Simulation-Based Augmented Reality for Sensor Network Development [*]

Ye Wen Wei Zhang
Department of Computer Science
UC, Santa Barbara
{wenye,wei}@cs.ucsb.edu

Rich Wolski
Department of Computer Science
UC, Santa Barbara
rich@cs.ucsb.edu

Navraj Chohan
Department of Computer Science
UC, Santa Barbara
nlake44@gmail.com

Abstract

Software development for sensor network is made difficult by resource constrained sensor devices, distributed system complexity, communication unreliability, and high labor cost. Simulation, as a useful tool, provides an affordable way to study algorithmic problems with flexibility and controllability. However, in exchange for speed simulation often trades detail that ultimately limits its utility. In this paper, we propose a new development paradigm, *simulation-based augmented reality*, in which simulation is used to enhance development on physical hardware by seamlessly integrating a running simulated network with a physical deployment in a way that is transparent to each. The advantages of such an augmented network include the ability to study a large sensor network with limited hardware and the convenience of studying a part of the physical network with simulation's debugging, profiling and tracing capabilities. We implement the *augmented reality* system based on a sensor network simulator with high fidelity and high scalability. Key to the design are "super" sensor nodes which are half virtual and half physical that interconnect simulation and physical network with fine-grained traffic forwarding and accurate time synchronization. Our results detail the overhead associated with integrating live and simulated networks and the timing accuracy between virtual and physical parts of the network. We also discuss various application scenarios for our system.

Categories and Subject Descriptors

I.6 [**Simulation and Modeling**]

General Terms

Experimentation, Performance

[*]This work was supported by grants from Intel/UCMicro, Microsoft, and the National Science Foundation (No. EHS-0209195 No. CNF-0423336, and No. NGS-0204019).

Keywords

Sensor Network, Simulation, Debugging

1 Introduction

Technological advances in CMOS integrated circuits, low power wireless and microelectromechanical systems make it possible to embed tiny, digitally controlled sensor devices that include computation and communication capabilities into the environment. These non-intrusive computers aggregate into wireless sensor networks (WSNs) that can be programmed to work in concert to monitor phenomena in settings where larger, more functional machines would perturb the environment under study. However, the technological characteristics of wireless sensor networks, including limited processing and energy resources and unreliable and fluctuating network performance, raise extensive distributed systems software development challenges, especially for application debugging and profiling. In particular, using *in situ* physical deployments for application development is costly, labor intensive, and may be infeasible in some settings.

Because of these difficulties, simulation is widely adopted as a practical method to study sensor network behavior and verify application designs. As a consequence, numerous simulation tools [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13] have been developed for sensor networks which share many advantages over the use of physical deployments. They are low cost, produce repeatable results, are controllable and flexible, and provide meaningful insights into the behavior of the system in study from both broad and detailed perspectives. Using simulation to eliminate algorithmic errors in applications before deployment to a physical network has resulted in significant labor and cost savings. However, to date, simulation has not been able to completely supplant the use of physical hardware. While it cuts the development time necessary to achieve functionality, sensor network engineers must still conduct significant debugging and verification activities using physical devices.

One of the primary impediments to the greater success of simulation as a software engineering tool stems from the lack of an accurate model for ensemble behavior. High-level functional simulators [1, 2, 3, 10, 11, 12] model the process of radio communication in terms of the interactions of events. As a result, they achieve excellent simulation performance but normally do not provide accurate timing information which is critical for debugging and power optimization.

Full-system simulators [4, 5] are able to emulate sensor device hardware at the machine cycle level so individual device timings are correct, but they lack an accurate, yet scalable method of modeling the timings associated with radio communication making simulation of a sizable network of sensors problematic.

One approach to overcoming this shortfall is to combine simulated sensor devices with "live" physical devices to improve simulation fidelity. Many simulation systems and testbeds [2, 13, 6, 7, 8, 14] support "hybrid simulation", a method that uses real sensor hardware to perform radio communication which is initiated by and delivered to simulated sensor devices. In this mode of operation, actual sensor communication hardware replaces the radio and sensing models in the simulator. Radio traffic and sensor data are routed in between the simulator and the sensor device which is then responsible for implementing the communication and sensing and then passing the results back to the simulation. In this way, hybrid simulation achieves higher fidelity in radio transmission and environment sensing than pure simulation. Although this method of enhancing fidelity is interesting and useful, it does not enable investigations at scale (a potential benefit of simulation) since it requires the same amount of hardware as in a physical deployment.

Thus we propose a new scheme of hybrid simulation. Instead of using real hardware to enhance simulations, we propose to use high-fidelity simulations as a partner to a physical hardware deployment. To distinguish from the conventional hybrid simulation, we borrow a term from computer graphics area [15] and call it *simulation-based augmented reality* for sensor networks. Intuitively, we want to create a sensor network deployment that is partially physical and partially virtual (i.e. simulated). A key requirement, however, is that the delineation between virtual and physical devices, communication, etc. is completely transparent to applications that are running in an augmented sensor network. In such a deployment, reality (real hardware network) is "augmented" with simulation from various perspectives:

- Simulated devices can be used to extend the scale of the network under study. It is possible to study a larger networks by running critical parts of the application on real hardware and using simulated augmentation to generate meaningful test inputs, control the boundaries, etc.
- Simulated devices can be used to inject traffic or profile application execution seamlessly throughout a deployment without application or operating system modification, and these functions can be moved between network components easily. Since the application and/or OS cannot determine which sensors are real and which are virtual from the communication topology or communication method, it is possible to move an application "probe" through the sensor network by changing the sensors that being simulated.
- Augmented networks provide a progressive transition from simulation to deployment during development. By incrementally changing the ratio between physical nodes and virtual nodes and selectively picking the configuration, the testing environment makes a smooth transition from high flexibility to high fidelity, which best reflects the shifting of requirements at different development stages.

To achieve these goals, the simulated devices must be able to be interchanged with physical ones without modification to the system under study (i.e. it must support transparent execution of application binaries). In addition, the simulation of an ensemble of devices (and not just the individuals) must be high fidelity, and it must scale. In previous work [9] we developed a scalable, high-fidelity ensemble simulator designed for execution on distributed-memory clusters to achieve performance. The simulator also supports a non-intrusive, multi-level debugging facility [16] as a fully integrated system component. We use these tools to develop an infrastructure for the simulation-based augmented reality system.

To achieve seamless interaction, there must be a way to deliver radio traffic from/to the real sensor devices with minimal performance impact in the form of introduced overhead. Although it is possible to extend existing hybrid simulation facilities to support such radio traffic forwarding. Their coarse (packet-level) forwarding granularity and lack of precise time synchronization make it difficult to obtain uniformly accurate timing across the complete augmented network. Under our approach, we couple a subset of the physical devices with simulated devices into "super nodes" that stream data between simulation and live networks at the byte level. Compared to previous hybrid simulation work, our design and implementation of proxy-based traffic forwarding achieve higher timing precision for the purpose of seamless interaction. Note that our intention is to intermingle simulated devices that use a radio simulation to communicate with physical devices that communicate using the physical radios. Super nodes only facilitate the bridging of communication between the two environments and in particular, are not required for all simulated sensors.

Note also that although our system focuses on using simulation to enhance and augment development on physical network deployment, it can also benefit the typical hybrid simulation methodology. That is, the communication forwarding functionality implemented by super nodes can be used in "traditional" hybrid simulations for better results. Furthermore, in an augmented network, physical deployment can be considered as a realistic radio traffic environment to generate useful test input for more realistic simulation.

In summary, we developed a system that fuses simulation with real sensor networks in a seamless and transparent manner, such that the scalability, flexibility and controllability of simulation can be used to facilitate easier and better development. Our system is based on our previous works [9, 16], a scalable distributed sensor network simulator. The contribution of this paper is the extension of the base simulator to support the proposed augmented reality of sensor networks, which includes, most importantly, the design and implementation of a "super node" for bridging simulated network and physical network, and relevant improvements to the base simulator on radio models and randomness models. We evaluate the performance of our system and discuss various usage anecdotes in this paper.

In the rest of the paper, we first briefly introduce our distributed sensor simulation system as the foundation of our work in Section 2. Then we present the design for interconnecting simulation with real sensor network in Section 3. We evaluate our system in Section 4 and discuss application scenarios in Section 5. In Section 7, we discuss related work and in Section 8 we conclude the paper.

2 Foundation: An Accurate and Scalable Sensor Network Simulator

In an augmented simulation, virtual sensor nodes are used to extend and enhance a physical, real sensor network. Applications executed in this augmented network see no difference between the virtual part and the real part, in terms of the hardware they can access, the timing characteristics of their execution and the information they get from others. To create such a seamless, transparent execution environment, the simulation system to be used must be accurate enough to provide a convincing imitation of real sensor devices, and fast enough to interact with real sensors in real time. Another critical requirement of the simulation system is scalability. The quantity of available sensor network hardware is inevitably limited. Thus the ability to scale beyond what is convenient to deploy is one advantage that makes simulations attractive and useful to complement a physical deployment.

We developed a parallel distributed sensor network simulator for execution on clusters that is not only accurate at cycle level, but also fast and scalable [9]. We believe such simulation system lays a solid foundation for the augmented sensor reality. In this section, we briefly introduce the relevant design aspects of our distributed simulation system and explain how the features are important for an augmented network.

2.1 Accurate Full-System Simulation

The core of our simulator is a full-system hardware simulation component with extensive support for various popular sensor network devices, including mote [17] devices (Mica2 and MicaZ), Stargate device and iPAQ devices. At the current stage, only mote simulation can achieve faster than real time speed and thus can be used to augment real sensor devices. We only discuss the simulation of Mica2 device throughout the remaining paper.

Specifically, we model the core components of a Mica2 sensor device, including the AVR instruction set, the ATmega128L microcontroller with most on-chip functions, the on-board flash, the Serial ID chip, the CC1000 radio chip, the LEDs and the sensor boards. These features are enough to support the transparent execution (i.e. running an unmodified binary) of most sensor network applications, including complex ones like Deluge, TinyDB and Surge, and all TinyOS [18] components.

The hardware simulation is driven by an AVR instruction interpreter. On a typical desktop PC (Intel Xeon 2.8GHz), the simulated processor runs approximately 10 times faster than real time speed. A virtual clock is maintained to record the progress of execution. The interpreter advances the clock cycle-by-cycle according to the cycle count of each instruction. Asynchronous hardware events are registered and issued through an event queue. Event timings are obtained either from hardware specifications, which are typical values, or gathered with hardware benchmarking. Overall, this hardware simulation can achieve cycle-accuracy. We detail this accuracy more completely in Section 4.

2.2 Scalable Parallel Distributed Simulation

Cycle-accurate full-system hardware simulation is complex and computationally expensive. Moreover, if an ensemble of sensors is to be simulated, the virtual clocks of the individual devices must be synchronized [5]. Previous systems [4] rely on shared memory and lock-step clock synchronization between separate device emulations. As a result, they must execute on shared-memory multi-processors (SMPs) to achieve scale. Both the expense of these SMP systems and ultimately the limit they place on the size of the simulations they support have prompted us to develop a distributed parallel environment suitable for cluster implementation.

To do so, we have developed a message-based clock synchronization protocol that is low-latency enough to work on tightly coupled but distributed memory machines (e.g. clusters.) The execution model of our parallel simulation can be described as follows. Each sensor node is simulated in an independent operating system thread to exploit the hardware parallelism. Virtual clock synchronization happens only during communications. Two communicating sensor nodes build a causal relationship and thus have to exchange clock values to ensure the correct order of transmitted messages.

To model the radio communication, each sensor node maintains a radio media structure that buffers radio data. Two operations, *Write* and *Read*, can be performed on the radio media. The neighbors of the sensor node, i.e. nodes within its maximal transmission range, can *Write* (send) radio data to the radio media. The written radio data is time-stamped and buffered. Sensor node itself *Reads* (receives or senses channel) from the buffer to retrieve received radio data. With this model, the radio sender is free to send anytime since all the sent data is time-stamped and buffered. The synchronization burden falls on the receiver. The radio receiver has to maintain synchronous execution with potential senders and assemble correct radio data from the radio media structure. We thus design the following synchronization protocol for distributed simulation:

1. A node that reads must wait for all its neighbors to catch up with its current clock time to make sure it will receive all the data it potentially should receive;
2. All nodes must periodically broadcast their clock updates to neighbors to notify others of their progresses;
3. Before any wait, a node must first send its clock update to avoid loop waiting;
4. Data byte is always sent with a clock update at the end of its last bit transmission to avoid broken bytes.

Rule 1 is the key rule to achieve synchronization. Rule 2 and rule 3 function similar to the “null message” in parallel distributed discrete event simulation (PDES) to avoid loop waiting. Rule 4 is necessary because we transmit radio data in byte units.

The above protocol captures the essential synchronization relationship in sensor communications and has the ability to scale in the distributed environment. However, according to our analysis [9], the realization of this ability depends critically on a good load balancing scheme. Since the simulation of sensor nodes has to be distributed among multiple machines and the inter-host synchronization overhead and messaging results in actual inter-machine communication, load balancing a given simulation can be converted into a classic graph partition problem [19, 20, 21] for which many good solutions exist. We use a state-of-the-art partitioning package, Chaco [22], to automatically assign simulated sensor devices to machine nodes in a cluster so as to minimize the inter-process communication overhead.

2.3 Fidelity-Enhancing Pluggable Models

In a perfect augmented reality system, virtual entities are indistinguishable from corporeal ones not only in terms of function, but also in terms of performance profile. Thus radio transmission lossiness and delays, power consumption, etc. should all be indistinguishable in simulation and *in situ*. At present, achieving this "Turing Test" level of faithfulness remains a challenge for several aspects of sensor interaction, most especially radio communication. To allow for the possibility of incorporating new models, the simulation framework supports pluggable extensions. New radio, power, and sensing models can be added without reengineering the overall system. The current implementation also includes basic models described elsewhere in the literature [23, 24, 25, 26].

2.3.1 Radio Model

Our basic radio model is similar to that used by TOSSIM [1] and Avrora [4]. Radio packets are transmitted in a lossy way according to a reception rate specified for each link. The conflict radio transmission data is bit *OR*ed so that the "hidden terminal" effect [23] can be simulated.

The links' channel loss parameters can be specified manually by developers. A more interesting approach that is also supported is the ability to derive a statistical channel loss model from measurement data of physical deployment [24, 25, 26]. In this approach, a large set of radio transmission data is collected using different parameters. The trace data is then "mined" using statistical methods to derive distributional descriptions of characteristics, such as reception rate.

We find that this approach is especially applicable in our system. In an augmented system, virtual sensor nodes coexist with real ones. The distribution of channel characteristics can be estimated using real sensor nodes and applied to virtual nodes. In this way, the virtual portion of the network resembles the physical network in terms of its summary statistics (e.g. mean, quantiles, variance, etc.) We term this method *self-reflected feedback*. Our experience with the use of empirically gathered statistical samples to build a radio model is that the resulting ensemble simulations are more accurate than when fixed parameterizations are used.

The process of self-reflected feedback can be performed offline or online. In the offline version, reception rates are collected on the physical part of the network and then embedded into simulation. Alternatively, this sampling and model fitting can also occur online. In this approach first instrument the applications running on real sensor nodes to perform benchmarking in the background. The reception rates are collected at the virtual/real boundary by super nodes and stored in a database. The live distribution is then computed from the database to dynamically adjust the reception rate, loss rate, etc. for the virtual sensor nodes. The online method can adapt to the temporal variations in the network, and thus is more realistic. In this work, we report results only for the offline method as the online system is not yet fully operational.

2.3.2 Randomness Model

We observe that non-determinism (which we model as randomness) is important to high-fidelity simulations. For the purposes of statistically accurate simulations we see two important types of randomness on real sensor hardware. The first is time randomness, including the non-deterministic system start and time drift due to the imperfection in the crystals used to implement the clock signals on each device. The second is radio randomness, including the background interference and temporal variation of radio transmission.

We currently include two randomness modeling features in the simulation. The first is to set up random initial clock value for each device simulation thread to prevent interlocking of simulation. The second is to model the background interference in a simple way. We observed that in a sensor network, even when there is no active node, the radio of a transmitting node will still back off after checking the channel with RSSI. We thus implement an non-deterministic RSSI operation in the simulation to emulate this behavior. We are considering the time drift and temporal variation of radio transmission in future work.

2.3.3 Power and Battery Model

A number of power models for sensor network devices have been proposed and investigated in the literature [27, 28, 29]. These models are typically based on the measurements obtained by using benchmarks to exercise the sensor device in various modes yielding different levels of fidelity. In this work, we incorporate one such model [27] in our simulator.

We also provide a simple linear battery model. Several battery models have been proposed in the literature[30, 31, 32]. The linear model is the simplest, again representing the ideal case in debugging and "back-of-the-envelope" settings. Moreover, in fast, lower-fidelity simulations of "steady-state", a linear model is often preferred since the middle of the discharge curve is often close to linear [33].

2.4 Debugging and Profiling Facilities

One of the main motivations for the augmented reality system is to support online tracing and profiling for debugging sensor network applications. Our simulator provides several such useful facilities [16]. The first feature we have introduced is virtual debugging hardware. We designate three virtual registers, allocated in reserved register space of ATmega128L microcontroller to build a bi-directional communication channel between simulation and simulated program. These three registers can be used by applications to output debugging or profiling information. Since the access to the debugging registers is just one load/store instruction, little overhead is introduced to the simulation, and thus is barely intrusive. Also, since accessing the debugging registers when the code is running on a physical device generates "noops" in the instruction stream the transparency of our system is still maintained.

We also build profiling mechanisms into the simulator, which we call *debugging points*. Debugging points are the access points to the internal states of the simulation, including program status, important hardware status and synthetic hardware events. Debugging points can be used to profile the execution of sensor applications.

Another important feature of our simulator is the support of checkpoints. Checkpoints are used in our simulator to support "time traveling" [34, 35] with which we can quickly return to a history point in a trace and replay the simulation. In an augmented reality setting, with the existence of real hardware, it is harder to achieve time traveling since physical process is irreversible. However, by recording all the incoming radio traffic at the virtual/real boundary for the virtual portion of the network, we are still able to replay the execution of virtual part, which is possibly useful in some situations.

3 Super Nodes: Bridging Reality and Simulation

Similar to hybrid simulation, we attach a real sensor device to a simulated device via a serial interface, to bridge traffic between virtual and real networks. These *super nodes* exploit the much faster than real time performance of the device simulation to stream bytes between the simulation and live network while introducing minimal overhead. Figure 1

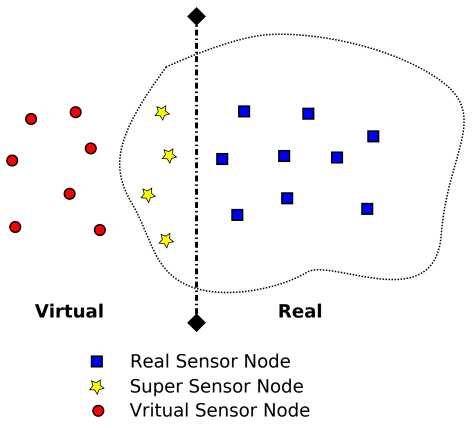

Figure 1. Augmented network

illustrates three types of sensor nodes in an augmented network. The squares represent real sensor nodes, and circles represent virtual nodes in simulation. The super nodes consist of an artificially fast virtual node connected to a physical node via a serial interface. It can still behave as a real-time virtual node by delaying the updates to its virtual clock, but it passes data at the maximum transfer rate possible. The assignment of super nodes is determined by the aggregate radio range of real sensor nodes. All the virtual nodes within that range need to be equipped as super nodes since they all have the possibility to communicate with real sensor nodes. Note that although the graph shows a clear division between the virtual nodes and physical nodes, in reality, the network topology is not restricted to such patterns. Virtual nodes can be deployed anywhere in the space as long as any of them that can communicate with real sensor nodes is equipped as a super node.

There are two problems in the design of a super node. First, the virtual world time and real world time have to be synchronized to ensure the correct radio communication between them. This problem has been addressed in most hybrid simulation systems [2, 14, 6]. However, since these previous systems are based on high-level functional simulation, which is not able to advance virtual clock precisely according to program execution, it is difficult to achieve precision. Some systems [2] do not synchronize at all and leave it for future work. The others [13, 14, 6] use coarse time estimation in the event simulation. Since we run virtual node on cycle-accurate simulator, the program performance is accurately timed. It becomes possible to precisely synchronize virtual time with real wall clock time.

The second problem is the accuracy of radio reception via the radio proxy device. In the perfect scenario, we would want to build sensor radio (e.g. CC1000 radio or Zigbee radio) directly into the simulation host (e.g. by attaching communication peripherals to the cluster nodes running the simulation). Then all the radio functionality could be controlled directly from within the simulation without the use of an intermediate sensor device to act as a proxy. The cost of these peripherals, however, and their correct programming substantially increases both the cost and complexity associated with the simulation environment. We thus use a real sensor device attached via serial line to the simulation host, which is feasible and flexible. The problem is that there is a non-negligible delay introduced by sending radio data through the serial line (Mica2 serial speed is 56KB/second, and radio speed is at 19KB/second – only a factor of three slower). Also, since we access the radio function indirectly through the physical proxy sensor device, it is impossible to capture every bit actually transmitted. Despite these limitations, however, we find that a careful engineering of the super nodes can minimize their overall impact.

Figure 2 shows the structure of the super node (right) and compares it with a normal virtual node (left). A virtual node uses a radio node component to synchronize and communicate with other nodes. A super node needs to talk to both virtual nodes and real sensors. It thus supports two radio interfaces: a virtual interface (i.e. radio node) and a physical interface. The physical interface has "incarnations" in both physical space and virtual space, which are connected via serial line. In physical space, a real sensor device is used to actually transmit the radio data, which is controlled by a TinyOS application, *Cyborg*, using an instrumented version of the MAC layer. In virtual space, a thread, called *shadow*, interacts with super node's radio simulation and synchronizes the virtual time with wall clock time (i.e. real world time).

The process of radio communication for a super node is described as follows. Since the simulation models radio

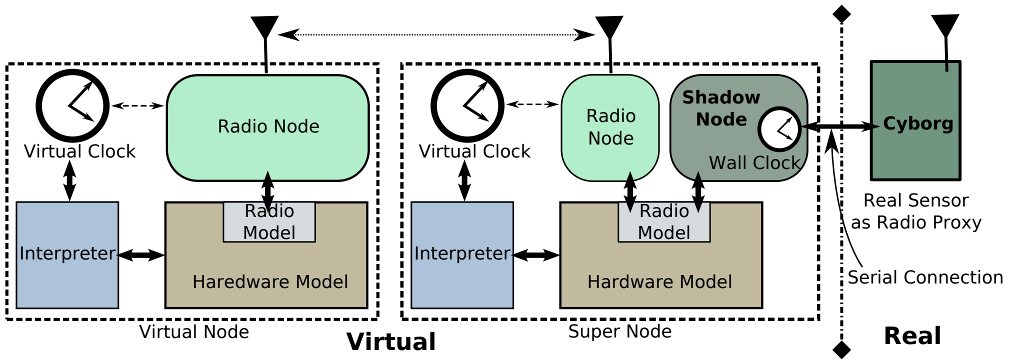

Figure 2. **Comparison of a virtual node and a super node**

communication at the byte level, we want to forward radio traffic from real space at the same granularity. We thus modified the MAC layer of TinyOS radio protocol stack to intercept every received byte so that it can be forwarded via serial the port to *shadow* as soon as possible. *Shadow* then time-stamps the received bytes and buffers them in the super node's radio media data structure as we described in Section 2. Note that in this the algorithm forwards not only successfully received packets but also corrupted packets. This functionality is desirable because the super node's radio environment is half-virtual and half-real and it needs every radio byte in both spaces (corrupt and otherwise) to model correctly radio function, especially collision detection. However, the system still misses some radio signals that escape the packet recognition process in the MAC layer. Currently, there is no direct way in the radio hardware to enable the capture of every radio bit transmitted within range. If such a tap were made available in future devices, we believe we could exploit it to improve the accuracy of augmented simulations accordingly.

Another issue is the delay of forwarding. Theoretically, a radio byte is forwarded to the radio simulation of super node after slightly more than $1/3$ of a byte time (140 microseconds or 1024 mote CPU cycles). This delay can be reduced by increasing serial speed of the mote hardware design in the future.

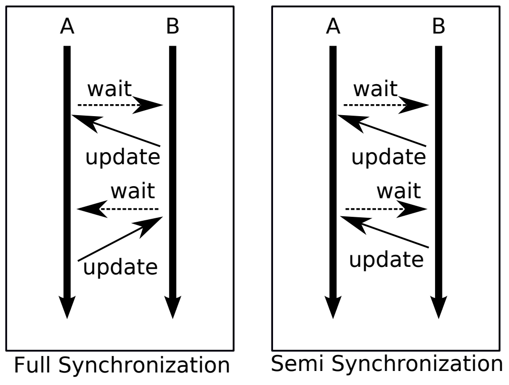

Figure 3. **Full synchronization and semi synchronization**

The super node's virtual time has to be synchronized with real time to correctly model radio reception. We show that this synchronization can be elegantly implemented within our distributed synchronization framework as we described in Section 2. In the simulation, every virtual node waits before receiving until all its neighbors pass its current time to ensure correct radio reception, and it sends clock updates to avoid loop waiting. On a super node, *shadow* acts like a normal radio node and adds itself to the super node's radio neighbor list. The super node then will wait for *shadow* every time it receives. We call this as *semi synchronization* compared to the *full synchronization* for pure simulation, as shown in Figure 3, since *shadow*, which represents wall clock time, will never wait for the simulation.

Shadow works in two ways. When there is a byte received from *Cyborg*, it sends the byte along with a clock update to the super node simulation. The wall clock time is obtained using the function *gettimeofday()*, which has microsecond precision, and converted into virtual clock time (in mote CPU cycles). When there is no incoming data, *shadow* periodically (every 1000 microseconds, the finest sleep precision possible) updates its wall clock time. One problem is that since the simulation host is a time sharing system, two contiguous bytes in a packet may arrive at *shadow* a few milliseconds apart, which will cause a corrupted reception in simulation if we depend on the actual arrival time to timestamp radio bytes. We solve this problem by parsing the format of the radio byte stream. We mark the first byte of a block of a continuous byte stream (normally a radio packet) and use its actual arrival time to update the clock. Then for every following byte, we update the clock with a theoretical byte transmission time increment (3072 mote CPU cycles). And we update the clock again using actual arrival time for the last byte in the stream.

To transmit, we use an optimistic design to send out data as fast as possible. Whenever the super node starts transmitting, a marker is sent to *Cyborg* to initiate the radio transmission on the sensor proxy. *Cyborg* passes a dummy message to the MAC layer. The following bytes are then patched into the dummy message when they arrive later. As long as the simulation is running equal to or faster than the real time speed (note that in transmitting mode, the super node does not "read" the radio and thus will not be throttled by the inter-node virtual clock synchronization protocol at this stage), given that serial speed is 3 times of radio speed, the dummy message can always be patched in time for transmission.

At first, it may seem that it is also necessary to synchronize virtual time with wall clock time before transmitting over the radio. However, before each radio transmission, an RSSI (i.e. channel sensing) is performed to avoid collisions. This is an implicit radio media read access and thus the time is already synchronized. After the transmitting is started, the pace is then controlled by the real hardware.

Both radio receiving and transmitting rely on the prerequisite that the simulation speed must be faster than real time speed. It may happen occasionally that there are fluctuations in the CPU load on a simulation host that cause transient slowdowns. For radio receiving, since all the received radio bytes are buffered in the radio media data structure, the fluctuation will be absorbed. For radio transmitting, since the real sensor device as the proxy will not wait for simulation once it starts, a packet may be corrupted. According to our experience, this almost never happens in a dedicated simulation environment but if the simulation shares processors with other programs, it is a possibility.

4 Evaluation

We begin this section with an illustration of the simulator's performance and scalability. The purpose of the inclusion of these results (of which a more complete accounting is provided in [9]) is to demonstrate that the simulator's performance is high enough to implement a virtual sensor network that can interact seamlessly with a "live" deployment. However, we wish to be unambiguous about the novelty of this initial justification since the simulator (without super nodes and the support need for them) is prior art. It is the augmentation of a physical deployment that the extensions to the simulator make possible which constitutes the novel contributions of this paper. In the second subsection, we evaluate several example augmented networks to substantiate these contributions.

4.1 Accuracy and Scalability of Simulation

Benchmark	Measurement	Simulation	Error
cpu	$3.498E+06$	$3.499E+06$	0.02%
flash_read	$2.884E+06$	$2.954E+06$	2.44%
flash_write	$2.521E+03$	$2.434E+03$	3.44%
radio	$4.672E+05$	$4.862E+05$	4.06%

Table 1. Average timing for timing benchmarks. First column shows the benchmarks. Second column shows the measured CPU cycles on real motes. Third column shows the simulated cycles. The last column shows error percentage.

We use four benchmarks to test the cycle-accuracy that together exercise the most important subsystems on the mote device. The *cpu* benchmark runs CPU intensive computations, *flash_read* performs small reads from the on-board flash chip, *flash_write* writes to the on-board flash. *radio* exercises the CC1000 radio chip and transfers a small amount of data, under nearly perfect conditions (i.e. 100% reception rate). We use an oscilloscope to measure the benchmark execution time on physical device and convert it into mote CPU cycles.

The result is shown in table 1. All cycle counts are the average of 20 measurements. The last column of the table shows the error as the difference between the observed average and the simulated one. We see that the *cpu* benchmark generates the lowest error and that radio operation (*radio*) generates the largest error. In general, simulation is fairly close to physical measurements.

Nodes	Total Number of Hosts				
per host	1	2	4	8	16
1	9.28	2.26	1.96	1.72	1.67
2	6.68	2.12	1.82	1.68	1.68
4	2.18	1.83	1.70	1.68	1.67
8	1.20	1.21	1.18	1.16	1.15
16	0.78	0.61	0.60	0.60	0.60
32	0.35	0.36	0.31	0.31	0.31
64	0.18	0.15	0.17	0.15	0.14
128	0.09	0.09	0.09	0.08	0.08

Table 2. Simulated clock speed for 1-D topology. Each row has fixed number of nodes per host and each column has fixed number of hosts. All values are normalized to real time clock speed.

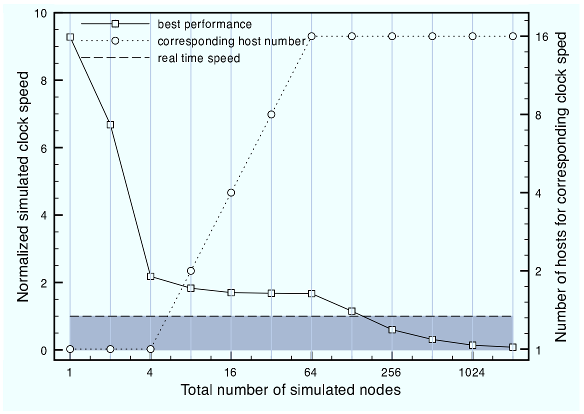

Figure 4. Gold curves for 1-D topology. X-axis is total number of nodes simulated. The left Y-axis is normalized simulated clock speed and the right Y-axis labels the number hosts. The decreasing curve is the fastest speed curve generated by choosing the best node configuration at each host count. The increasing curve gives the corresponding host count for the decreasing curve at each point. The dashed horizontal line shows the real time clock speed (7372800 cycles/second)

We evaluate scalability of simulation on a cluster consisting of 16 dual-processor 3.2GHz Intel Xeon machines interconnected by switched gigabit Ethernet. For all the scalability experiments, we use example application *CntToRfm*, in TinyOS version 1.1.15, as the benchmark, as it has been used extensively in previous scalability studies [1, 4] (offering greater potential for comparison). *CntToRfm* periodically broadcasts the value of an integer counter to keep the radio busy.

We perform scalability experiments using various network topologies. In this paper, we only show the result for the simplest case: a linear topology where all nodes are laid on a straight line, 50 meters apart (maximal radio range is 60 meters). Other results are elaborated in [9].

Table 2 presents the result. Each cell of the table shows the ratio of the simulated average clock speed to the real time clock speed, of 7372800 cycles per second. To compute the average simulated clock speed, the simulator records the number of clock cycles each mote executed during a 60 second execution run. The sum of the cycles is divided by the number of motes, and that number is divided by 60. Thus each cell depicts the average slowdown or speedup factor relative to native execution speed.

From the table, the best performance is a speedup of **9.28** times real time speed when simulating one node on one host (the upper lefthand corner in the table). When the simulation is distributed to multiple hosts, the simulation speed drops due to the overhead of distributed synchronization. However, for a specific number of nodes per host, simulation speed does not decay significantly with increasing total number of hosts. The overall result is that nearly **160** nodes can be simulated in real time speed using 16 hosts, and improvement of almost a factor of 5 over comparable previous TOSSIM results [1].

In Figure 4, we plot the best performance obtained from Table 2 for each node count. Note that to compute the node count, the nodes per host value is multiplied by the number of hosts. The units of the *y*-axis on the lefthand side of the graph are the same as in the table. For each point, we also plot the corresponding host number at which the best performance is achieved (the host count is shown on the *y*-axis at the righthand side of the graph). We call the two curves "gold curves" since they show the number of hosts necessary to obtain the fastest simulation of a specific number of nodes. Note that the second fall off in the best performance curve occurs when the number of hosts reaches 16 (the maximum number in the cluster) and the total node count is increased beyond 64.

4.2 Evaluation of an Example Network

In this section, we evaluate the cycle precision for an augmented network in which simulated and physical networks are co-mingled. The experimental apparatus we have implemented for this evaluation is shown in Figure 5. For the physical deployment part of the network, we use Mica2 motes with 433MHz CC1000 radio. In order to build a multihop network with less-than-perfect loss characteristics in limited lab space, however, we use shorter antennae and reduce the transmit power to the minimal level: -20dbm. Although this "desktop" version of sensor network deployment is not realistic in terms of emulating wireless communication, it is practical enough for our purpose since we focus on evaluating the accuracy of traffic bridging between the virtual network and live, physical sensors. For super nodes, we use MIB510 boards attached with motes connected to a PC equipped with PCI extended serial ports, and MIB520 boards connected to the PC's USB ports. As an illustration of the requirements imposed by byte-level accuracy, note that the usual MIB510 with USB to serial adaptor can not be used since the internal buffer of the USB to serial adaptor disrupts the scheduling of individual byte transmissions within the super nodes. For physical portability and to minimize the need for machine-room access, we use a cluster consisting of only two computers for the simulation, A 3.2GHz Intel Pentium D dual-core desktop PC is used as an "interface machine", which is connected to "mote proxy" devices, and runs the simulation of the super nodes. A 2.8GHz 4-AMD Opteron dual-core processor server is used to run the rest of the simulation. The two computers are connected via the campus LAN network (100Mbps) and are located in separate rooms (offices in our department). The interface machine, however, is located in the same room as the physical sensor network devices. Unless specified, we use a benchmark TinyOS program based on a multihop routing algorithm, which is adapted from MintRoute, a standard multihop routing algorithm in TinyOS. This benchmark program can ping a node in the network and query the status of the node, including its parent in routing tree, etc. We use the same binary of the program, without modification, for both real sensors and in simulation.

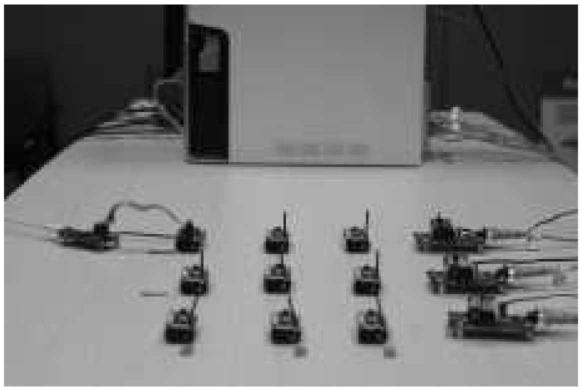
Figure 5. **Experimental apparatus**

4.2.1 *Super Node Overhead*

We first evaluate the latency of network traffic forwarded through a super node. As we describe in section 3, super nodes forward traffic at the byte level. We build a 1-hop network with two physical sensors and a corresponding 1-hop augmented network with one physical sensor and one super node. The topology is shown is figure 6. We ping node #1 via node #0 and measure the round trip time (RTT). We collected 100 samples of RTT for both physical and augmented deployments. The results are shown in figure 7. The top graph shows the RTT histogram for the physical deployment and the bottom is for augmented deployment. The distributions are similar in shape and the augmented case is right slightly, indicating the additional overhead introduced by super nodes (sample means and standard deviations are given in the figure). On average, the super node has a 4.5% (about 3 milliseconds) overhead in RTT.

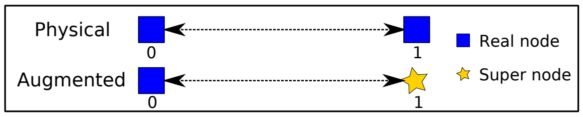

Figure 6. **1-hop network test. The top is the physical deployment and the bottom is the augmented deployment. Squares represent real nodes and stars represent super nodes.**

4.2.2 *Multihop Round Trip Time*

We measure RTT in a multihop network to further evaluate the accuracy of time synchronization in an augmented setting. To minimize the influence of the radio model, we use a simple network topology in which sensors oriented linearly. We first use real sensors to measure the RTT from

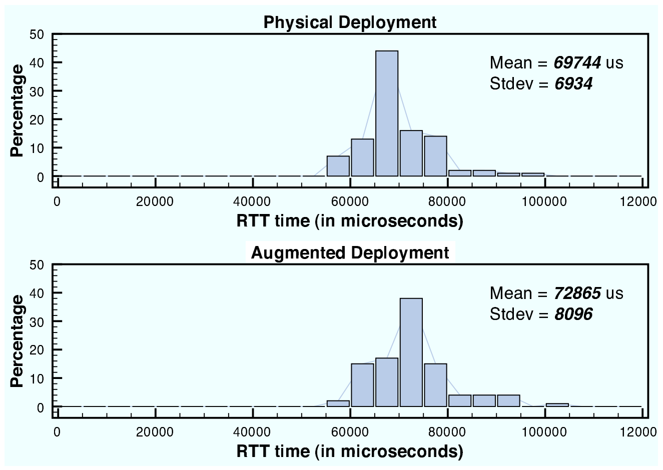

Figure 7. **Round trip time (RTT) histograms in a 1-hop network.**

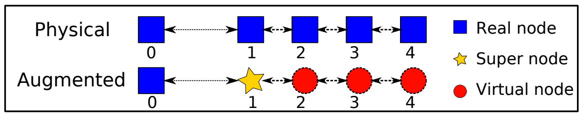

Figure 8. **A multi-hop network. The top is physical deployment and the bottom is augmented deployment.**

node #0 to node #1, #2, #3 and #4. We adjust the distances to make sure that the routing tree is linear and the ping actually takes 1, 2, 3 and 4 hops. Then we replace node #1 to #4 with simulations (#1 as a super node) and repeat the same experiment. Note that we use measured reception rates in the physical deployment to model the simulated radio.

The result is shown in figure 9. For one and two hops, the RTT in the augmented deployment is slightly larger than in the physical deployment. This delay is due to the latency of traffic forwarding on a super node. For three and four hops, the RTT of augmented deployment is slightly lower. We believe this effect is due to the simplicity of radio model and its inability to capture various physical phenomena that influence radio transmission. For example, background RF interference, which may cause a radio to back off even when there is no other active radio, is not captured in the empirical loss distributions we use for the simulated radio model. Indeed, radio model development is an active and important area of research and part of the motivation for developing the augmented framework is so that models can be developed more effectively. In this simple example, however, the two RTT curves are close (although statistically different – note the error bars in the figure). We believe this level of accuracy is both new, and sufficient to support more detailed development.

4.2.3 *Simulated Clock Speed*

To further illustrate the degree to which the simulation and physical networks are synchronized, we show the simulated clock speeds of the node #1 to node #4 in the above multihop RTT experiment for a 200-second execution. Each data point is an average clock speed over a 10-second period (in virtual time) in simulation. The results are shown in Figure 10. According to the device specifications, the sus-

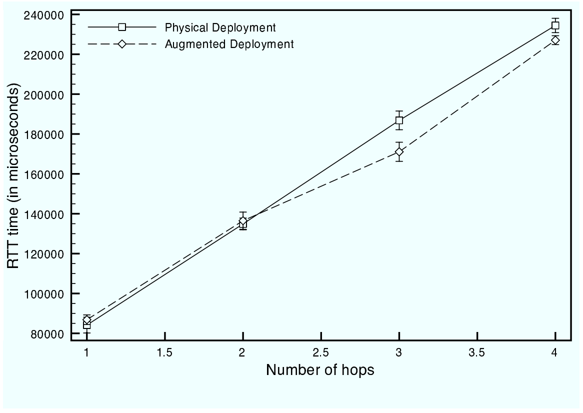

Figure 9. **Round trip time (RTT) for a multi-hop network. Each data point is an average of 30 measurements. Error bars are displayed at each data point.**

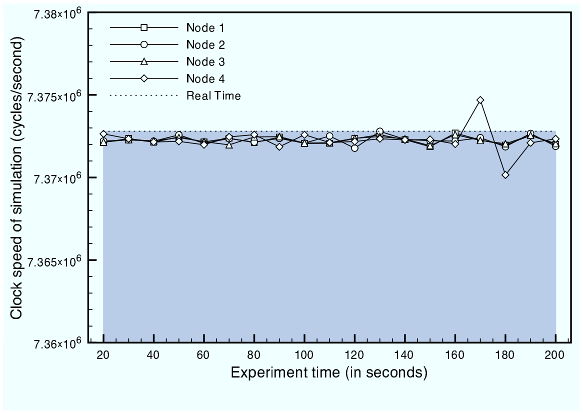

Figure 10. **Clock speed of simulation for multihop network RTT experiment. Each data point is an average clock speed of 10-second simulation (in virtual time). The real time clock speed is shown as the dashed line.**

tained cycle rate for this benchmark should be 7372800 cycles/second. The average measured clock speed of the complete simulation is 7372262 cycles/second, with only 538-cycle difference (-0.007%). In figure 10, there is a small discrepancy at the end of the curve. This discrepancy may look big in the graph, but is actually very tiny (about 2000 cycles, 0.027% in percentage). Although we are not sure about the exact cause, we believe it is due to the sudden load change of the host operating system since we were sharing our simulation machines with other research activities in the lab. We will investiage the exact cause of this discrepancy in our future work.

4.2.4 *Application Example: Audio Packet Throughput*

In this subsection, we evaluate the behavior of augmented networks using a simple application (i.e. as opposed to the micro-benchmark results presented above). The application is a simplified implementation of the core algorithm of a real-time audio transfer program. The basic algorithm uses a pre-calculated route to send radio packets carrying audio data as fast as possible from a specific source node to a specific

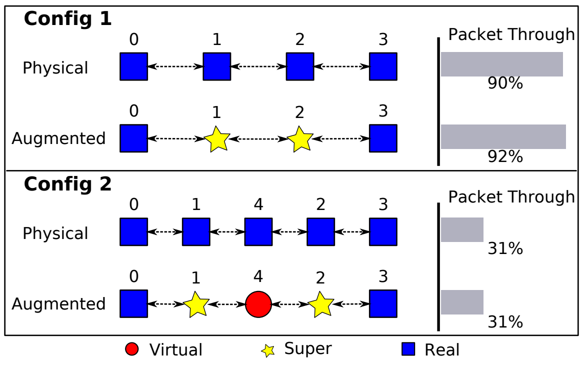

Figure 11. **Throughput test. The top is a configuration with 4 nodes in a line. The percentage of "through" packets is shown at the right for both physical and augmented deployments.**

sink. We implement this algorithm and measure the packet throughput in physical and augmented counterparts to illustrate the perceived difference between a physical deployment and an augmented deployment. Figure 11 shows the topology of the network configurations used in the experiment. In the first configuration, there are 4 nodes in a linear arrangement. For the matching augmented deployment, we replace node #1 and node #2 with two super nodes. In the second configuration, we add a simulated node #4 between the above two nodes. The radio model for simulation in the augmented deployment is again built from collected reception rate measurements of a similar physical deployment. The percentage of packets correctly transferred by the application are shown at the right of each configuration for both physical and augmented deployments. For this experiment, the packet throughput observed by the application in both cases is quite similar between the fully physical deployment and the augmented system.

5 Discussion of Application Scenarios

Hybrid simulation, proposed by a number of previous works, has not yet been fully proved its usefulness in real development. Likewise, our augmented system is still new and its true utility is not yet clear. However, through our experiences in development, we believe it enables novel capabilites and is more flexible than conventional hybrid simulations. In this section, we discuss several possible applications of augmented sensor network deployments. We frame this discussion in terms our anecdotal experiences with augmented sensor networks in different development and testing circumstances as well as more general utilities for the overall approach.

5.1 Critical Component Testing

It is often useful to stress-test a small critical part of a sensor network using large and varying input traffic under differing network conditions. For example, in applications where the data sink is a both a performance and reliability critical point, it may be advantageous to determine the load and scale levels that define its useful operational range. It is possible to stress-test a single node in isolation, but to determine the response of the system and the degree of system activity that causes failure or performance degradation re-

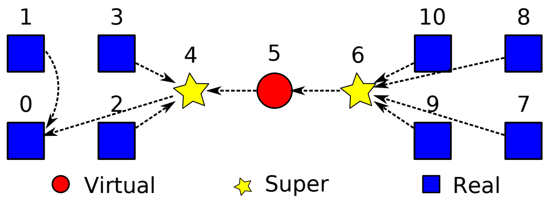

Figure 12. **Two 4×4 networks are connected via three routing nodes. The overall routing tree is displayed.**

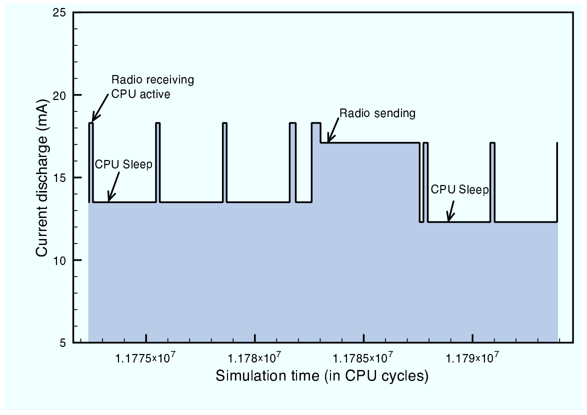

Figure 13. **Snapshot of current discharge for a short run of node #5. Note that the transmitting power is lower than receiving power due to our minimal radio power setting, which takes less power when sending than receiving.**

quires interaction with a complete network. However, scaling and reconfiguring a physical deployment is potentially tedious and labor intensive. Using an augmented system that supports the transparent substitution of physical nodes with simulated ones provides the opportunity to fine tune the introduced load in terms of a specific application, and also to observe the overall response of the ensemble sensor network to stress placed upon a critical node.

We show in Figure 12 and Figure 13 an example of how we used this approach to study the power consumption of critical sensor network nodes. In the studied network, two 4×4 networks are connected via three nodes connected in sequence for the purpose of routing between them (#4 to #6). In the augmented version we use two super nodes (#4 and #6) and a simulated node (#5) to replace the routing nodes so that we can easily instrument them as they pass traffic between the two network fields. We are then able to plot the current discharge of node #5 according to the measurements gathered during its simulation. The results are shown in Figure 13.

Another useful possibility is to study how heterogeneous devices in the network function, especially gateway level devices. As mentioned in Section 2, the simulator supports Stargate [36] devices, although not at real-time speeds. However, because they are more fully featured, debugging and instrumentation support are significantly more advanced than for motes. Thus, to test an application that uses both motes and Stargate devices, we use "live" instrumented Stargates communicating with a simulated mote network. Again, because the simulated motes can be instrumented more effectively than their physical counterparts, it is possible to determine their response more easily and with a finer level of detail using this methodology.

5.2 Cost Effective Scaling Experiments

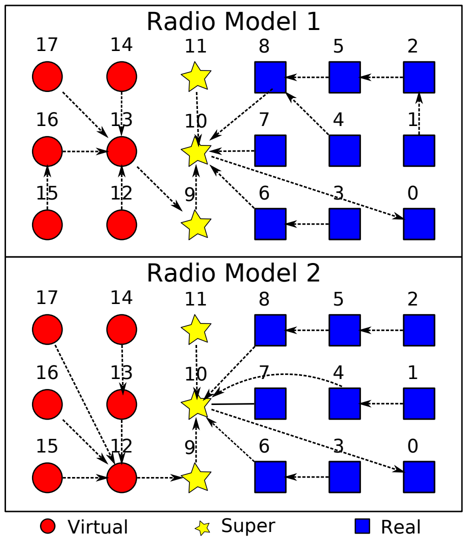

Figure 14. **Multihop routing trees for two different radio models in an augmented network.**

While sensor networks are intended, ultimately, to consist of ubiquitous low-cost devices, the currently available research platforms, particularly at scale, can be expensive. Moreover, in the early development stages of a project, it is not always cost effective to procure the maximal hardware configuration that is large enough to span all design points under consideration. As an example, consider an audio streaming sensor network. In such a network, some sensors are equipped with microphones as the sources of audio. Other sensors may act as the relays in a multihop wireless network. There are also some devices used as stream sinks. Considerable cost savings are possible if the most effective deployment topologies and network density can be investigated using simulated motes. At the same time, the fidelity of these simulations is enhanced if the application that will drive the system can be used to determine the best possible configuration. In this audio example, we can connect physical source and sink motes to different simulated routing motes to determine the effect of different configurations at different scales. In each experiment, we run the actual audio application to drive the test. In this way, we are able to thoroughly test a large size network using a severely limited hardware budget.

We illustrate this technique in Figure 14. In the figure a 3×3 physical network is connected to a 3×3 simulated one. Three of the simulated nodes (9,10, and 11) are super nodes. In the top half of the figure, we show a multi-hop route determined by a simple radio model in which each node knows its geographic neighbors and the reception rates are all 100%. In the lower half of the figure, we show the routing tree determined by the same algorithm when empirical distributions of loss and reception rates are used to model radio behavior. We are able to test the audio application without modification using all 18 nodes using both routing trees using $1/2$ the necessary hardware.

5.3 Fidelity-Progressive Development Model

By progressively increasing real nodes and decreasing virtual nodes, the fidelity of the augmented network makes smooth transition from more virtual, and thus more flexible and debuggable, to more realistic and thus more trustable in terms of observed response. As the previous anecdotes indicate, we are using a development approach based on this fidelity-progressive method. At the early stages of development, the application is run completely in simulation so that the focus can be on the software logic without non-deterministic and non-repeatable perturbations caused by noise. As components and functionality are tested, more and more of the application is moved out of simulation and into a physical deployment. At any stage where instrumentation, profiling, or debugging is required, only the relevant components needs be simulated until finally the entire application is ready for full-scale operation. This development model provides a migration path from the idealized, controllable, and repeatable environment offered by the simulator to the real world execution platform embodied in a physical deployment without making the entire jump all at once.

5.4 Augmented Testbed

As a summary, we envision a sensor network development harness implemented as a re-configurable augmented system. The harness we are considering is composed of sensor hardware (motes, Stargates, etc.) and interface machines than can be linked (via a LAN) to different clusters for simulation. All sensor devices will be connected via USB programming boards (like Xbow MIB520) to a set of interface machines. With appropriate software configuration support, any part of the network can be configured either as physical or virtual, while in the physical sensor hardware, it is only necessary to re-program the boundary nodes with the Cyborg application so that they can become super nodes. Due to the binary transparency supported by the simulator, once the application is compiled, it can be used throughout the experiment until the source of the application is changed. This testbed can easily support the fidelity-progressive development model. It can also be used for experimentation with a much larger size network than the the available hardware can support with the help of high performance simulation.

6 Limitations

The capability of our system does have its limitations under certain circumstances. In this section, we discuss these restrictive conditions.

6.1 Topology

We've seen in section 3 that in an augmented network, each "boundary" node, i.e. virtual node that can communicate with physical nodes, needs to be assigned as a super node. A physical sensor device, thus, has to be used to equip the node. For a "dense" network, in which the majority of virtual nodes are super nodes, we may need a large number of real sensor devices to build an augmented network. In this case, although simulated nodes are still useful for debugging and profiling critical nodes, an augmented network is certainly not attracting anymore in terms of using small number of sensor hardware to evaluate large scale networks.

6.2 Simulation Speed

Unlike pure simulation, the hybrid approach of our system restricts the execution speed of the complete network since the virtual nodes are always synchronized with real sensors. Thus an augmented network is not able to run at its maximal simulation speed allowed by simulation hosts. This may be good in many cases, but insufficient for experiments that require much faster than real speed performance.

6.3 Cost

Although our system enables the capability to use a small number of sensor hardware to study a large scale sensor network, the overall cost of experiment setup may be higher than using pure sensor hardware due to the requirement of distributed computing resources. This is especially true in our current design, in which a dedicated cluster is needed for optimal performance. As part of our future work, we are currently designing an automatic load balancing scheme. Given the increasing popularity of parallel computing technologies, e.g. the multi-core processors, researchers will be able to utilize the shared, heterogeneous computing resources, e.g. desktops and servers, without purchasing dedicated clusters.

6.4 Faster Radio

Our work in this paper focuses on a sensor network using low speed CC1000 radio. Faster radios, e.g. Zigbee, are becoming more popular. It is necessary to extend our system to support them. Our previous work [9] has shown that it is possible to achieve faster than real time speed performance for Zigbee systems (MicaZ). Now the problem is the speed gap between a fast radio (250Kbps) and a slow serial connection (57.6Kbps to 115.2Kbps) because our system requires real time traffic forwarding via serial link. We believe it is not a serious issue for many sensor network applications since they usually have very long duty cycles and the radio is never saturated. Also, new hardware design is increasing the speed of serial link by utilizing USB technology (e.g. the TelosB platform). Given its potential of capability, this speed gap will be bridged in the near future.

7 Related Work

We discuss related work in this section in two categories: network emulation and hybrid simulation. Network emulation [37, 38, 39, 40] is conceptually closely related to our work. Network emulation simulates the properties of a network in order to assess the network performance in the real application environment. Usually, a simulation device, which can be a software component in the network protocol stack, a general-purpose computer running simulation, or a dedicated simulation device, is introduced into the network. The network attributes, like latency, bandwidth, congestion, packet loss, re-ordering and duplication, etc., are modeled in the simulation. Network traffic from/to applications is re-shaped through the simulation so that developers can experience and investigate the effect of interested network attributes.

NIST Net [37] is a Linux-based network emulation tool. NIST Net can be considered as a specialized router which statistically emulates the behavior of an entire network in a single hop. NIST Net re-shapes the traffic passing through it by applying network effects, like packet delay, loss, duplication, reordering and bandwidth limitations, based on user settings. NIST Net is implemented on Linux as a kernel module and can be controlled by users through a set of APIs. The emulation is rule-based. A table of rule entries, which consist of specification of packets to be matched, a set of effects to be applied and a set of statistics to be logged, is used to match and re-shape network traffic passing through the emulation.

Dummynet [39] is a simple, yet flexible and accurate network simulator. Dummynet is built into an existing protocol stack, allowing experiments to be run on a standalone system. Dummynet intercepts communications between protocol layers and simulating the effects of finite queues, bandwidth limitations and communication delays. Dummynet is implemented on FreeBSD and targeted to TCP protocol stack.

Network emulation can also be performed using existing network simulator. Network emulation with NS [40] intercepts live network data and feeds them into NS network simulator [41]. The traffic is then re-shaped through the simulated network. A real-time scheduler ties event execution within the simulator to real time. The simulation speed will affect the time conversion. If the simulation host does not have enough computation power, the simulation can lag behind real time and cause undesired packet loss.

Conceptually, like our work, network emulation also creates an illusion of a real network through simulations to “augment” an existing network development environment. However, there are some fundamental differences compared to our work. Network emulation focuses on “re-shaping” the traffic while our augmented reality system may be used to “generates” traffic. Network emulation studies how the traffic is changed in a network by attributes like latency, jitter and bandwidth. The wireless sensor network, however, is much more complicated due to its peer-to-peer nature. The radio traffic in a sensor network hops from one node to another, greatly influenced by the applications running on them. In network emulation, the application layer (which is responsible for generating the traffic that traverses most networks) is not usually modeled. Also because of this, the conversion between real time and virtual time is much more complicated in our system. Our system also has broader application as a development tool. We can not only use virtual networks to connect real sensor networks and applications like network emulation tools, but also debug and profile applications within the virtual network, which provides a controllable development environment, using the physical network as realistic input.

Hybrid simulation [7, 8, 2, 13, 14, 6] is a popular methodology in simulating sensor networks with many similar aspects to our work. Hybrid simulation attaches real sensor devices to simulated sensor nodes to perform realistic radio communications or get genuine sensor data.

EmStar [7] and EmTOS [8] are testbeds for developing heterogeneous sensor network applications. The hybrid simulation is called “emulation mode” in the system. In the emulation mode, real motes are attached to the simulation system via a serial protocol, HostMote. The software service that forwards the radio traffic (sometimes also including the mote device itself) is called *MoteNIC*. MoteNIC sends radio packets to *Transceiver* running on motes – an application similar to *GenericBase* (a TinyOS packet proxy application). The purpose of emulation mode is to replace the computed radio channel model with realistic radio communications. EmTOS also has a "hybrid mode" in which hybrid nodes (i.e., simulated nodes with attached motes) coexist with real nodes. Note that this hybrid mode is different from our augmented network in that hybrid mode does not contain any simulated nodes communicating via a simulated radio model as in our system and all network traffic is routed through a real radio channel. EmTOS does not preserve precise timing or interrupt handling in simulation. It is unclear how time is synchronized in EmTOS.

MULE [13] is a simulator designed especially for hybrid simulation, based on TOSSIM [1]. The motivation for MULE is to use realistic sensor data in simulation. A mote is attached to the virtual node in TOSSIM simulation, used as both a radio proxy and a sensor data collector. The time synchronization is discussed in detail in [13]. MULE reconciles virtual time and real time by suspending simulation before a real event and resuming simulation based on collected event timing. This method makes concurrent message transmission difficult. Kansei [14] is also a sensor network testbed that uses a similar hybrid simulation method to implement a realistic radio channel.

SensorSim [2] is a simulator based on NS-2 [41] providing comprehensive hardware, software and power models and hybrid simulation is one of its most important features. Similar to other hybrid simulators, real motes are used for radio communication. SensorSim does not synchronize virtual time and real time, however. A "pause simulation" method like that developed in MULE, is considered for future work.

Many sensor network simulators, like TOSSIM [1] and Avrora [4], have a built-in serial forwarder so that they can communicate with real sensors via a proxy application, like *TOSBase* or *GenericBase* TinyOS applications. TOSSIM also has the capability to be injected real world radio traffic for simulation.

SEMU [6] is the only hybrid simulator we have seen that is built on virtual machine(QEMU [42], an open source binary emulator). SEMU also uses "pause simulation" method to achieve time synchronization. Because it is based on a virtual machine, SEMU can convert times with better precision by profiling the execution and estimating its performance.

As we have emphasized, our work has a much broader design motivation than that which has driven previous work in hybrid simulation. Our main focus is to use accurate, scalable simulation to enable a better development environment for physical network deployments. With this purpose, the simulated virtual network can be used to generate convincing traffic for physical networks as test input. Conversely, virtual networks expand the size of network that we can study given limited hardware budget. The key to the approach is the ability to bridge network traffic and synchronize virtual time and physical time at a fine level of control. However, our system can also be used to implement the same radio channel realism as previous efforts. Given cycle-accurate simulation, transparent binary execution, precise time synchronization and fine-grained byte-level radio forwarding, our system can be used to implement more accurate and versatile hybrid simulation.

8 Conclusion

In this paper, we present a new sensor network development paradigm, *simulation-based augmented reality*, in which scalable and accurate simulation is used to "augment" a physical sensor network deployment for various benefits. To enable seamless fusion of simulation and a live sensor network, thus making them transparent to each other, we base the design of augmented reality on a scalable, cycle-accurate, distributed simulator. We discuss the aspects of the simulator that helps our cause. Built upon that, we design and implement a super sensor node which talks to both simulation and live network with fine-grained radio traffic forwarding and precise time synchronization. We show the timing accuracy of our augmented network with experiments. We also demonstrate its behavior using application examples discuss usage anecdotes based on our current work.

In the future, we are interested in building more realistic radio model for augmented network using *self-reflected feedback* method, and better randomness models including time drifting and temporal variations of radio, as we described in Section 2. We also plan to explore the optimistic simulation approach [43] to enable better performance.

9 References

[1] Philip Levis, Nelson Lee, Matt Welsh, and David Culler. TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications. *ACM Conference on Embedded Networked Sensor Systems*, November 2003.

[2] Sung Park, Andreas Savvides, , and Mani B. Srivastava. SensorSim: a simulation framework for sensor networks. *ACM International workshop on Modeling, analysis and simulation of wireless and mobile systems*, pages 104–111, 2000.

[3] Sameer Sundresh, Wooyoung Kim, and Gul Agha. SENS: A Sensor, Environment and Network Simulator. *The IEEE 37th Annual Simulation Symposium*, 2004.

[4] Ben L. Titzer, Daniel K. Lee, and Jens Palsberg. Avrora: Scalable Sensor Network Simulation with Precise Timing. *The Fourth International Symposium on Information Processing in Sensor Networks*, April 2005.

[5] Jonathan Polley, Dionysys Blazakis, Jonathan McGee, Dan Rusk, and John S. Baras. ATEMU: A Fine-grained Sensor Network Simulator. *IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks*, 2004.

[6] Shih-Hsiang Lo, Jiun-Hung Ding, Sheng-Je Hung, Jin-Wei Tang, and Wei-Lun Tsai. SEMU : A Framework of Simulation Environment for Wireless Sensor Networks with co-simulation model. *In the Proceedings of International Conference on Grid and Pervasive Computing (GPC), Lecture Notes in Computer Science (LNCS)*, May 2007. France.

[7] Lewis Girod, Jeremy Elson, Alberto Cerpa, Thanos Stathopoulos, Nithya Ramanathan, and Deborah Estrin. EmStar: a Software Environment for Developing and Deploying Wireless Sensor Networks. *USENIX Technical Conference*, 2004.

[8] Lewis Girod, Thanos Stathopoulos, Nithya Ramanathan, Jeremy Elson, Deborah Estrin, Eric Osterweil, and Tom Schoellhammer. A System for Simulation, Emulation, and Deployment of Heterogeneous

Sensor Networks. *ACM Conference on Embedded Networked Sensor Systems*, November 2004.

[9] Ye Wen, Rich Wolski, and Greg Moore. DiSenS: Scalable Distributed Sensor Network Simulation. *In Proceedings of ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP 07)*, March 2007. San Jose, CA.

[10] Alexander Kroeller, Dennis Pfisterer, Carsten Buschmann, Sandor P. Fekete, and Stefan Fischer. Shawn: A new approach to simulating wireless sensor networks. *eprint arXiv:cs/0502003*, February 2005.

[11] ElMoustapha Ould-Ahmed-Vall, George F. Riley, Bonnie S. Heck, and Dheeraj Reddy. Simulation of Large-Scale Sensor Networks Using GTSNetS. *In Proceedings of the 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'05)*, 2005.

[12] J. Barbancho, F.J. Molina, C. Len, J. Ropero, and A. Barbancho. OLIMPO, An Ad-Hoc Wireless Sensor Network Simulator for Optimal SCADA-Applications. *Communication Systems and Networks (CSN 2004)*, 450, September 2004.

[13] D. Watson and M. Nesterenko. Mule: Hybrid Simulator for Testing and Debugging Wireless Sensor Networks. In *Workshop on Sensor and Actor Network Protocols and Applications*, August 2004.

[14] Ohio State University,Kansei: Sensor Testbed for At-Scale Experiments. Poster, 2nd International TinyOS Technology Exchange, Berkeley, February 2005.

[15] Rolf R. Hainich. *The End of Hardware: A Novel Approach to Augmented Reality*. BookSurge Publishing, 2006.

[16] Ye Wen and Rich Wolski. S^{2}DB: A Novel Simulation-Based Debugger for Sensor Network Applications. *In the Proceedings of 6th Annual ACM Conference on Embedded Software (EmSoft 06)*, October 2006. Seoul, South Korea.

[17] Mote hardware platform. `http://www.tinyos.net/scoop/special/hardware`.

[18] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister. System architecture directions for network sensors. *International Conference on Architectural Support for Programming Languages and Operating Systems*, October 2000.

[19] Kirk Schloegel, George Karypis, and Vipin Kumar. Graph Partitioning for High Performance Scientific Simulations. *Draft to be included in CRPC Parallel Computing Handbook, Morgan Kaufmann*, September 2000.

[20] Horst D. Simon. Partitioning of Unstructured Problems for Parallel Processing. *Computing Systems in Engineering*, 2:135–148, 1991.

[21] Alex Pothen. Graph partitioning algorithms with applications to scientific computing. *Parallel Numerical Algorithms*, pages 323–368, 1997. Kluwer.

[22] Bruce Hendrickson and Robert Leland. The Chaco User's Guide: Version 2.0. Technical Report SAND94–2692, Sandia National Lab, 1994.

[23] F. A. Tobagi and L. Kleinrock. Packet switching in radio channels: Part II-The hidden terminal problem in carrier sense multiple-access and the busy-tone solution. *IEEE Transactions on Communications*, COM-23:1417–1433, 1975.

[24] Alberto Cerpa, Jennifer L. Wong, Louane Kuang, Miodrag Potkonjak, and Deborah Estrin. Statistical Model of Lossy Links in Wireless Sensor Networks. *In the ACM/IEEE Fourth International Conference on Information Processing in Sensor Networks (IPSN'05)*, April 2005. Los Angeles, California.

[25] Gang Zhou, Tian He, Sudha Krishnamurthy, and John A. Stankovic. Impact of radio irregularity on wireless sensor networks. *In Proceedings of the 2nd international conference on Mobile systems, applications, and services (MobiSYS'04)*, 2004.

[26] Jerry Zhao and Ramesh Govindan. Understanding packet delivery performance in dense wireless sensor networks. *In Proceedings of the 1st international conference on Embedded networked sensor systems (SenSys'03)*, 2003.

[27] Olaf Landsiedel, Klaus Wehrle, and Stefan Gotz. Accurate Prediction of Power Consumption in Sensor Networks. *In Proceedings of The Second IEEE Workshop on Embedded Networked Sensors (EmNetS-II)*, May 2005. Sydney, Australia.

[28] Victor Shnayder, Mark Hempstead, Bor-rong Chen, Geoff Werner-Allen, and Matt Welsh. Simulating the Power Consumption of Large-Scale Sensor Network Applications. *In Proceedings of the Second ACM Conference on Embedded Networked Sensor Systems (SenSys'04)*, November 2004. Baltimore, MD.

[29] Victor Shnayder, Mark Hempstead, Bor-rong Chen, and Matt Welsh. PowerTOSSIM: Efficient Power Simulation for TinyOS Applications. *In Proceedings of the Second ACM Conference on Embedded Networked Sensor Systems (SenSys'04)*, November 2004. Baltimore, MD.

[30] K. C. Syracuse and W. Clark. A statistical approach to domain performance modeling for oxyhalide primary lithium batteries. *In Proceedings of Annual Battery Conference on Applications and Advances*, January 1997.

[31] L. Benini, G. Castelli, A. Macii, E. Macii, M. Poncino, and R. Scarsi. A discrete-time battery model for high-level power estimation. *In Proceedings of Design, Automation and Test in Europe*, 2000.

[32] D. Rakhmatov and S. Vrudhula. Time-to-failure estimation for batteries in portable electronic systems. *In Proceedings of the International Symposium on Low Power Electronics and Design*, August 2001.

[33] D. Linden and T. B. Reddy. *Handbook of Batteries(3rd edition)*. McGraw-Hill, 2002.

[34] Samuel T. King, George W. Dunlap, and Peter M. Chen. Debugging Operating Systems with Time-Traveling Virtual Machines. *In the Proceedings of USENIX Annual Technical Conference 2005*, April 2005. Anaheim, CA.

[35] Sudarshan M. Srinivasan, Srikanth Kandula, Christopher R. Andrews, and Yuanyuan Zhou. Flashback: A Lightweight Extension for Rollback and Deterministic Replay for Software Debugging. *In the Proceedings of USENIX Annual Technical Conference 2004*, June 2004. Boston, MA.

[36] Stargate: a platform X project. `http://platformx.sourceforge.net/`.

[37] Mark Carson and Darrin Santay. NIST Net - A Linux-based Network Emulation Tool. *In the Proceedings of ACM SIGCOMM special issue of Computer Communication Review*, 2003.

[38] Stephen Hemminger. Network Emulation with NetEm. *In the Proceedings of Linux Conference AU*, April. 2005.

[39] Luigi Rizzo. Dummynet: a simple approach to the evaluation of network protocols. *In ACM Computer Communication Review*, 27(1):31–41, 1997.

[40] Network Emulation with the NS Simulator. `http://www.isi.edu/nsnam/ns/ns-emulation.html`.

[41] NS-2 network simulator. `http://www.isi.edu/nsnam/ns/`.

[42] QEMU: A Generic and Open Source Processor Emulator. `http://fabrice.bellard.free.fr/qemu/`.

[43] Richard M. Fujimoto. Time warp on a shared memory multiprocessor. *Transactions of the Society for Computer Simulation International*, 6(3):211–239, 1989.