PROVIZ: An Integrated *Vis*ualization and *Pro*gramming Framework for WSNs

Ramalingam K. Chandrasekar, A. Selcuk Uluagac and Raheem Beyah
Communications Assurance and Performance Group, School of Electrical and Computer Engineering
Georgia Institute of Technology, Atlanta, Georgia 30332, USA
ramalingam.chandrasekar@gatech.edu, {selcuk, rbeyah}@ece.gatech.edu

***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. Independent of the functionalities provided by the WSN applications, many of the developers use visualization, simulation, and programming tools. However, these tools are designed as separate stand-alone applications, which force developers to use multiple tools. This situation often poses confusion and hampers an efficient development experience. To avoid the complexity of using multiple tools, we have designed a new extensible, multi-platform, scalable, and open-source framework called PROVIZ, which is an integrated visualization and programming framework. In this paper, we explain the various features of PROVIZ's visualization and programming framework and discuss how PROVIZ can be used as a visual debugging tool to aid in providing a software fix.**

Index Terms - Wireless Sensor Networks, PROVIZ, Visualization and Monitoring tool, Programming tool, Simulation tool

I. Introduction

Wireless Sensor Networks (WSNs) are used in various applications like remote health monitoring, volcanic activity monitoring, and critical infrastructure monitoring. Hence, the WSN researchers' community comprises of advanced WSN application developers, who develop scalable and reliable WSN algorithms and applications, and developers who use WSNs for simple environment data gathering. In order to develop, test, visualize, and monitor a WSN application, a developer may need a programming tool, a simulator [1], [2], and a visualization tool [3], [4]. However, such tools are currently available as separate stand-alone tools. Therefore, to aid in the process of WSN application development, we propose a new framework called PROVIZ [5], which provides visualization, monitoring, and programming functionalities into a common platform. Furthermore, as sensors are resource-constrained devices running with a limited source of power, critical applications cannot afford the failure of any sensor or sensor application. Thus, a framework like PROVIZ which can visualize the WSN by continuously monitoring the network activity, can be used by researchers for visual debugging purposes.

The PROVIZ framework is an open-source, platform-independent, extensible, and scalable framework developed for heterogeneous WSNs. It is an integrated framework that can visualize various sensor types (e.g., MicaZ, Iris, TelosB) and their traffic in realtime. It provides a set of built-in tools for developing sensor applications and for programming sensor nodes over-the-air [6]. PROVIZ can also visualize a WSN simulator (e.g., OMNeT [7]) generated trace data. Moreover, PROVIZ includes built-in extensible visual demo deployment capabilities, which allow users to quickly craft network scenarios. It is possible to export these demo scenarios as XML files so that users can easily modify and even share them among themselves.

PROVIZ realizes the visualization of the packet transfers by parsing the packet data captured either from a packet sniffer (a live sensor-based sniffer or from a binary packet trace file (PSD format) generated by the TI SmartRF packet sniffer [8]) or from the OMNeT simulator [7] running a WSN application. For visualization, PROVIZ can take different packet payload formats as input while providing support for defining the packet payload format through a graphical user interface. Moreover, PROVIZ can work with multiple sniffers simultaneously in a distributed setup to visualize a large WSN deployment.

PROVIZ includes an optional thin-client, which exists in the sensor node and sends the sensor state information periodically. With the help of these periodic reports, researchers can easily identify a network anomaly and can take necessary corrective measures for avoiding failures in the WSN. Finally, PROVIZ includes a programming editor for generating TinyOS [9] application code. This editor can be used to edit both NesC [10] code and MCL [11] scripts. After the application code image is generated, PROVIZ has the capability to disseminate the code image wirelessly to the sensor nodes [6].

The rest of the paper is organized as follows. Related work is discussed in Section II. An outline of PROVIZ's design is given in Section III. Section IV gives an overview of the PROVIZ visualization framework and Section V concludes the paper.

II. Related Work

NetViewer [3] is a visualization tool used for visualizing and monitoring a realtime WSN. NetViewer is designed in such a way that it will work for any user-defined packet format, which is facilitated by the *Packet Format Setter* module in NetViewer. NetViewer has a *Information Translator* module to translate the packet data to useful information based on the user-defined packet format. NetViewer, then, visualizes the packet transmissions and displays the translated packet data either in a tabular format or in the form of a graph. However, NetViewer cannot do an off-line WSN visualization (by parsing the commercial packet sniffer's packet trace file (PSD format)) and cannot visualize a data trace generated by an external WSN simulator. Also, NetViewer is a standalone visualization tool, which does not have the capability to wirelessly program sensor nodes.

978-1-4799-0540-9/13/$31.00 ©2013 IEEE

Octopus [4] is an integrated monitoring, visualization, and control tool developed for WSNs. Octopus is a protocol independent tool which gets information about the sensor nodes and the network topology, which can be used to visualize and monitor realtime sensor nodes in a WSN. Octopus also provides *Control Operators*, which can reconfigure the network behavior by sending short request messages. Although Octopus, as a network controlling tool, can reconfigure or change the working methodology of a WSN, it does not have an integrated programming tool to develop, simulate, and completely reprogram a sensor node over-the-air. Before deploying the sensor nodes, Octopus requires the user to program the different behavioral codes in the sensor nodes and associate them with a specific request message type. Therefore, these behaviors, which can change the network or individual nodes behavior, can be triggered with the help of short request messages. Similar to [3], Octopus cannot do an off-line visualization and cannot visualize by using data from a WSN simulator.

NetTopo [2] 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 for both scalability and accuracy. It provides a visualization tool 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 NetTopo cannot do live or off-line WSN visualization and also cannot program a sensor node.

TOSSIM [1] 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 [1], 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.

In short, all the tools discussed above are a standalone visualization tool or a programming tool or a simulator. PROVIZ is unique in its approach by providing all these functionalities under a single platform.

III. PROVIZ Design Overview

PROVIZ is a multi-platform, modular, generic, extensible, and scalable programming and visualization framework [5] developed using the QT GUI Framework [12]. 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 [13]. Figure 1 shows the modular PROVIZ framework design, where PROVIZ runs on a host machine and has a *PROVIZ Client*, which can run on a local or remote host and gather packet data using different sniffers. PROVIZ provides two main functionalities. One is the functionality to visualize the data and the other is the functionality to program sensor nodes. For the former, PROVIZ can visualize the data either captured in realtime or from a WSN simulator. The following sub-sections discuss the design of PROVIZ programming and visualization functionalities.

A. Modules for Programming Functionality

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

1) Programming Window: The PROVIZ programming tool has an editor window, which can be used to edit the NesC [10] code of a TinyOS application or a MCL [11] script. The MCL scripting framework provides user-friendly commands for creating new NesC components, interfaces, and wiring, which reduces development efforts.

2) Code Generator: After entering the MCL script in the editor window, the user can use the *Generate Code* option in PROVIZ to generate the NesC code from the MCL script.

3) Code Distributor: After generating the NesC code, the user can use the *Program WSN* option to generate the WSN application binary. The Code Distributor uses the *Communication Interface* to remotely program wireless sensor nodes using wireless code dissemination as discussed below.

4) Communication Interface: After the Code Distributor generates the code image, the Communication Interface is utilized to disseminate the code image. This module also has functionalities in the visualization of the captured data as explained in the next sub-section.

5) Wireless Code Dissemination: For wireless code dissemination, PROVIZ uses the SIMAGE protocol [6]. SIMAGE uses the LQI value to asses 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 like confidentiality and integrity, provided by the CC2420 [14] transceiver module in the sensors (e.g., MicaZ, TelosB).

For programming a WSN, PROVIZ uses the Control Node 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.

B. Modules for Visualization Functionality

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

1) Communication Interface: This interface has a generic packet receive module for the visualization functionality, which can receive packets from different sources such as: a 1) PROVIZ Client with a sensor-based sniffer attached to either a remote or a local host; 2) WSN simulators such as OMNeT

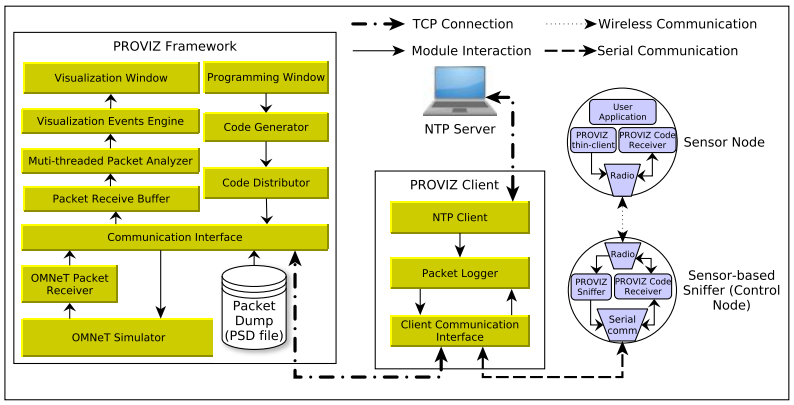

Fig. 1: PROVIZ Architecture

[7]; and 3) Packet trace file, generated by commercial sniffers, e.g., the PSD [8] file format created by the TI SmartRF packet sniffer [8].

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 fetches the network simulator trace and sends it to the Communication Interface.

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

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 [15], 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 computation intensive operation. In order to achieve faster processing and avoid any 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 developers based on the incoming packet arrival rate.

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 sorted multi-map data structure with the time of event as the key. 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. 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.

After starting the visualization, PROVIZ provides a graphical canvas (Figure 2) for visualizing the sensor nodes and packet transfers.

6) 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.

IV. Visualization Tool Features

This section discusses the various features of PROVIZ visualization functionality. Figure 2 shows the screen shot of the PROVIZ visualization tool with a group of infrastructure monitoring sensors sending structural health information [16] 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.

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

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 in the PROVIZ Client. PROVIZ uses this count value associated with each sensor icon to restrict 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

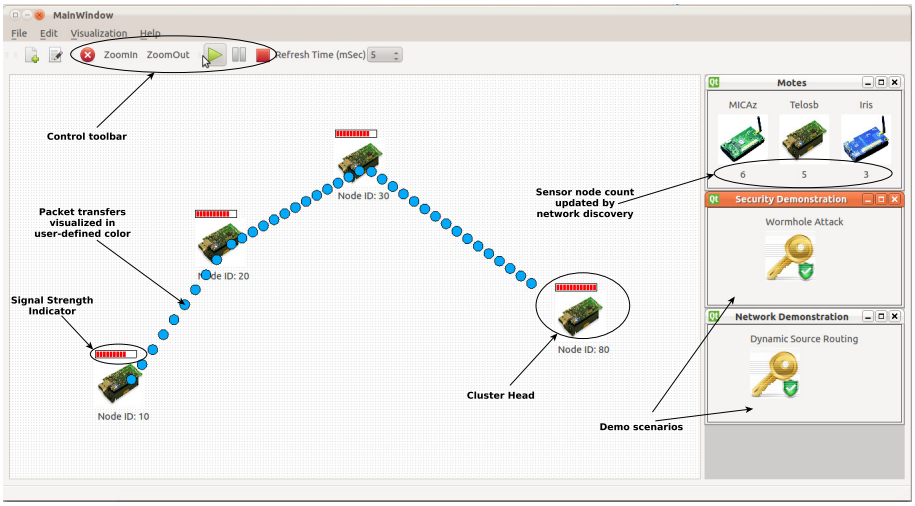

Fig. 2: Screen shot of PROVIZ visualizing an infrastructure monitoring WSN

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.

Demo Scenario Visualization: PROVIZ includes built-in extensible visual demo deployment scenarios, on which users can click and visualize easily as shown in Figure 2. 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.

V. 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. The 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 [8]) and the OMNeT simulator [7]. 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. Also, PROVIZ includes built-in extensible visual demo deployment scenarios that can even be shared among the users in the form of XML files. PROVIZ includes an editor window, which can edit a NesC code and MCL [11] script. PROVIZ is capable of remotely programming the sensor nodes by wireless code dissemination.

Our future work will extend PROVIZ to get live sensor state information like temperature, battery level, etc. We will improve PROVIZ to include a capability to define notifications to the user whenever a predefined network condition is reached. 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. Also, we will extend the PROVIZ programming functionality to support graphical programming using drag-and-drop icon-based programming style for easy and rapid WSN application development.

References

[1] P. Levis, N. Lee, M. Welsh, and D. Culler, "Tossim: accurate and scalable simulation of entire tinyos applications," in *International conference on Embedded networked sensor systems*. ACM, 2003.
[2] L. Shu, C. Wu, Y. Zhang, J. Chen, L. Wang, and M. Hauswirth, "Nettopo: beyond simulator and visualizer for wireless sensor networks," *SIGBED Rev.*, vol. 5, no. 3, Oct. 2008.
[3] L. Ma, L. Wang, L. Shu, J. Zhao, S. Li, Z. Yuan, and N. Ding, "Netviewer: A universal visualization tool for wireless sensor networks," in *IEEE Global Telecommunications Conference*, Dec. 2010.
[4] A. Ruzzelli, R. Jurdak, M. Dragone, A. Barbirato, G. OHare, S. Boivineau, and V. Roy, "Octopus: A dashboard for sensor networks visual control," in *Proceeding of the 14th Annual* ., 2008.
[5] "PROVIZ: A Visualization and Programming tool for WSNs," http://www.ece.gatech.edu/cap/proviz.
[6] R. K. Chandrasekar, V. Subramanian, S. Uluagac, and R. Beyah, "SIMAGE: secure and Link-Quality cognizant image distribution for wireless sensor networks," in *IEEE Global Telecommunications Conference*, Dec. 2012.
[7] "OMNET Simulator for WSN running TinyOS programs," http://www.omnetpp.org/pmwiki/index.php?n=Main.NesCT.
[8] "Ti packet sniffer," http://www.ti.com/tool/packet-sniffer.
[9] "Tinyos documentation," http://docs.tinyos.net/.
[10] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler, "The nesc language: A holistic approach to networked embedded systems," ser. PLDI '03. ACM, 2003.
[11] M. Valero, S. Uluagac, V. Subramanian, R. K. Chandrasekar, and R. Beyah, "The monitoring core: A framework for sensor security application development," in *IEEE International Conference on Mobile Ad hoc and Sensor Systems*, Oct. 2012.
[12] "Qt: A gui development framework," http://qt-project.org/.
[13] "Qt: A multi-platform tool," http://doc.qt.digia.com/qt/supported-platforms.html.
[14] "CC2420 datasheet," http://focus.ti.com/lit/ds/symlink/cc2420.pdf.
[15] "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.
[16] "Infrastructure monitoring wsn," http://wang.ce.gatech.edu/research.htm.