NetEye: A User-Centered Wireless Sensor Network Testbed for High-Fidelity, Robust Experimentation

Xi Ju*, Hongwei Zhang*, Divya Sakamuri†
*Wayne State University, {xiju,hongwei}@wayne.edu
†Ford Motor Company, divya@ford.com

ABSTRACT

Experimentation infrastructures for wireless sensor networks (WSNs) have been widely adopted over the past decade for hi-fidelity validation and rapid prototyping of applications and for discovering and modeling phenomena that have led to improvements in analysis and simulation capabilities. Nonetheless, the community is still looking for more heterogeneous WSN testbeds for purposes such as federation, experiment regression analysis, and improved testbed availability. In addition, most existing WSN testbeds lack robustness and predictability in the presence of failures, and they do not provide user-friendly support for scientific experimentation (e.g., quick repetition of experiments). We develop the *NetEye* testbed as a high-fidelity, robust WSN experimentation infrastructure that provides the ease of scientific experimentation. Consisting of 130 TelosB motes with IEEE 802.15.4 radios and 15 Linux laptops with IEEE 802.11a/b/g radios, NetEye provides a WSN deployment that enables high-fidelity experimentation with single-hop as well as multi-hop WSN networking, and the lab-scale deployment mimics network properties of building-scale and outdoor WSN deployments. Building upon the authors' experience in developing and using WSN testbeds in the past years, NetEye also has enhanced features that enables easy, robust scientific experimentation. In particular, NetEye provide users with a simple, powerful web portal, which streamlines users' experience in the whole lifecycle of scientific experimentation (e.g., experiment scheduling, status monitoring, and data exfiltration). NetEye employs a stable, fault-tolerant state machine to improve the robustness of the testbed in the presence of failures. NetEye also develops a health monitoring service "NetEye Doctor" that monitors hardware and software status; the real-time health monitoring information about the testbed is seamlessly integrated with the lifecycle of experiment scheduling, experiment status monitoring, and experiment data analysis for both robust experimentation and for informed experiment analysis. To address the log-data congestion problem in long-running, large scale experiments, NetEye employs a TDMA mechanism to ensure reliability in exfiltrating experiment data to users. To facilitate quick, repetitive scientific experimentation, NetEye also speeds up the mote programming cycle through fast job status transition and multi-thread-based mote programming. NetEye has been successfully running for over 3.5 years, and it has been integrated with the federated WSN experimental infrastructure *KanseiGenie*. NetEye is widely used by researchers from USA and Asia, and more than 10 experiments are executed through NetEye per day on average.

1. INTRODUCTION

Society has witnessed a dramatic growth in its use of sensing technologies over the last decade or two. The emergent ubiquity of sensing has been fueled by the ability to network "edge" sensors, often in a wireless manner. In particular, a number of cyber-physical monitoring and control applications have been based on Wireless Sensor Networks (WSNs) at the edge that interoperate with enterprise networks and other sensor networks. WSNs are providing not only persistent, fine-grain information, they are essentially becoming programmable "fabrics" — virtualized, service-oriented, commodity platforms that support application evolution and co-existence with other networks that span multiple administrative domains and even national boundaries [15]. Next generation networks embodying such interoperation and programmability include OGC Sensor Web [8], MSN/Virtual Earth extensions for sensor data [2], NEON ecological observatory [7], and Next Generation 9-1-1 [3].

Experimentation infrastructures for WSNs have progressed hand-in-hand with their growth. Early infrastructure consisted of testbeds that enabled fidelity in emulating field deployments, which was lacking in analysis and simulations. Testbeds have been great time savers: they inverted the equation between deployment time and data collection time: today, the former takes minutes while the latter can last from days to years. Testbeds have facilitated testing over multiple sensor data sets, environments, signal processing algorithms, and networking protocols. Importantly, from a science perspective, they have led to discovery of phenomena in low-power radios and validation of models, thereby improving the effectiveness of analysis and simulation tools [15, 18, 21, 16].

Despite the widespread use of WSN testbeds and the evolution of WSN testbeds themselves, the community is still looking for more heterogeneous WSN testbeds for purposes

such as federation, experiment regression analysis, and improved testbed availability. As WSNs are increasingly being applied to mission-critical scenarios (e.g., fire detection and industrial control), it becomes critical to ensure predictable network performance across a wide range of environmental and system settings. WSN testbeds deployed in different environmental and system settings can enable regression analysis of protocol sensitivity and predictability in different use cases, thus serving as a basic tool of designing and analyzing predictable WSN protocols and systems. Therefore, the more testbeds that are deployed, the more capability the federated, heterogeneous testbeds will have in enabling experiment regression analysis and in enabling the design and analysis of predictable WSN protocols [15, 19]. Additionally, the more testbeds that are deployed, the higher availability of testbed resources will be as far as users are concerned; this is especially the case when multiple users need to access testbeds at the same time window (e.g., for working towards a shared conference/journal deadline).

Another issue with most existing WSN testbeds is that they lack robustness and predictability in the presence of failures. A testbed may fail or become unavailable to users for an extended period of time, thus rendering the testbed undependable for users who rely on it for scientific exploration. A testbed may also fail during an experiment; this makes it difficult to analyze experimental results, for instance, whether a packet loss is due to testbed node failure or due to inherent wireless transmission failure. Moreover, existing WSN testbeds do not provide enough user-friendly support for scientific experimentation. For instance, an experiment may well be repeated several times for analyzing the confidence interval and sensitivity of a protocol; yet it may take a long time (e.g., up to 10 minutes for Kansei [14]) for the testbed to program the involved motes for each repetition, and this is the case even if each experiment itself takes a short amount of time.

Towards addressing the aforementioned issues in WSN experimental infrastructures, our work makes the following contributions:

- We develop the *NetEye* testbed as a high-fidelity, robust WSN experimentation infrastructure that provides the ease of scientific experimentation. Consisting of 130 TelosB motes with IEEE 802.15.4 radios and 15 Linux laptops with IEEE 802.11a/b/g radios, NetEye provides a WSN deployment that enables high-fidelity experimentation with single-hop as well as multi-hop WSN networking, and the lab-scale deployment mimics network properties of building-scale and outdoor WSN deployments.

- Building upon the authors' experience in developing and using WSN testbeds in the past years, NetEye also has enhanced features that enables easy, robust scientific experimentation. Based on understanding testbed users' needs and behavior, in particular, NetEye provide a simple, powerful web portal, which streamlines users' experience in the whole lifecycle of scientific experimentation (e.g., experiment scheduling, status monitoring, and data exfiltration). NetEye employs a stable, fault-tolerant state machine to improve the robustness of the testbed in the presence of failures. NetEye also develops a health monitoring service "NetEye Doctor" that monitors hardware and software status; the real-time health monitoring information about the testbed is seamlessly integrated with the lifecycle of experiment scheduling, experiment status monitoring, and experiment data analysis for both robust experimentation and for informed experiment analysis. To address the log-data congestion problem in long-running, large scale experiments, NetEye employs a TDMA mechanism to ensure reliability in exfiltrating experiment data to users. To facilitate quick, repetitive scientific experimentation, NetEye also speeds up the mote programming cycle through fast job status transition and multi-thread-based mote programming. In this paper, we explain the aforementioned features and design of NetEye in great detail so that NetEye can serve as a reference design and implementation for future efforts in WSN testbed design and deployment.

- NetEye has been successfully running for over 3.5 years, and it has been integrated with the federated WSN experimental infrastructure *KanseiGenie* [19]. NetEye is widely used by researchers from USA (e.g., Stanford University, University of Southern California, Ohio State University, and Wayne State University) and Asia (e.g., City University of Hong Kong, Southeast University), and more than 10 experiments are executed through NetEye per day on average.

The rest of the paper is organized as follows. We present the NetEye system architecture and the NetEye control software in Sections 2 and 3 respectively. We discuss related work in Section 4, and we make concluding remarks in Section 5.

2. NETEYE SYSTEM ARCHITECTURE

We design NetEye as a high-fidelity WSN testbed that mimics network properties of building-scale and outdoor testbeds with an indoor, lab-scale deployment. The ease of lab-scale deployment and the high-fidelity of NetEye make NetEye readily replicable to other institutions, to extend the reach of WSN testbeds and to scale up federated WSN experimental infrastructures. In what follows, we first present the system architecture of NetEye, then we discuss the high-fidelity of the NetEye testbed.

2.1 System architecture

To enable high-fidelity, fine-grained measurement of WSN protocol behavior in a large-lab setting, we design NetEye as a three-tier architecture consisting of a NetEye server, 15

Linux laptops, and 130 TelosB motes. More specifically, NetEye is deployed in a lab room of Computer Science Department as is shown in Figure 1 . This deployment con-

Figure 1: NetEye testbed

sists of 15 distributed laptops designed for the purpose of complex data processing and powerful computing, and 130 sensor nodes placed on a 9×17 rectangular grid wooden benches with around 2 feet spacing. Both laptops and sensor nodes are sub-controlled elements of NetEye, but sitting at the different control level in NetEye system architecture, which will be further discussed in section 3.

In NetEye, laptop is a Dell Vostro 1400 model with a Intel dual core processor, 1 GB Ram. It is equipped with IEEE802.11a/b/g wireless radio. The operating system on laptop is the Linux Fedora that runs on a 24/7 basis. The 15 laptops are physically arranged on 15 wooden benches which are customized for NetEye deployment and spaced from each other properly.

The sensing element — TelosB mote in NetEye, is an open source platform designed to enable the application deployment for experimentation. It is an IEEE802.15.4 compliant device designed by UC Berkeley and manufactured by Crossbow Solutions. Each TelosB mote is equipped with a T1 MSP430 microcontroller, a 2.4 GHz Chipcon CC2420 radio, 4MHz 8bit CPU, 48KB instruction memory, 10KB RAM and a USB interface. The radio's reliable transmission range is roughly between 20m to 30m when placed in an indoor environment. Each mote accommodates a light sensor. The motes run TinyOS 2.1 [11] that is a light-weight and event driven operating system which implements the networking stack and communication with sensors, and provides the programming environment for the platform. Figure 2 shows the system architecture of NetEye.

The question of prototyping the NetEye system as mentioned in section 1 is how to wire and configure these hardwares for the regulatory deployment. The sensing element should be managed by laptops in an efficient way so that the whole testbed could be further partitioned into multiple autonomous sub-systems. NetEye platform addresses the challenges by subdividing the sensor nodes into different groups.

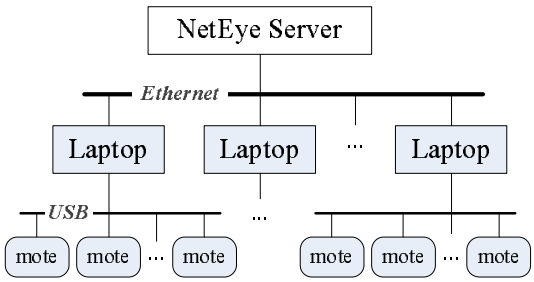

Figure 2: NetEye System Architecture

These groups are the collection of bunch of adjacent sensor nodes representing a small scale of partitioned and self-admined sensing infrastructures. Each individual group will be in the prototyping phase assigned to one management laptop. Laptop with its affiliated group of sensors conforms to a sensing "cluster" as is a basic entity of the NetEye testbed. In order to facilitate the topology control and improve the programming efficiency, the number of sensor nodes in one cluster is fine tuned and also balanced among adjacent clusters. In the practical setup of NetEye, there are 4 to 12 motes connected to one laptop through the USB interface depending on the topology design that intends to provide a multi-hop network for experimentation. Figure 3 shows that six

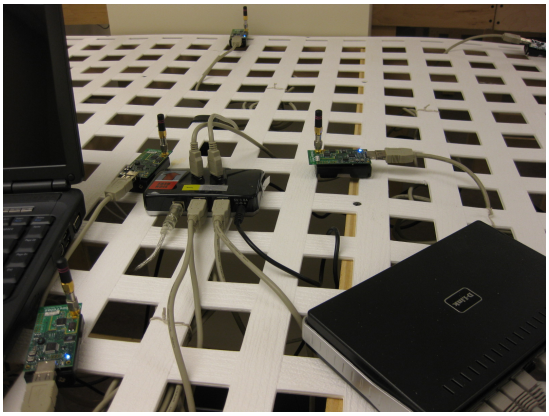

Figure 3: NetEye sensor nodes deployment

sensor nodes are collected with one laptop. The exact number of sensor nodes connected to laptop is also optimized in order to prevent programming errors on sensor nodes, and reduce the number of USB hubs and cables in use.

Laptops in each column of sensing infrastructure are connected through a Gigabit Ethernet switch to an Ethernet back-channel network that provides high-bandwidth connectivity to and from the sensor nodes for management operations. Apart from the sensing infrastructure, there is a Linux Fedora server working as a controller of the whole testbed. This NetEye server connects to the local Ethernet back-channel network of sensing infrastructure and also has another Ethernet interface through which to connect to the public network. From the system architecture's view, NetEye server bridges

a remote access between sensing elements and users for creating, editing, and scheduling jobs during experimentation.

2.2 High-fidelity of NetEye

Network environment. NetEye is deployed in a lab room of Maccabees building at Wayne State University. It provides an indoor wireless environment for WSN infrastructure. As is known that the WSN application and protocol performance are heavily affected by the featured wireless environment. Therefore, it is crucial to study these dominating factors in NetEye environment. In this section, we study the measured CC2420 radio model and its behavior in the NetEye sensing background. Additionally, we study the background noise power in the NetEye environment. Figure 4 shows the radio model, as represented by the relation between packet delivery ratio (PDR) and Signal-to-Interference plus Noise Ratio (SINR) at a receiver, for a typical TelosB sensor node in NetEye. This measured radio model captures the CC2420 radio behaviors in the NetEye environment, which is useful for determining the SINR threshold for satisfying certain wireless link reliability.

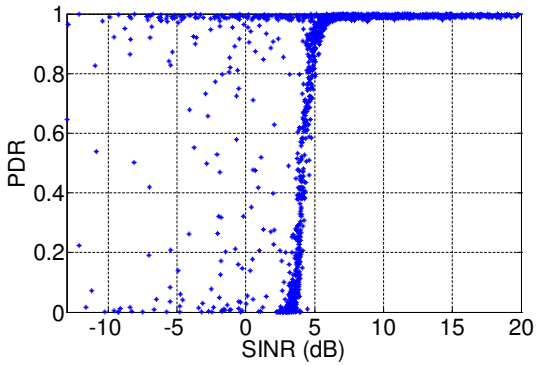

Figure 4: PDR vs. SINR in NetEye

Figure 5 shows the relation between PDR and SINR in Motelab. Figure 6 and Figure 7 show the histogram of the background noise power in NetEye and Motelab. We can see that there is a high degree of variability of background noise power in NetEye. And the testbed setup is able to mimic building-scale deployment of other environment. Thus the testbed enables user to do experimentation in non-uniform settings.

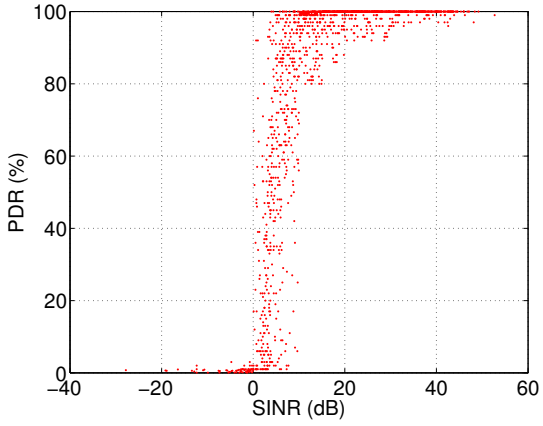

Figure 5: PDR vs. SINR in Motelab

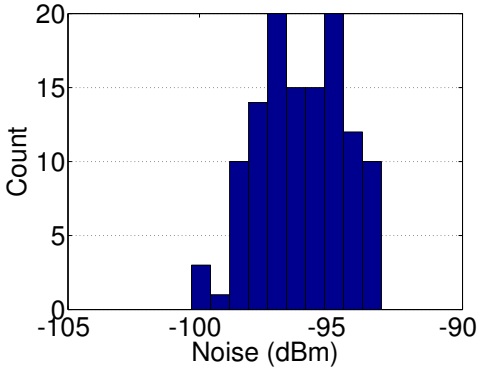

Figure 6: NetEye background noise histogram

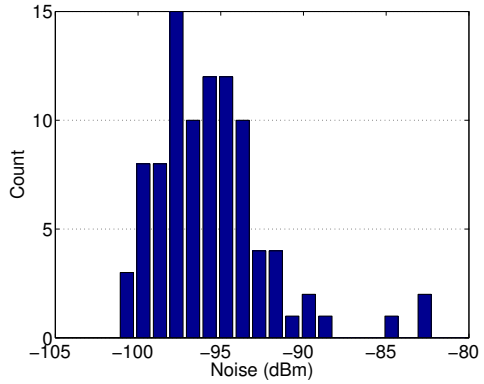

Figure 7: Motelab background noise histogram

Topology control and attenuator design. In order to have a realistic connectivity of multi-hop network, we should seek a solution to emulate the link properties observed in the outdoor environment so that the resulting Packet Delivery Rate (PDR) in testbed will follow the general characteristics of the various environment. One possible solution to have control of network topology is to apply an attenuator on antenna and set the transmission power level properly. Attenuator design is supposed to reduce the radio communications over a large range for topology control purpose.

Topology control has been one of the key design issues for setting up NetEye. The desired and controllable topology should have the variability that embodies the measured link characteristics in both small, medium and large scale of WSN testbeds. And also, it should support the multi-hop transmission between sensor nodes in NetEye indoor environment. Thereafter, to address this issue, we instrument each sensor node with an 3dB SMA(SA3-03) attenuator, an omnidirectional RH Series Monopole antenna, and power control to emulate the different scale of network. Figure 8 and Figure 9 show the SMA attenuator and the reduced

Figure 8: SMA(SA3-03) attenuator

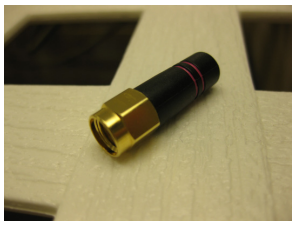

Figure 9: Reduced height monopole antenna

height monopole antenna used in NetEye. Through attenuator design and power control, we can configure any experiments across the NetEye substrates to have the connectivity as high as 6 hops.

One of the important issues when we started building NetEye is that sensor devices may co-exist with other wireless devices, such as laptops equipped with IEEE 802.11 radio. It is observed that the RF spectrum of channels 15-24 of 802.15.4 overlap with the channels 1-11 of 802.11, which shows that channel 25 and 26 in the 802.15.4 standard are interference free from any channels of the 802.11. Channels selected by experiments should be correctly set on the sensor nodes to provide an interference free or controlled environment.

Channel 26 is set on the sensor nodes to avoid the interference with any other channels in wireless environment. The following experimentation is conducted to choose the minimum power level while it still guarantees reliable PDR in a multi-hop network. The contribution of this experiment is using the applied 3dB attenuator to limit the transmission range.

To achieve the multi-hop network in NetEye, a NesC program that broadcasts 10k packets is burned on one of the sensor nodes. All the other nodes are programmed with a receiver routine in which the experiment data is collected and dumped to the USB interface. Figure 10 shows the PDR versus link length measured in feet. Figure 11 shows the PDR versus distances at different power levels. We can see that based on the topology control and attenuator design, NetEye exhibits diversity of link reliability and creates multi-hop wireless environment.

3. NETEYE CONTROL SOFTWARE

To enable robust, efficient utilization of the NetEye testbed, a control software is needed for purposes such as scheduling WSN experiments, allocating testbed resources, exfiltraing

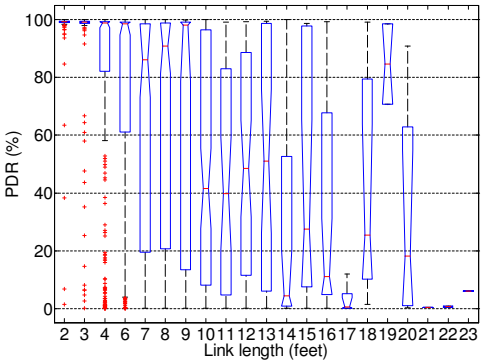

Figure 10: PDR vs. Distances in NetEye

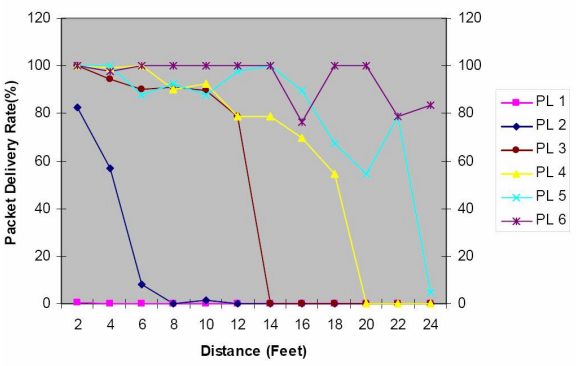

Figure 11: PDR vs. Distances at different power levels in NetEye

experimental results, and monitoring testbed health status. In what follows, we first present the design objectives of the NetEye control software, then we discuss the architecture and components of the NetEye control software.

3.1 Design objectives

In order to meet the requirement of deploying NetEye testbed and obtain the goal of achieving robust programming platform, proper software design and implementation is crucial to the WSN testbed. The software design is supposed to be handled in such a way that user can easily access the testbed and confidently rely on the outcome of the testbed facility. To help obtain the accurate results and provide the robust application services, the testbed setup, especially the developed testbed platform, should sustain its controllability and reliability, supply user with a high-fidelity and realistic wireless environment. The main implementation goals to be considered while developing any WSN testbed are as follows.

Stable job-status state machine, fault-tolerant recovery mechanism. It is necessary that the wireless sensing platform implements a stable software state machine in consideration of tracking any job status in real time. The state machine shall be designed and implemented as the core of the software architecture, since the experiments are driven

by the core of the state machine that initiates the actions as needed to start, manipulate, and maintain the status of substrates. For example, a submitted job can only be "reserved" if all selected sensor nodes associated with the job report to the control module of state machine that they are idle and ready for programming. At any time, a job state is in one of a fixed set of discrete states defined by a finite state machine. This finite state machine should be working as a kernel that converges all the system actions in one state at a time. Most importantly, it should also be governing all functions of the experiments scheduling engine. Because of the importance of time in the state machine management, server and laptop clocks should require to be loosely synchronized using a time service such as NTP. While the state machines are robust to timing errors, unsynchronized clocks shall not lead to anomalies from the perspective of one or more laptops.

Apart form the state transition defined by the state machine, a recovery mechanism should be well considered. Because with regards to the robust experimentation design, it is required and mandated by wireless sensing platform that consistency of the whole system shall be achieved. In a running system, it happens very often that if one of the system elements quits inappropriately or unintentionally due to the hardware errors or software glitches, the experimentation could collapse. A well designed recovery mechanism should tolerate the system error or devices off-line, and have the capability of recovering the system state to the right one at any time. For an instance, if one of the laptops are accidentally powered off, the job could be terminated by user or its status could be recovered to the previous correct state if the laptop is back on-line again.

Real-time sensing array diagnose. A health monitoring service has to be developed to diagnose the system and report the run-time health information to user. A large scale testbed usually contains lots of wired or wireless hardware devices. It is quite common for regular faults and unexpected errors to occur in the middle of user's experimentation, which may affect the quality or even the correctness of output data. If a testbed such as NetEye can provide diagnostic information about the sensing elements of substrate array and assist users in configuring experiments to avoid using failed devices,the experiments can be kept with high fidelity and completed with fewer errors caused by the platform failures.

There are several critical system requirements that need to be taken into account while implementing the "doctor" services for the NetEye testbed, because on one hand they assist administrator in building and managing testbed, on the other hand they guide users in configuring their experiments.

a). Correctness and reliability: The diagnostic information obtained in testbed should be correct and testified. Incorrect data leads to misuse of the testbed facility. Further more, the status of each sensing element in the network should be acquired reliably. The diagnostic service should have the fidelity in a manner that experiment results will not be affected by the scheduled experiment.

b). Real-time and efficiency: The diagnostic information should be pushed to the user at run time. What's more, it should also not take too long to finish one inspecting cycle and cost too many resources during its execution.

c). Independence and adaptability: The diagnose daemon should not interfere or be interfered by other running code, and the diagnose code itself should adapt to any software changes or upgrades in the testbed. It should involve less effort to adapt to the new system even if the devices, interfaces, and the configurations have significantly changed.

d). Handling heterogeneity: Current WSN testbed deployment often has various type of devices with different capabilities and limitations. The diagnose daemon should be able to monitor the different hardware components such as Stargate that operates on an embedded Linux system.

Policy-based resource scheduling of best-effort experiments. The challenge for resource scheduling is how to assign all the contending jobs to NetEye at one particular time. In the current implementation, most of services may only need the best-effort model. This model represents the scheduling policy which should be employed by the scheduling code as the resource allocation algorithm.

The amount of resources that can be reserved by an application, the relative priority of best-effort services, and the proportion of available global resources will have to be subject to the control policy. Any design of policy-based resource scheduling needs to separate the resource allocation from the low level resource control.

Simple user and administrative tools. A graphical web interface should be presented to each user for easy access and control of her experiments. These tools should assist user in uploading her configuration files, customizing desired topologies, specifying start and end time, as well as scheduling experiments. Sensor nodes selection for the topology customization should be visualized for ease of use. User tools should give users full control of their experiments, including monitoring the job status, cancelling a job, tracing the sensor node programming logs, etc. These tools should also be integrated with the diagnostic module to help user choose "healthy" sensor nodes and validate the quality and correctness of experiment results. Tools should be implemented in such a way that user has the privilege of collecting running time data and accessing the historical data. In addition, injection tools should be provided to enable online interaction between user application and the experimenting sensor nodes.

Administrative tools must be necessary for testbed maintenance. The tools should have privilege of managing users and assigning user quotas based on different user groups. Since the diagnostic tools serve both administrators and users, it should certainly facilitate the process of maintaining sensing substrates health information. Administrative tools can utilize diagnostic information of system to "mute" sensor

nodes or bring nodes back online without any manual work of intervention.

3.2 Software architecture

A suite of software tools are developed for managing the sensing substrates of NetEye array and allowing a large community of users to share testbed access. The goal of designing and implementing these software tools is to provide a user-friendly interface and a robust experimentation platform. The implementation of NetEye software obeys the development mode of the client/server communication in which PC server runs as controller while laptops act as the subordinates.

The software suite consists of several major components including the web-based experimentation interface (NetEye web portal), MySQL experiment database, experiment scheduler, job controller, subordinate controller, injector, data and programming logger, and NetEye doctor. In the concrete practice of building wireless sensing platform, NetEye software development also relies on the open-source Linux-Apache-MySQL-PHP/PERL (LAMP) implementation technology of which both Kansei and MoteLab platform are basis. Figure 12 shows the NetEye software architecture in which the

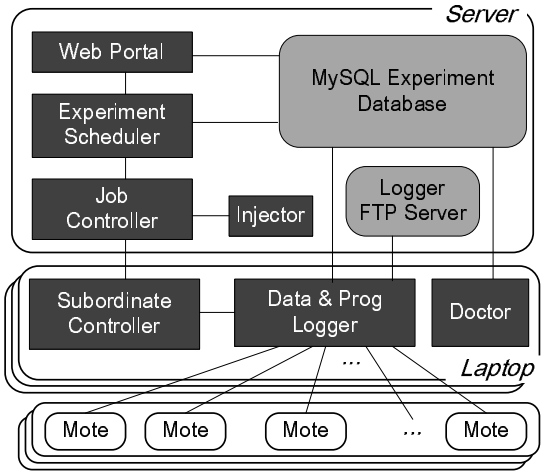

Figure 12: NetEye software architecture

software components are functioning at different levels in which they direct the experiment control work flow from web interface to sensor nodes, and then going back to the NetEye controller. In this figure, basic elements of the communication between these components are depicted and a typical NetEye testbed activity is also illustrated.

3.3 Software components

In this section, we discuss the technical issues on implementing the aforementioned software components in more detail and explain how one job is involved in the typical processing phase of NetEye platform. In the meanwhile, we describe several unique features that are designed and crafted towards improving user experience and enhancing the system performance of the platform.

Stable, fault-tolerant state machine of NetEye. Figure 13 shows the state machine. The "hollow-circle" means

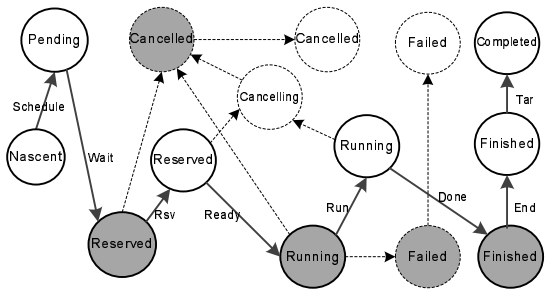

Figure 13: NetEye software state machine

that this job state is maintained by server, while the "filled-circle" denotes the state kept by sensor nodes. In this figure, some of state transitions shall be triggered by user actions, for an instance, user wants to cancel a job during the running state, and later on will know it being cancelled if all involved sensor nodes give a "cancelled" reply. Other state transitions are driven by timer events, since job status activates and expires at specified times. For instance, the job controller has to schedule to shutdown the experiment at the end of the job.

NetEye web-based interface. The NetEye web-based interface are PHP and javascript generated pages that serve as experimentation web portal for job creating, scheduling and data collection. Once user gets access to NetEye and logs into the system through the web interface, the job management web page provides a summary of pending, reserved, running, and completed experiments. The common status of individual sensor nodes can be retrieved through the NetEye Doctor web page where we implement the node status tracking scripts that interact with diagnostic daemons in the background. In dashboard of the NetEye Doctor page, user can easily track the node status about "busy", "free", and "unavailable" information, before configuring and creating a new job. A powerful interface for scheduling experiments is implemented to allow users to access and manipulate the sensing array in a platform independent way. This interface provides rich options for users to compose the experiments and orchestrate their needs. It exhibits to users a visualized topology and miniature the NetEye testbed where users are given some options to select sensor nodes and include them in their further experiments. This interface is also integrated with the diagnostic daemon in such a way that prior to sensor node topology being completely pushed onto users, unavailable sensor nodes are automatically disabled and will not possibly be chosen on the scheduling web page. Each user is affiliated with a pre-defined group which ties the user quotas to the scheduling web page. Figure 14 shows the NetEye experiment scheduling web page.

Here is one example that illustrates how these functionalities are presented to a user to facilitate job scheduling,

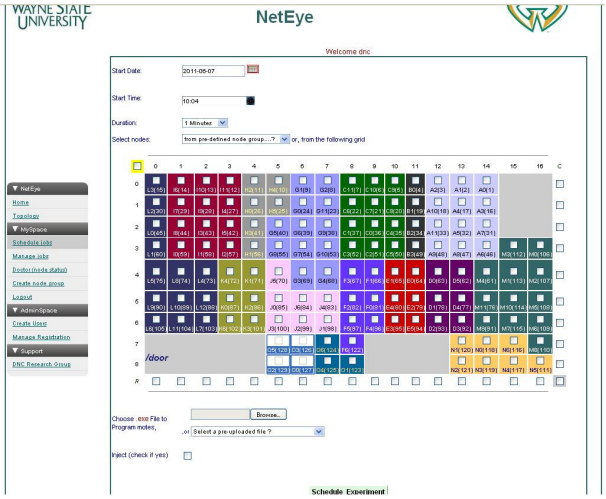

Figure 14: NetEye experiment scheduling interface

and how simple an autenticated user gets his first job running within only a few steps of configutaion. After a user is authorized to access the NetEye system through the web portal, he will be directed to a testbed status web page where detailed information about testbed status pops up in front of the user, which clearly enumerates the current users and their active jobs. According to the real-time testbed usage, he is able to check out node status and get to know which set of nodes have been used and cannot be included in his experiments. Then, he goes ahead and clicks on the "schedule job" button on the web portal. Once the user is directed to the scheduling page, he is able to specify when his job shall start and end, which set of nodes will be selected in his job. Furthermore, the user is given the option to upload his complied executable binary files to the testbed. The final step is to double check the job specification and hit the "schedule" botton. The job will be on its way automatically. The user can also check job status, stop a job, and watch job logs in real-time.

MySQL experiment database. In NetEye, MySQL database is used to store all the information necessary for running experiments and monitoring sensing substrates. For this purpose, there are two categories of information stored in the database. One is related to the jobs, which includes the start/end time, input files, executables, states of state machine, and data log links. The other is about the testbed status. The user information, laptop configuration, and the sensor nodes status are major parts of the stored data.

Experiment scheduler. When one submits an experiment on the web interface, a job that fully describes the experiment is created automatically. Then the experiment scheduler takes over the control of the job. All submitted jobs are queued by the experiment scheduler at run time and scheduled based on the FCFS or priority policy. According to the policy, experiment scheduler makes the decision on which one will be scheduled and inserted to the job table first in MySQL database.

Job controller. The job controller running as cron tasks will be immediately notified if any job has been scheduled on testbed. It is implemented by Perl and Shell scripts, and responsible for setting up, launching, and tearing down the scheduled jobs. As a controller daemon running at NetEye server, it maintains the job state machine on server and issues corresponding commands to the subordinate controller to program sensor nodes. The job controller dumps the job state to the database periodically so that even if the server goes off-line, the job status won't be lost and job can be resumed when the server is on-line again.

Subordinate controller. Based on the similar design as job controller, the subordinate controller runs as a distributed cron task that involve preparing sensor node programing environment, issuing programming commands, and killing the related processes. It maintains the job state for the sensor nodes so that the job status on server will be consistent and updated properly. For example, the "reserved" job status on server will migrate to "running" only if all the subordinates controllers finish programming sensor nodes and report this information to the job controller. Whenever laptops need to be rebooted due to the hardware or software problem, the subordinate controller is also capable of recovering its job status. Figure 15 shows how job controller coordinates with

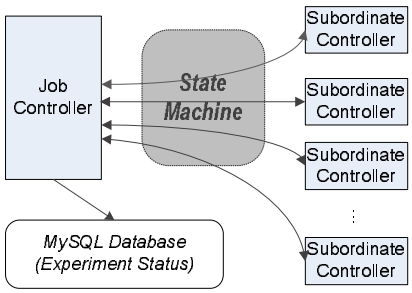

Figure 15: Coordination between Controller and Subordinate Controller

subordinate controller in NetEye.

Injector. The injector is invoked by the job controller if there is a data injection request specified on the web interface. The injector provides an on-line service that takes the form of compile-time data forwarding. The injection service is basically derived from the pre-defined injection file supplied by the user that specifies when to inject what data into which sensor nodes. Thus, the content of the injection file is computed off-line and already determined at compile-time. According to the content, the injector builds a connection pipe between server and the involved sensor nodes through serial forwarders to forward data at run time. Figure 16 shows the format of injection file, and an example in which data transmission happens concurrently in a unidirection from mote-65 to mote-76, and from mote-121 to

format	Target Laptop IP	UART ID	<Unlimited Sender Receiver Pairs>						Time to Fire
			Sender Mote ID	Receiver Mote ID	…	…	Sender ID	Receiver ID	
example	172.30.10.13	5	65	76			121	124	3000ms

Figure 16: Injection File Format and Example

mote-124 at global time 3000 milliseconds.

Fast real-time data and programming logger. The data logger takes responsibility of programming sensor nodes, logging experiment data, and uploading logs to server. In a large scale WSN testbed, it may take a bit long time on reprogramming all sensor nodes. To address this problem, data logger utilizes the multi-thread mechanism to program sensor nodes concurrently, which confines the programming time in less than 1 minute for a 130 sensor node testbed. Figure 17 shows the impact of using Multi-thread mechanism

	Number of Motes	Programming Time
Without multi-thread	10	340s
With multi-thread	130	40s

Figure 17: Programming speed comparison

on mote programming speeds when complied code size of TinyOS executable is around 80k.

Another challenge for data logging is that job involved in a long-term running experiment may have huge data output, which causes the log uploading processes to compete the limited TCP bandwidth between server and laptops. To deal with this problem, a TDMA based file uploading mechanism has been designed to mitigate the contention.

The programming errors on sensor node happen very frequently, which should be observable to users in real time because this affects the correctness of experimentation. Programming logger tracks the sensor programming information and push them to the web interface, so that user can have access to the log in real time. If anything goes wrong with the sensor node programming, user could consider to cancel jobs instead of waiting for a long time to check the experiment results.

NetEye doctor. The NetEye doctor runs periodically as diagnostic daemons taking charge of monitoring the sensor nodes health status. To prevent the regular scheduled jobs from being interfered or interrupted by the diagnostic daemons, we design a tiny cron task to keep an eye on the status of any job in NetEye. If there is no job running on the preferred sensor nodes, the diagnostic daemons will chip in to check the node status and dump the real-time information about sensor nodes to the MySQL database. Besides, the diagnostic daemons has been actively integrated with the web interface to advertise the real-time node status to users. Figure 18 shows the NetEye Doctor component and its interaction with MySQL database.

Here is a use case of NetEye doctor in realistic experimentation, which exemplifies the robust feature and also exhibits the real-time error tracing functionality. Once a job is submitted through the NetEye webportal, job status and events will be logged and provided to the user during transitions of job status. For example, if the user would like to check how many nodes have been programmed and been active, the best place to go is the doctor page. Nodes' status information cleary shows the node running status. If nodess status meet the user requirement, he can continue his experimentation further, for instance, programming another set of nodes as controller that communicate with the previously scheduled nodes. What's more, if the user wants to check whether nodes are programmed correctly, he may check the programming statics where he can decide if he needs to stop the job or have it continue running.

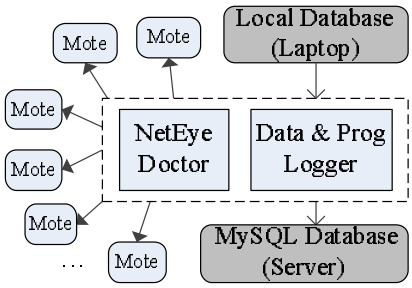

Figure 18: NetEye Doctor

4. RELATED WORK

This work is an enhancement of the testbed deployment and software suite reported in the Master Thesis of Divya Sakamuri [17]. The improved system has been in use and going through heavy-duty unit-tests for many years. There are also a number of other wireless sensor network testbeds such as MoteLab[20], Kansei[14], TutorNet[9], Mirage[13], Indriya[1], and WISEBED[12]. In MoteLab, Kansei, Indriya, and WISEBED, the experiment specification, similar to NetEye, is done via web interfaces where users define the number and types of resources needed, the program executables to be uploaded, and the data metrics to be collected. Other testbeds, such as TUTORNET, Vinelab[10], and Motescope[6] only provide script-based interfaces implemented by some homebrew tools. WISEBED and Kansei define a set of web service APIs that testbeds can implement to be compatible with one another. Based on the web service APIs, clients (e.g., a web interface or a script) invoke these services to run experiments and to interact with the testbeds. To alleviate the users' programming burden, NetEye simplifies these steps to include most of the experimentation requests and job customization capabilities in a unified web interface. Users have the freedom to choose the heterogeneous substrates, track the substrate status, and launch their jobs at any time.

The FRONT[4] testbed uses a calendar based approach for reservation and scheduling in which one experiment takes place at any given time and users gets access to the testbed after the current job finishes. To address the resource con-

tention issues, Intel Research at Berkeley has developed the Mirage testbed that applies the microeconomics theory to arbitrate resource allocation among competing users. Users has to bid for resources on specified auctions through virtual currency. This scheduling policy can't guarantee that users request is satisfied or can be delayed whenever user issues a job request through the auction system. User may have to manually re-issue the bid request and get the confirmation from Mirage site if resources are available. NetEye utilizes FCFS and priority-based policies to schedule experiments, which enables the immediate resource allocation as soon as resources become available. This policy basically guarantee that each request may be met instantly or be delayed for running after some time period.

The Kansei testbed utilizes the simlimar scheduling policy to allocate the resources. Nevertheless, user may have no access to what specific nodes are occupied, or idle. And the real-time doctor and logging functionalies are not fully realized, which renders no check ponit for users in running time. Another feature that may be missing for Kansei and other testbeds is if subcontrollers of sensor nodes failed, whole job may fail accordingly. NetEye provides the failure recovery mechanism which can detect the subcontrollers offline events. When subcontrollers are up running, the job status is recovered automatically. And the logged data is still kept in disk, which can be downloaded via the NetEye web portal.

In most testbeds, before an experiment is executed, the devices have to be reprogrammed. Typically, reprogramming uses a wired back-channel which may take a long time to get all sensor nodes programmed. NetEye testbed implements a multi-thread tool to improve the programming speed.

Different from the MoteLab implementation on how users interact with sensor nodes, NetEye provides the experiment execution control that allows for controlling the experimentation flow, e.g. by injecting events or sensor readings according to pre-compiled injection file.

The experiment monitoring and fast data collecting services at run time allow users to debug experiments, to follow the progress of the execution or to store performance statistics for job control. Only a few testbeds such as Kansei and NetEype provide the real time diagnostic information about sensor nodes to guide users' decisions.

5. CONCLUDING REMARKS

We have developed the high-fidelity, robust WSN testbed NetEye. NetEye consists of 130 TelosB motes with 802.15.4 radios and 15 Linux laptops with 802.11a/b/g radios, and it enables high-fidelity experimentation with single-hop as well as multi-hop WSN networking. The constrol software of NetEye enhances existing WSN testbed control software (e.g., that of Kansei) by employing a stable, fault-tolerant state machine, an integrated health monitoring service, and mechanisms for fast experiment deployment and data exfiltration. NetEye has been successfully running for over 3.5 years, and it has been integrated with the federated WSN experimental infrastructure KanseiGenie [19]. NetEye is widely used by researchers from USA and Asia, and more than 10 experiments are executed through NetEye per day on average. Besides the TelosB motes and the Linux laptops of NetEye, we are currently deploying 20+ LOTUS motes [5] into NetEye, and we expect the LOTUS motes to be available for experimentation within half a year. Having more computation, memory, and storage capacity than the TelosB motes, the LOTUS motes will enable experimentation with advanced WSN protocols (e.g., real-time routing protocols) that require more platform resources, thus enabling designing and analyzing protocols that are important for mission-critical WSN applications such as industrial control.

Acknowledgment

This work is supported in part by NSF awards CNS-1136007, CNS-1054634, GENI-1890, and GENI-1633, as well as grants from Ford Research and GM Research. The design and implementation of NetEye has also benefited from the authors' involvement with the Kansei and KanseiGenie development.

6. REFERENCES

[1] Indriya testbed. http://indriya.comp.nus.edu.sg/.

[2] Microsoft Research SensorMap project. http://atom.research.microsoft.com/sensormap/.

[3] Next generation 9-1-1. http://www.its.dot.gov/ng911/pdf/USDOT_NG911_FINAL_System_Design.pdf.

[4] FRONT: Front range observational network testbed. https://wiki.ucar.edu/display/iwrf/Front+Range+Observational+Network+Testbed+\%28FRONT\%29.

[5] LOTUS: An advanced wireless node platform. http://www.memsic.com/support/documentation/wireless-sensor-networks/category/7-datasheets.html?download=186\%3Alotus.

[6] Motescope wireless sensor networks. http://smote.cs.berkeley.edu/motescope.

[7] NEON ecological observatory. http://www.neoninc.org/.

[8] OGC sensor web. http://www.opengeospatial.org/pub/www/ows6/index.html.

[9] Tutornet: A tiered wireless sensor network testbed. http://enl.usc.edu/projects/tutornet.

[10] Vinelab. http://www.cs.virginia.edu/mailman/listinfo/vinelab.

[11] TinyOS. http://www.tinyos.net/.

[12] I. Chatzigiannakis, S. Fischer, C. Koninis, G. Mylonas, and D. Pfisterer. Wisebed: an open large-scale

wireless sensor network testbed. *Sensor Applications, Experimentation, and Logistics*, pages 68–87, 2010.

[13] B. Chun, P. Buonadonna, A. AuYoung, C. Ng, D. Parkes, J. Shneidman, A. Snoeren, and A. Vahdat. Mirage: A microeconomic resource allocation system for sensornet testbeds. In *Proceedings of the 2nd IEEE workshop on Embedded Networked Sensors*, pages 19–28. Citeseer, 2005.

[14] E. Ertin, A. Arora, R. Ramnath, M. Nesterenko, V. Naik, S. Bapat, V. Kulathumani, M. Sridharan, H. Zhang, and H. Cao. Kansei: A testbed for sensing at scale. In *IEEE/ACM IPSN/SPOTS*, 2006.

[15] X. Ju, H. Zhang, W. Zeng, M. Sridharan, J. Li, A. Arora, R. Ramnath, and Y. Xin. LENS: Resource specification for wireless sensor network experimentation infrastructures. In *ACM International Workshop on Wireless Network Testbeds, Experimental evaluation and Characterization (WiNTECH)*, 2011.

[16] H. Lee, A. Cerpa, and P. Levis. Improving wireless simulation through noise modeling. In *ACM/IEEE IPSN*, 2007.

[17] D. Sakamuri and H. Zhang. Elements of sensornet testbed design. Handbook of Sensor Networks, Yang Xiao, Hui Chen, and Frank H. Li (editors), World Scientific Publishing Co, 2009.

[18] D. Son, B. Krishnamachari, and J. Heidemann. Experimental analysis of concurrent packet transmissions in low-power wireless networks. In *ACM SenSys*, 2006.

[19] M. Sridharan, W. Zeng, W. Leal, X. Ju, R. Ramnath, H. Zhang, and A. Arora. From Kansei to KanseiGenie: Architecture of federated, programmable wireless sensor fabrics. In *ICST TridentCom*, 2010.

[20] G. Werner-Allen, P. Swieskowski, and M. Welsh. Motelab: A wireless sensor network testbed. In *IEEE/ACM IPSN/SPOTS*, 2005.

[21] M. Zuniga and B. Krishnamachari. An analysis of unreliability and asymmetry in low-power wireless links. *ACM Transactions on Sensor Networks*, 3(2), 2007.