stops receiving packets and can visually debug and identify that node 40 is compromised. Fig. 8 shows that node 40 receives packets from nodes 10 and 20 and drops the packets, which depicts a black-hole attack. On identifying the black-hole attack, the researcher can use the PROVIZ programming functionality to modify the WSN application and program the compromised node over-the-air. Thus, the visualization tool can be used as a visual monitoring and debugging tool to detect a network anomaly and its programming functionality can be used to provide an instant fix for the attack by reprogramming the sensor node.

### 8.2. Network comparison

The packet trace comparison tool was tested on real sensors (MicaZ) and WSNs to ensure robustness and efficiency. We set up five MicaZ sensors in two different network configurations with 5 nodes each. Packet trace (.psd) files of the two networks were obtained from the TI SmartRF Packet Sniffer. These traces were compared by the packet comparison tool and the graph comparison UI was generated using the Graphviz.

In this demo, we first receive weights from the user for each of the five link characteristics. With the weights, the two networks are compared as discussed earlier. Fig. 9 shows the two traces' topology in comparison to each other. The extra nodes/edges are drawn in blue and the missing nodes/edges in red. On right-clicking each link, the properties of that link and the link in the other graph are displayed respectively. When a missing or extra edge is right-clicked, only link properties are reported.

Finally after the comparison is computed, the tool reports via a dialog pop-up the two traces' percent matching. In our case, the match percentage was 93. We note that our purpose is not to provide a higher percentage in matching. Indeed, a higher percentage means that the two networks are very similar, while a lower percentage means that they differ in many aspects based on the weight of every characteristic provided by the user. This is especially useful for educators or developers when they are trying to debug or understand the results. For instance, an educator would be able to understand how much a student's submission matches a specific expected result.

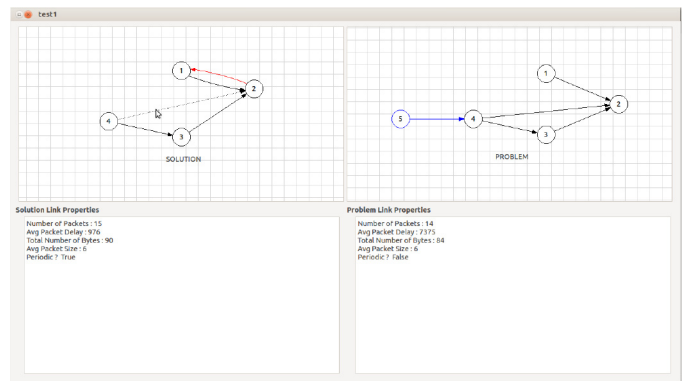


Fig. 9. Comparison of graphs when right-clicking a common link.

### 8.3. Visual programming

In this sub-section, we demonstrate a usage scenario for the visual programming tool provided in PROVIZ.

Fig. 10 shows a sequence of actions for developing a two-node heterogeneous WSN. The user drags and drops operators/operations to create a row of sensor actions, gives the sensor a node number and places the sensor on the canvas. Lines drawn between sensors show the direction of communication between sensors. In this scenario, Node 4 senses light and sends the value over the radio. Node 5 receives packets from the radio and flashes the payload to the LEDs. Node 4 is connected to Node 5 in the sensor network canvas and packets generated in Node 4 are sent to Node 5. The automatic code generated in the scripting language in PROVIZ for the two sensors created with the visual programming tool are shown in Listings 8 and 9. With this step-by-step approach to sensor programming, novices can first experiment with the visual programming tool and study the code generated by the scripting language to get a better grasp of programming in nesC.

## 9. Benefits of PROVIZ

In this section, we articulate various ways PROVIZ would be instrumental to engineers, scientists, and educators in accomplishing tasks in their domains.

- *Working with heterogeneous wireless sensors:* Today, most wireless sensor deployments are heterogeneous with different types of sensor nodes (e.g., MicaZ, IRIS, Telosb) sending packets with different packet payload formats. In these settings, the sensor nodes have varying architectures with different limited hard-



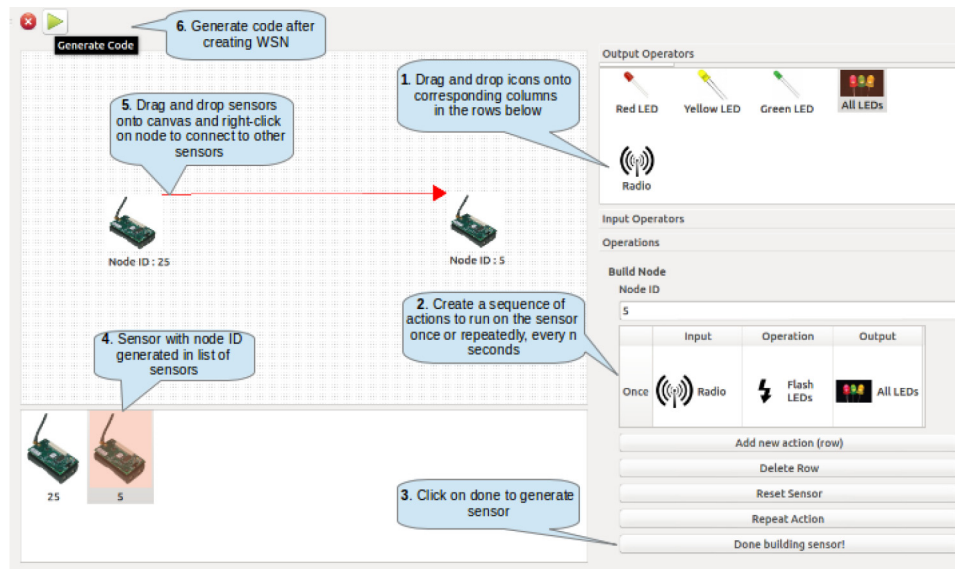


Fig. 10. Creation of a WSN and generating its code.

```

struct sensor_reading
{
    unsigned int light;
};
void INITIAL()
{
    startPeriodicTimer_1(1000);
}
event timerTimeout_1()
{
    lightInfo = senseLight();
    createPacket(packet1, sensor_reading, lightInfo);
    sendPacket(packet1, 5);
}
void main()
{
    INITIAL();
}

```

Listing 8. Program for node 25.

```

struct sensor_reading
{
    unsigned int light;
};
void INITIAL()
{
}
event receivePacket(sensor_reading * msg)
{
    if(msg->light & 0x1)
        led0On();
    else
        led0Off();
    if(msg->light & 0x2)
        led1On();
    else
        led1Off();
    if(msg->light & 0x4)
        led2On();
    else
        led2Off();
}
void main()
{
    INITIAL();
}

```

Listing 9. Program for node 5.

ware components (e.g., memory size, battery sizes) and a heterogeneous wireless sensor network deployment allows for efficient utilization of the limited resources onboard sensors and to maximize the lifetime of the sensor network with sensors with different composition of resources. PROVIZ is capable of visualizing and programming such a heterogeneous wireless sensor network and it is able to understand different packet formats generated by different sensors.

- **Support for distributed deployment of wireless sensors:** For visualizing a large wireless sensor deployment, PROVIZ is able to support a distributed approach that uses multiple sniffers. In the distributed setup, the PROVIZ visualization tool can run in

a host machine and multiple sniffers are placed in a distributed fashion so that a packet transmitted by a sensor node can be sniffed by at least one of the sniffers. The PROVIZ framework can also utilize the NTP protocol to synchronize the packets received from multiple sniffers.

- **Built-in extensible visual demos:** PROVIZ includes built-in extensible visual demo deployment scenarios, which enables one to easily visualize pre-defined sensor scenarios. Researchers can use this feature to create a demo scenario to visualize a critical/complex wireless sensor deployment and share it with other PROVIZ users. In addition to default scenarios, the demo visualization scenarios can be extended or customized by users with an XML file, in which users will be able to specify the sensor type to be used, number of sensors, and other details. These easy-to-run demonstrations would be also instrumental in pedagogical contexts. For instance, undergraduate, graduate, and even students in K-12 can have an opportunity to learn to use different sensors and appreciate their use in science and engineering fields.
- **Connection to external simulators:** PROVIZ framework includes a capability to connect to external network simulators (e.g., OM-NeT). Specifically, it will be able to visualize data trace generated by a wireless sensor network deployment in an external simulation environment. Further, it is able to generate the code that can run in the simulator. In this way, simulators popularly used by computer engineers and scientists can also be accessible to researchers and engineers of other fields (e.g., civil engineers, geophysicists) and PROVIZ is able to alleviate the complexity associated with programming network simulators.
- **Visual debugging of wireless sensor deployments and data:** PROVIZ can visualize the wireless sensors and their data by continuously monitoring the network activity and can be used by researchers for visual debugging purposes. Further, it can also be utilized for detecting possible malicious activity by visual inspection and can aid in providing a software fix (re-programming the sensor).
- **Easy programming of wireless sensors:** PROVIZ framework includes two simplified development environments (languages) to program wireless sensors. The first one is a visual programming language where scientists and researchers can use visual blocks and certain shapes as programming constructs and the other one is a scripting language, which is a domain specific

simplified language with simple commands to ease programming tasks. These simplified languages can allow researchers in other disciplines to easily program the sensors and focus more on the tasks in their domains rather than on the overhead of learning how to program sensors and the details of the underlying sensor software architecture (i.e., TinyOS, nesC). Hence, PROVIZ can significantly expedite the development of programs for sensors and, therefore, give the scientists and engineers from other disciplines more time to make stronger research contributions.

- *Over-The-Air Remote (Re)programming of wireless sensors:* Sensors are deployed in various locations such as underwater, volcano regions, and underground, making them hard to recover or replace. Due to this diverse deployment of wireless sensors, reprogramming the deployed sensor nodes through wireless links becomes a necessary and desirable task. For instance, if software running on the sensor requires an update as a result of a security patch or additional functionality, it would be necessary to replace the existing code on the sensor with the new updated code. This situation is daunting and error-prone for a civil engineer or geophysicist who have to connect hundreds of sensors to a desktop computer with a cable to update their code. To address these needs, the PROVIZ framework includes a capability to program/reprogram sensors remotely over the air. This is an instrumental feature for researchers who would not be able to recover their wireless sensors easily.

## 10. Conclusion and Future work

In this work, we introduced PROVIZ, which is an integrated visualization and programming framework developed for WSNs. PROVIZ is an open-source framework, which is designed to be modular, scalable, and platform independent. PROVIZ is capable of visualizing a WSN and the packet transfers occurring between the sensor nodes in real-time. PROVIZ visualization tool is generic and extensible such that it can take packet data input from various sources like live sensor-based sniffers, commercial sniffers (e.g., TI SmartRF packet sniffer [36]) and the OMNeT simulator [1]. PROVIZ can take multiple user-defined packet formats as input and translate the raw packet data from a heterogeneous network into a user readable content. Users can define the packet formats graphically using the PROVIZ Packet Format Specifier. Also, PROVIZ includes built-in extensible visual demo deployment scenarios that can even be shared among the users in the form of XML files. The network comparison tool can enable users to compare two network deployments for differences in link characteristics such as average packet delay, number of bytes in payload, etc. This tool can help make programming concepts easier for K-12 students. It can aid teachers unfamiliar with programming to grade WSN programming assignments. The scripting language inside PROVIZ offers application development for TinyOS in a single file, eliminating complicated concepts such as the wiring and interface in TinyOS. It has a scalable design where more add-on modules for projects like data collection, monitoring, routing, etc. can be programmed in TinyOS and abstracted to functions. The tool also allows programmers to reuse WSN functions developed in some other file. The visual programming tool has simple drag-and-drop icons to create WSN programs and connect sensors via the radio interface. Again, this tool can help teach students basic programming concepts. Finally, PROVIZ is capable of remotely programming the sensor nodes by disseminating the code wirelessly. Compared to other similar useful sensor programming works in literature, PROVIZ's main advantage is that it is a complete integrated system where visualization, programming, and simulation can be done from the same software architecture.

Our future work will extend PROVIZ to get sensory data like temperature, battery level, etc. and enhance the functionalities of the network comparison tool. We will also improve PROVIZ to include a capability to define notifications to the user whenever a predefined network condition is reached. These features will be helpful when a researcher, developer, or user need to continuously monitor a WSN to get notifications. For instance, the user can be notified whenever the battery level of a sensor node goes down beyond certain limit or when the temperature readings goes beyond certain threshold.

## Acknowledgment

This work was supported by NSF-ACI-1339781. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the funding agencies. The authors would also like to thank Mike Alan of Florida International University for his contributions to the project.

## References

- [1] A. Varga, Omnet simulator, 2014, (<http://www.omnetpp.org/>).
- [2] R.K. Chandrasekar, V. Subramanian, S. Uluagac, R. Beyah, SIMAGE: secure and Link-Quality cognizant image distribution for wireless sensor networks, in: IEEE Global Telecommunications Conference, 2012, pp. 616–621.
- [3] L. Ma, L. Wang, L. Shu, J. Zhao, S. Li, Z. Yuan, N. Ding, Netviewer: a universal visualization tool for wireless sensor networks, in: Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM), 2010, pp. 1–5.
- [4] R. Jurdak, A.G. Ruzzelli, A. Barbirato, S. Boivineau, Octopus: monitoring, visualization, and control of sensor networks, Wiley Wireless Commun. Mobile Comput. 11 (8) (2011) 1073–1091.
- [5] L. Shu, C. Wu, Y. Zhang, J. Chen, L. Wang, M. Hauswirth, Nettopo: beyond simulator and visualizer for wireless sensor networks, in: Proceedings of the Second IEEE International Conference on Future Generation Communication and Networking, 1, 2008, pp. 17–20.
- [6] P. Levis, N. Lee, M. Welsh, D. Culler, TOSSIM: accurate and scalable simulation of entire tinyOS applications, in: Proceedings of the 1st ACM international conference on Embedded networked sensor systems, 2003, pp. 126–137.
- [7] TinyOS, TinyOS documentation, 2014. URL <http://docs.tinyos.net/>.
- [8] A. Burns, B.R. Greene, M.J. McGrath, T.J. OShea, B. Kuris, S.M. Ayer, F. Stroiescu, V. Cionca, SHIMMER – a wireless sensor platform for noninvasive biomedical research, IEEE Sens. J. 10 (9) (2010) 1527–1534.
- [9] N. Kurata, B. Spencer, M. Ruiz-Sandoval, Risk monitoring of buildings with wireless sensor networks, Struct. Cont. Health Monit. 12 (3–4) (2005) 315–327.
- [10] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, D. Culler, The nesc language: a holistic approach to networked embedded systems, in: Proceedings of the ACM Conference on Programming Language Design and Implementation, 2003, pp. 1–11.
- [11] M. Dyer, J. Beutel, T. Kalt, P. Oehen, L. Thiele, K. Martin, P. Blum, Deployment support network, in: Springer Wireless Sensor Networks, 2007, pp. 195–211.
- [12] B. Greenstein, E. Kohler, D. Estrin, A sensor network application construction kit (SNACK), in: Proceedings of the 2nd ACM International Conference on Embedded Networked Sensor Systems, 2004, pp. 69–80.
- [13] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, Y. Kafai, Scratch: programming for all, Commun. ACM 52 (11) (2009) 60–67, doi:10.1145/1592761.1592779.
- [14] Alice, Alice, 2014. URL [www.alice.org/](http://www.alice.org/).
- [15] M. Turon, J. Suh, MOTE-VIEW: a sensor network monitoring and management tool, in: Proceedings of the 2nd IEEE workshop on Embedded Networked Sensors, 2005, pp. 11–17.
- [16] MEMSIC, memsic, 2014. URL [www.memsic.com/](http://www.memsic.com/).
- [17] B.-R. Chen, G. Peterson, G. Mainland, M. Welsh, Livenet: using passive monitoring to reconstruct sensor network dynamics, in: Proceedings of Springer Distributed Computing in Sensor Systems, 2008, pp. 79–98.
- [18] M. Ringwald, K. RÄjmer, A. Vitaletti, SNIF: Sensor Network Inspection Framework, Technical Report, ETH Zurich, Institute for Pervasive Computing, 2006, 535.
- [19] N. Ramanathan, E. Kohler, L. Girod, D. Estrin, Sympathy: a debugging system for sensor networks [wireless networks], in: Proceedings of the 9th Annual IEEE International Conference on Local Computer Networks, 2004, pp. 554–555.
- [20] F. SantAnna, N.d. L. R. Rodriguez, R. Ierusalimsky, Ceu: embedded, safe, and reactive programming, PUC-Rio, Tech. Rep 12 (2012) 12.
- [21] D. Chu, L. Popa, A. Tavakoli, J.M. Hellerstein, P. Levis, S. Shenker, I. Stoica, The design and implementation of a declarative sensor network system, in: Proceedings of the 5th ACM International Conference on Embedded Networked Sensor Systems, 2007, pp. 175–188.

- [22] J. McCarthy, Datalog: deductive database programming, 2014, ([docs.racket-lang.org/datalog/](https://docs.racket-lang.org/datalog/)).
- [23] E. Cheong, E.A. Lee, Y. Zhao, Vptos: a graphical development and simulation environment for tinyos-based wireless sensor networks, in: Proceedings of the 3rd ACM International Conference on Embedded Networked Sensor Systems, 2005, 302–302.
- [24] W.P. McCartney, N. Sridhar, Tosdev: a rapid development environment for tinyos, in: Proceedings of the 4th ACM International Conference on Embedded Networked Sensor Systems, 2006, pp. 387–388.
- [25] Eclipse, Eclipse, 2014, (<https://www.eclipse.org/>).
- [26] J.B. Lim, B. Jang, S. Yoon, M.L. Sichitiu, A.G. Dean, RaPTEX: rapid prototyping tool for embedded communication systems, *ACM Trans. Sens. Netw.* 7 (1) (2010).
- [27] B.L. Titzer, D.K. Lee, J. Palsberg, Avrora: scalable sensor network simulation with precise timing, in: Proceedings of the 4th IEEE International Symposium on Information Processing in Sensor Networks, 2005, p. 67.
- [28] F. Vieira, B.A. Vitorino, M.A. Vieira, D. Silva, A. Fernandes, A. Loureiro, Wisdom: a visual development framework for multi-platform wireless sensor networks, in: Proceedings of the 10th IEEE Conference on Emerging Technologies and Factory Automation, 2, 2005.
- [29] A. Elsts, J. Judvaitis, L. Selavo, SEAL: a domain-specific language for novice wireless sensor network programmers, in: Proceedings of the 39th IEEE EURO-MICRO Conference on Software Engineering and Advanced Applications (SEAA), 2013, pp. 220–227.
- [30] N. Fraser, Blockly: a visual programming editor, 2014. URL <https://code.google.com/p/blockly/>.
- [31] M.T. Hansen, B. Kusy, Tinyinventor: a holistic approach to sensor network application development, *Extending the Internet to Low power and Lossy Networks*, (2011).
- [32] R.V. Roque, OpenBlocks: an extendable framework for graphical block programming systems, Ph.D. thesis, Massachusetts Institute of Technology, 2007.
- [33] R.K. Chandrasekar, S. Uluagac, R. Beyah, Proviz: an integrated visualization and programming framework for wsns, in: Proceedings of the 38th IEEE Conference on Local Computer Networks Workshops (LCN Workshops), 2013, pp. 146–149.
- [34] Qt, Qt: A GUI development framework, 2014, (<http://qt-project.org/>).
- [35] TI, Cc2420 rf transceiver, 2014. URL <http://www.ti.com/lit/ds/symlink/cc2420.pdf>.
- [36] TI, TI smartrf packet sniffer, 2014, (<http://www.ti.com/tool/packet-sniffer>).
- [37] IEEE-SA, in: Approved draft revision for ieee standard for information technology-telecommunications and information exchange between systems-local and metropolitan area networks-specific requirements-part 15.4b: Wireless medium access control (mac) and physical layer (phy) specifications for low rate wireless personal area networks (wpans) (amendment of ieee std 802.15.4-2003), IEEE Std P802.15.4/D6, 2006.
- [38] D.L. Mills, Network time protocol project, 2014, (<http://www.ntp.org/>).
- [39] A.L. Research, Graphviz – graph visualization software, 2014, (<http://www.graphviz.org/>).
- [40] N. Bergont, Interactive qt graphviz display, 2014, (<http://code.google.com/p/qgv/>).
- [41] O.S. Kaya, Nesct: a language translator, 2014, (<http://nesct.sourceforge.net/>).
- [42] Y.-C. Hu, A. Perrig, D.B. Johnson, Wormhole attacks in wireless networks, *IEEE J. Sel. Areas Commun.* 24 (2) (2006) 370–380.
- [43] D.B. Johnson, D.A. Maltz, Dynamic source routing in ad hoc wireless networks, in: T. Imielinski, H. Korth (Eds.), *Mobile Computing*, The Kluwer International Series in Engineering and Computer Science, 353, Springer US, 1996, pp. 153–181.



**Shruthi Ravichandran** received her B.E. degree in Electronics and Communications Engineering from Anna University, India in 2012 and an M.S. degree in Electrical and Computer Engineering from Georgia Institute of Technology, Atlanta, USA in 2014. Since then, she has been with National Instruments as a Software Engineer and works on drivers for Timing and Synchronization modules.



**Ramalingam** was born on July 8, 1986 in Madurai, India. He received his B.E. in Electrical and Electronics Engineering (EEE) from PSG College of Technology, affiliated to Anna University, Tamilnadu, India, in 2008. He was awarded with the most prestigious Koorathalwar award, for his excellent work in academics, sports, and service to the society. After his graduation, he worked with TOSHIBA, India, for three years as a Software Engineer in the Multi-Function Printer (MFP) network solutions group. He received his M.Sc. in Electrical and Computer Engineering (ECE) from Georgia Institute of Technology, Atlanta, GA, in 2013. During his Masters, he served as a Graduate Research Assistant (GRA) in the Communications Assurance and Performance (CAP) group at Georgia Tech and has several publications in the area of security and Wireless Sensor Networks (WSNs). In Summer 2012, he did his internship with Qualcomm Innovation Center (QUIC), Boulder, CO. In QUIC, he worked with the Android power management team and after his graduation; he continued working as a full-time Software Engineer at QUIC from July 2013. Also, he is a trustee of the non-profit organization named Computer Kindness Foundation (CKF), which work towards helping students to get basic education.



**Dr. A. Selcuk** Uluagac is currently an Assistant Professor in the Department of Electrical and Computer Engineering (ECE) at Florida International University (FIU). Before joining FIU, he was a Senior Research Engineer in the School of Electrical and Computer Engineering (ECE) at Georgia Institute of Technology. Prior to Georgia Tech, he was a Senior Research Engineer at Symantec. He earned his Ph.D. with a concentration in information security and networking from the School of ECE, Georgia Tech in 2010. He also received an M.Sc. in Information Security from the School of Computer Science, Georgia Tech and an M.Sc. in ECE from Carnegie Mellon University in 2009 and 2002, respectively. The focus of his research is on cyber security topics with an emphasis on its practical and applied aspects. He is interested in and currently working on problems pertinent to the security of Cyber-Physical Systems and Internet of Things. In 2015, he received a Faculty Early Career Development (CAREER) Award from the US National Science Foundation (NSF), which is NSF's most prestigious award in support of junior faculty who exemplify the role of teacher-scholars through outstanding research, excellent education and the integration of education and research within the context of the mission of their organizations. In 2015, he was awarded the US Air Force Office of Sponsored Research (AFOSR)'s 2015 Summer Faculty Fellowship. In 2007, he received the "Outstanding ECE Graduate Teaching Assistant Award" from the School of ECE, Georgia Tech. He is also an active member of IEEE (senior grade), ACM, and ASEE and a regular contributor to national panels and leading journals and conferences in the field. Currently, he is the area editor of Elsevier Journal of Network and Computer Applications and serves on the editorial board of the IEEE Communication Surveys and Tutorials. More information can be obtained from: <http://web.eng.fiu.edu/selcuk>.



**Raheem Beyah** is an Associate Professor in the School of Electrical and Computer Engineering at Georgia Tech where he leads the Communications Assurance and Performance Group (CAP) and is a member of the Institute for Information Security & Privacy (IISP) and the Communications Systems Center (CSC). Prior to returning to Georgia Tech, Dr. Beyah was an Assistant Professor in the Department of Computer Science at Georgia State University, a research faculty member with the Georgia Tech CSC, and a consultant in Andersen Consulting's (now Accenture) Network Solutions Group. He received his Bachelor of Science in Electrical Engineering from North Carolina A&T State University in 1998. He received his Masters and Ph.D. in Electrical and Computer Engineering from Georgia Tech in 1999 and 2003, respectively. Dr. Beyah has served as a Guest Editor for MONET and is currently an Associate Editor of the (Wiley) Wireless Communications and Mobile Computing Journal. His research interests include network security, wireless networks, network traffic characterization and performance, and critical infrastructure security. He received the National Science Foundation CAREER award in 2009 and was selected for DARPA's Computer Science Study Panel in 2010. He is a member of AAAS, ASEE, a lifetime member of NSBE, and a senior member of ACM and IEEE.