

sQualNet: A Scalable Simulation and Emulation Environment for Sensor Networks

Maneesh Varshney, Defeng Xu, Mani Srivastava, Rajive Bagrodia
University of California, Los Angeles

Abstract—Although there is growing interest in the use of physical testbeds to evaluate the performance of applications and protocols for sensor platforms, such studies also encounter significant challenges that include the lack of scalability and repeatability, as well as the inability to represent a diverse set of operational scenarios. On the other hand, simulators can typically address the preceding problems but often lack the high degree of fidelity available to the analysts with physical testbeds. In this paper, we present the design and implementation of sQualNet - an accurate and scalable evaluation framework for sensor networks that effectively addresses the preceding challenges. In particular, sQualNet integrates sensor network operating systems with a very high-fidelity simulation of wireless networks such that sensor network applications and protocols can be executed, without modifications, in a repeatable manner under a diverse set of scalable environments. sQualNet extends beyond the existing suite of simulators and emulators in four key aspects: first it supports emulation of sensor network applications and protocols in an efficient and flexible manner; second, it provides an efficient set of models of diverse sensing phenomena; third, it provides accurate models of both battery power and clock drift effect which have been shown to have a significant impact on sensor network studies; and finally it provides an efficient kernel that allows it to run experiments that provide substantial scalability in both the spatial and temporal contexts. We demonstrate these advantages, both qualitatively and quantitatively, by means of various case studies including a study that evaluates an experiment with up to 1000 nodes running networking application like Surge faster than real time.

I. INTRODUCTION

Over the past few years we have witnessed the growing use of sensor networks for a diverse set of real world applications. Developing such applications is made challenging owing not only to the resource constrained nature of the sensor platforms but also due to the manifestation of these constraints in the software development environment. For these systems the edict that the complexity in the implementation of protocols is equivalent to or even greater than the complexity of the protocol itself has significantly influenced how the researchers prototype, validate and evaluate their designs. Instead of generating simulation models to study protocols and applications and subsequently porting them to real implementation, as is the case in Wi-Fi network research, sensor network researchers prefer to prototype the implementations directly for the target platform in a physical test-bed, so that the operating constraints are appropriately reflected in the design of the software. The cheap cost of sensor nodes, availability of software and growing experience in programming them has further facilitated this approach.

Although the ability to prototype system software and applications directly into sensor test-beds offer numerous benefits, physical sensor network testbeds suffer from significant limitations, including those of scalability, in both the temporal and spatial dimensions, as well as the ability to evaluate operations of the networks in diverse operational scenarios. The scale of the test-bed is typically too restrictive to evaluate and validate the protocols for large target network sizes. Moreover, the execution cannot run faster than real time which makes it virtually impossible to observe the behavior of the network for large time durations. Any system failure or performance issue that is expected to arise, say, one month into the operation of a network

cannot be detected in the evaluation phase, unless the designer is willing to wait this duration. A physical test-bed also offers limited input conditions to the implementation. It cannot be as rich as a simulation testbed in creating diverse scenarios for test cases or observe the behavior in pathological conditions. Moreover, it is difficult to achieve repeatability in executions making comparative analyses, relatively difficult. Considerations such as these necessitate a role for simulation and emulation in the evaluation of sensor networks.

Ideally, an evaluation framework should be able to provide the same programming and execution environment as real platforms, while addressing the issues mentioned above. We consider the following to be the four key requirements of a sensor network evaluation framework:

- **Software Emulation**, that is, the framework must be able to run sensor code developed for a physical platform, with little or no modifications. In other words, the framework must support software prototyping, where the assertions made and behavior observed for the software during the evaluation phase, are valid for the real deployment.
- **Environment Modeling fidelity** or the accuracy with which the physical operating environment for the sensor network is represented accurately in the framework.
- **Scalability**: Support for scaling the network operation both in *spatial* (size of network) and *temporal* (simulation duration) sense.
- **Flexibility** or expressiveness in the ability of the kind of operational scenarios as well as types of networks that can be modeled.

We present in this paper the design and implementation of sQualNet¹ - an accurate and scalable evaluation framework for sensor networks that addresses the challenges mentioned above. The design approach considered is to extend and enhance a state-of-the-art network simulator, QualNet [1], with sensor network specific models. In the following we discuss why these challenges are of particular relevance to sensor networks, maybe above and beyond the need of traditional Wi-Fi or cellular networks, and how the sQualNet design addresses them.

One of the primary objective of a simulation framework is *environment modeling fidelity* or the accuracy with which the physical environment is represented. In general, modeling fidelity or accuracy is expressed at various levels:- wireless communication - radios and propagation models, sensors, protocol models, system issues (operating systems, timer granularity etc), energy usage, applications or traffic models etc. It has been shown that higher fidelity of one aspect of simulation can be overshadowed or significantly diminished by the model inaccuracy in other components. For example, in [2] it was shown that detailed modeling of propagation effects and radio

¹The software is available for download at <http://chenyen.cs.ucla.edu/projects/whynet/download.php>

operations can affect the metrics at higher layers, and [3] showed similar results for the case of accuracy in battery models and traffic models. Thus it is imperative that the evaluation framework be accurate or high-fidelity for *all* the components of the sensor networks. The issue of software (application and OS component) fidelity can be addressed via support for emulation of these components in the framework, but accurate representation of the physical environment must still be addressed via high-fidelity simulation models, that can be executed in a synchronous manner with the emulated software.

A second requirement of any evaluation framework is scalability. The spatial scalability - the number of nodes that can be simulated - is an issue of greater importance for sensor networks for two reasons: the sensor nodes are cheap and can be deployed in large quantities; and because the power source is usually non-renewable, they are deployed with higher redundancy. However, simulating large number of nodes is only a part of the picture, we should also consider the wall-clock time used to execute the model. Traditional simulation of Wi-Fi networks have typically focused on the evaluation of their stationary or steady state, where it is sufficient to simulate the network for a few tens of minutes of operation. Sensor network evaluations, on the other hand, require a more critical analysis in non-steady states too - for example, when a network is activated, when a node fails or when a part of the network runs out of its power source. Thus the simulation time is typically the lifetime of network, which could be many days to few months (of simulated time). Existing simulators that operate at real-time or with low execution speedups are not suited to meet such evaluation requirements.

Finally, the evaluation framework should be expressive in the kinds of scenarios and networks that can be modeled. One of the emerging network architecture is hierarchical organization of heterogeneous network components. The heterogeneity can be at the protocol stack level or device level. Unfortunately, current simulators provide limited to no support for modeling such mixed networks.

sQualnet, with its design approach of using a network simulator, enhancing it with sensor models, and integrating with sensor OS emulation, is able to address these challenges along three dimensions: using QualNet network simulator allows sQualnet to utilize the efficient simulation engine, accurate models of radio, channel and a rich suite of protocol models. The design of sensor OS emulation framework and its integration with the base simulator allows not only efficiency and scalability but also flexibility in modeling diverse network scenarios. Finally, we have added detailed models specific to sensor network operations such as sensing phenomenon, energy usage, clock drift etc, which enables evaluating these networks in a high fidelity environment.

We demonstrate these benefits by means of several case studies. The first case study used detailed channel models to study the sensor network operations and concluded that abstract models suffer the risk of predicting incorrect trends or behavior in the network. The second case study argues for emulating sensor nodes by studying the impact of sensor operating system's timer management on a MAC protocol's performance, an effect which would have otherwise gone unnoticed in a study that only relied on the simulation model of the same protocol. The third case study illustrates the use of the specific sensor network models that have been added to sQualNet including models of clock skew and battery power consumption. Finally, we demonstrate the ability of sQualNet in modeling a heterogeneous network, where the sensor nodes are emulated and the remainder of the wireless network is simulated.

II. SOFTWARE MODELING FIDELITY

The evaluation of a new protocol or a design via simulation has dual objectives: observing the behavior and performance under diverse input conditions to test *correctness of the algorithm* and identifying implementation bugs and flaws, that is, *validation of implementation*. Only after sufficient confidence has been achieved for both code and algorithm correctness in this phase, does the designer moves to the next step - deployment in physical systems. However, if there is little or no overlap between the programming methodology used in protocol simulation and that used for protocol implementation in sensor nodes, the code has to be ported or written again for these nodes. This porting can introduce new bugs which become harder to identify in the implementation. Moreover, the sensor nodes have their own design limitations that can affect the correctness of the implementation. As a result, it is generally preferable to use emulations where the software used for evaluation is portable with little or no modifications to the physical node and the behavior observed and assertions made regarding protocol operation in simulation remain valid in actual deployments.

A variety of emulators have been developed ([17], [20], [21], [18], [14], [19]) that offer varying levels of fidelity in terms of their ability to minimize the modifications necessary to port the emulated software to operational environments. However, almost all of them lack detailed models of the physical environment, most notably to accurately model the radio and channel effects, with the result that the behavior observed in the emulators is not representative of real deployments. Another drawback with existing emulators and simulators is that they are often designed for a specific OS and are not easily extensible to support diverse operating systems or networks (e.g. TOSSIM, TOSSF can only run TinyOS). Our goal is to extend the ability of sensor network evaluation frameworks by improving their accuracy (in representation of applications, protocols, and the physical environment) and their flexibility (to support heterogeneous networks as well as heterogeneous sensor operating environments)

A. Emulation Design Overview

We present first an abstract design overview, that address the requirements we have placed on an emulation framework: a) the emulator shall be independent of the underlying discrete event simulator, that is, the design should work with any network simulator, b) the emulator shall be independent of the underlying sensor node operating system; i.e. the design should incorporate any current or future OS and possibly multiple OS in a single evaluation c) the design shall be scalable to large number of nodes, d) the execution time should be meaningful to study networks that run for long durations, and e) the emulator shall provide the ability for the emulated sensor network to interact with other kinds of networks (e.g. IP, WiMaX).

The two key components of this emulation architecture are Discrete Event Simulator (DES) and the sensor node (SN). Fig 1 shows the block diagram of each component and delineates the integration approach. The DES has an event queue, an event scheduler, handlers for various types of events, a simulation clock and initialization routine. Simulation proceeds in this initialization routine that inserts some events into the event queue. Subsequently, the events scheduler dequeues an event with the earliest time-stamp, advances the simulation clock and calls the relevant event handler based on the type of event. This event handler can in turn insert more into the queue. At the other end, we have the sensor node with these key components: the OS or the kernel, protocols and applications and hardware devices

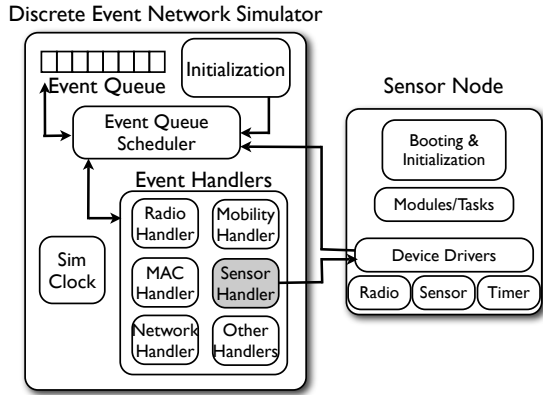


Fig. 1. Design of emulation

with device drivers. The devices of interest to us are radio, sensor ADC, UART serial port and hardware clock.

For the integration of DES and SN components, we propose two sets of interfaces. From the DES point of view, the SN is a special layer and we introduce an event handler for SN that will be called by the scheduler to process the events that are relevant to the sensor node. Conversely, at the SN, the DES is viewed as device platform, that is, set of drivers for the various hardware devices. The device managers, in any case, are abstracted out of the kernel to make it portable for different hardware architectures. We can add a “DES hardware platform” which masquerades as hardware resources to the SN. The details of these interfaces will be presented in the subsequent sections.

This design addresses the challenges that we have outlined earlier. We make no assumption on the DES or the SN except for the basic organization structure as presented in fig 1. This makes it possible to use different operating systems, say TinyOS and SOS, in a single simulation execution. Being integrated with DES allows not only the advantages of the efficient event scheduler but also the availability of models for radios, channels, mobility, protocols etc.

In the following we describe the specifics for the integration of the QualNet network simulator and the SOS sensor operating system[4]. Although, we use these specific components because of our familiarity with their design, the proposed architecture may be used to similarly integrate other simulators and or sensor operating systems.

B. Implementation

There are various ramifications of the design approach just outlined for the implementation purposes. The first one presents challenges in modeling the sensor node software directly as an event handler. Recall that the DES scheduler calls an event handler when a corresponding event occurs and will be able to reclaim control of execution only when the handler “returns”. The sensor node, however, is an operating system that once booted *never returns*. To address this problem we utilize *process-oriented simulations* as used by Simula and more recently PARSEC. In this approach, instead of implementing the simulation by a set of event handlers, each object is executed as a separate thread and interacts with the event scheduler via two interfaces: *wait_until(predicate)*, where the entity process is blocked until the predicate is true, and *advance_time(Δt)*, when it is suspended until the requested time has advanced in the simulation. These are the only two places where a context switch can happen between the scheduler and the entity process. So our first implementation guideline is to execute the SOS code as a separate

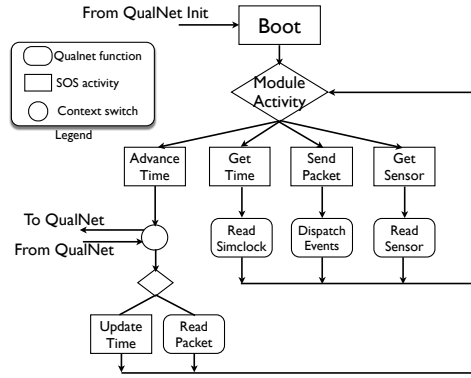


Fig. 2. Control flow of a SOS instance thread

thread or process, and the context switch between SOS and QualNet will happen at either *wait_until*(there is packet in receive buffer) or *advance_time*(timeout).

Having established that SOS code must be executed as a separate processing unit, an obvious design decision would be to execute it as a separate process. However, this would tend to restrict scalability due to their greater context switch time and higher memory requirements. So, for the sake of eventual scalability, we decided to implement the SOS emulation using the *pthread* library. However, using threads to implement SOS instances in the emulation framework implies that the global state used by each SOS node must be suitably protected so that each SOS node instance could continue to use this global data, however the different SOS nodes did not invariably share the global data. A simple implementation of user-level threads was used to avoid this problem: whenever a SOS thread does a context-switch or transfers control to QualNet, its global data is stored in a unique activation record indexed by its node address, and when this SOS instance is subsequently resumed, the corresponding activation record is used to restore its global data. This approach is different from TOSSIM, which used compiler modifications to change the references of global data into an array indexed by its address, making it more efficient than storing/restoring data. We, however, argue for our approach as it does not require any special knowledge of sensor node (the NesC language and the compiler, in case of TOSSIM) which was one of our key requirements. The use of user-level threads considerably reduces the context switching times as, in most cases, we are storing and restoring pointers which does not impose any significant overhead. The implementation efficiency has been verified via large scale tests which demonstrate our ability to emulate a sensor network with over 1000 nodes, each of which is running an independent copy of SOS.

The final piece of implementation is the mapping of the user-level threads that represent each SOS node instances to an OS thread (or a process). At one extreme, we can map each SOS instance to a unique thread, which will however violate our scalability requirements. The approach we have taken instead is to map *all* the SOS instances to a *single* OS thread. This is made possible due to the simulation technique we use and the data context switch that is implemented. The design also supports an alternative mapping that we plan to use for parallel execution of the emulation in the future. In this mapping, the implementation can have as many threads as the number of processors and the SOS instances are partitioned into sets, with each set assigned to one of the OS threads. In the remainder of this paper, we use the terms SOS node instance and thread interchangeably to refer to a unique instance of SOS.

Call	From → To	Context switch ?	Parameters	Effect
Boot	Qualnet → SOS	Yes	node_id	Boots SOS
Advance Time	SOS → Qualnet	Yes	$\Delta t_{request}$	SOS requesting to advance time
Time Advanced	Qualnet → SOS	Yes	$t_{current.time}$	the request time has advanced
Receive Packet	Qualnet → SOS	Yes	$t_{current.time}$, Packet	receive a packet and advance time
Get Current Time	SOS → Qualnet	No	none	Read Qualnet's Simclock
Send Packet	SOS → Qualnet	No	Packet	Dispatch radio events
Get Sensor Value	SOS → Qualnet	No	sensor_id	Read sensor value
Post on serial port	SOS → Qualnet	No	sensor_id	Post data on uart

TABLE I
INTERACTIONS BETWEEN QUALNET AND SOS

C. Scheduling between Simulator and Sensor Node

The state flow diagram of the SOS thread is shown in fig 2. As a part of QualNet's initialization, SOS node is booted up in the SOS thread. At this point, the QualNet thread blocks itself to transfer control to the SOS thread and will resume only when explicitly signaled. The SOS thread boots the operating system, initializes the hardware and other services and load the modules. These modules can interact with the following hardware resources: timers, radio, uart and sensor, all of which are modeled by Qualnet. What happens in each of these case will be considered in subsequent subsections, but it suffices to mention here that the context switch will occur only when the node is waiting the time to be advanced (after starting a timer) or a packet to be received. For all the other functions the SOS thread retains the execution and continue to run until it reaches the two conditions mentioned above. When that happens the SOS thread stores its global data, signals the Qualnet thread to continue and blocks itself.

Thus the simulator thread and the SOS thread(s) schedule each other, former when it is processing a SOS event, latter when blocked on timer or packet receive. It should be noted here that the SOS execution cannot be interrupted by asynchronous events, which is a limitation of all the discrete events simulation systems.

The way we have implemented scheduling and synchronization ensures that the SOS thread which is executing has the view of the current simulation time and that there is no sleeping node that is waiting to woken up before this time. This ensures the causality constraint, that is, no node can move so much ahead in time so that there can be some messages from other nodes which should be received "in the past".

D. Interfaces between Simulator and Sensor Node

There are three classes of interactions between the simulator and the sensor node: initialization, time management and hardware resource usage. The complete list of interactions is shown in table II-B. We have already seen how the SOS node is booted and that it waits for time to advance or to receive a packet. In both cases, a context switch occurs and control is transferred to simulator. When the thread is unblocked it is informed of the new time, along with received packet if there is one. The rest of interfaces do not cause context switch, they are executed from SOS thread. We will discuss these calls in this section.

Sending Packets: the node inserts radio events in event queue. These events in any network simulator are 'StartReceivingPacket' with the propagation delay and 'EndReceivingPacket' with an additional delay of transmission time. Note that since the SOS is running as user-level thread it has access to the address space of the simulator and uses the simulator's API to send packet directly. After dispatching the SOS continues with its execution.

A point to note here is that we have decided to use packet level transmission as opposed to bit or byte level. As mentioned before,

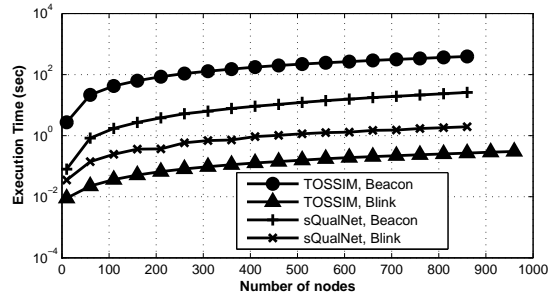


Fig. 3. Time required to execute 10 seconds of Blink and Beacon applications

the hardware and device drivers are abstracted by the simulator, and we consider the lower part of MAC protocol close to the hardware that it should also be modeled in the simulator. This design results in considerable speedup which is shown in fig 3. Execution time of two applications is considers: *Blink*, that does not make use of network stack and *Beacon* which periodically broadcasts beacons and listen for packets from others. We compare our design with TOSSIM that implements bit level transmission. For our benchmark we observe performance improvements of about 30X.

Read Current Time: Since the processing happens in zero time, a call to read the current time at SOS is simply the simulation clock time.

Reading sensor value: We have implemented sensing channel in Qualnet (sec III-B) which provide the interface *GetSensorReading(int sensor_id)*. The parameter is used to distinguish if there are multiple sensors in a single node. The SOS thread calls this function to retrieve the sensor reading and continue with its execution.

Send message on serial port: Messages can be transmitted over serial uart port to communicate with 'host' computer. The sensor nodes can dump the messages to another node over this interface. The functionality is similar to sending packets on radio.

E. Modeling Heterogeneous Networks

There has been a lot of interest in heterogeneous multi-tier networks that are envisioned to be the enabling architecture for large scale sensor networks. In these networks, there are some *gateway* nodes that have dual network interfaces: one to communicate in the sensor subset and another to communicate over high throughput, larger range networks such as 802.11. These gateway nodes can be constituted in two ways: a high end node with 802.11 interface and connected with sensor node over a serial port connection, or a unified approach in which case the node has two network interfaces, one each for the two networks.

In sQualNet, we provide modeling support for both device architectures. The two-node serial port connection architecture is realized using the serial uart communication that we have mentioned in the previous section. For the latter, we have modified QualNet to add support for multiple network protocol stacks, one for IP stack and

the other for sensor node emulation. In section IV-D we describe modeling of a hierarchical network.

III. MODELING FIDELITY

There remain many physical phenomenon that are specific to sensor network operations and therefore, not sufficiently addressed in network simulators. We present in this section the models we have added in the sQualNet framework.

A. Modeling Clock Drift

There are many activities relevant to sensor networks that require the different nodes to be synchronized. An illustrative example is duty cycling MAC (e.g. [7]) that expect a node to wake up at the same time as neighboring nodes to communicate. Also certain sensing applications require the time-stamp at which a particular event occurred at remote nodes. The performance of these protocols and applications can be highly dependent on the relative synchronization of the clocks in different nodes. However, experience with real nodes has taught us that the nodes are never synchronous and some amount of drift is always present. To address this problem several synchronization algorithms have been developed. For these applications and the algorithms to be studied in context of simulation there is a need to model this aspect of clock behavior.

For modeling purposes we distinguish between *clock drift* and *clock skew*. To understand these terms let us first consider the basic clock operation. A clock has an oscillator that vibrate at a set frequency and a counter that counts the vibrations and advances the time by one unit when a determined number of vibrations have occurred. Final piece is the *zero-point* or the initial time of the clock. Now errors can arise at two places: first, the zero-time may not be set synchronously for all clocks, that is, a given clock is ahead or behind other clocks as if they are in “different time zones”. This aspect is called clock skew. Also it can happen that the oscillator is not perfect and it vibrates at a frequency that is different from the correct value. In this case the meaning of a time unit changes – one second on this clock may be less than or greater than one second on another clock. This is called clock drift. So we can have the case when there is clock skew but no drift, however, if there is clock drift, a skew will always develop.

In [5] a model for clock drift has been presented:

$$f = f_{nom} + \Delta f_0 + a \cdot (t - t_0) + \Delta f_n(t) + \Delta f_e(t)$$

Here f is the frequency of an inaccurate clock, f_{nom} is the nominal frequency, t_0 is starting time, a is the aging rate, Δf_n is the short-term frequency instability or noise term and Δf_e is the environmental term, primarily temperature. The short term frequency variation was neglected as it was claimed to be zero mean and does not contribute to long term aging. After other assumptions, the model for clock time was derived as:

$$\Delta \phi(t) = \Delta \phi(t_0) + \Delta f_0(t - t_0) + \frac{a}{2} \cdot (t - t_0)^2$$

In the simulations, the aging rate was considered to be gaussian distributed with zero mean and variance of $2 \times 10^{-6} * f_{nom}$ and Δf_0 was also considered to be gaussian with zero mean with variance $2 \times 10^{-5} * f_{nom}$.

We have modeled both clock drift and skew in sQualNet. It should be noted that the simulation maintains a global view of time via the ‘simulation clock’ but the different nodes can have different notions of current time and time intervals based on the corresponding model of clock drift. What we need is a translation mechanism that converts the global simulation time to local time based on the clock at each node, and vice versa. This is accomplished by keeping the model

parameters for each node (s_n for skew, a_n and b_n for drift) and making the following two modifications to timing primitives:

- 1) *Get Time*: convert the simulation time t_{sim} to the node’s time t_{node} .
- 2) *Set Timer*: convert the node’s timer interval Δt_{node} to simulation interval Δt_{sim} .

For the quadratic drift model, the relation between these are:

$$t_{node} = t_0 + a_n \cdot (t_{sim} - t_0) + b_n \cdot (t_{sim} - t_0)^2 + s_n$$

$$\Delta t_{node} = a_n \Delta t_{sim} + b_n \Delta t_{sim} \cdot (2t_{sim} - 2t_0 + \Delta t_{sim})$$

Thus, when a node requests the current time, the simulation time is converted to local time using the first equation. The second equation is used when a node requests a timer of interval Δt_{node} , in which case an event is scheduled in the simulator with a delay of Δt_{sim} .

B. Modeling Sensing Channel

Sensing is the *raison d’être* for sensor networks; thus the quality of the sensing models will directly bear on the utility and accuracy of the corresponding simulator or emulator. Many existing sensor network simulators and emulators use a simplistic channel model that reports random readings to the sensors. The reason it is important to have good sensing channels is that they are the part of traffic model – they drive traffic in the network. Also there are many algorithms that aggregate the data and such algorithms can be evaluated in a better way if the sensing data is more representative of the real world.

In our opinion, a primary reason for the lack of sophisticated channel models is due to difficulty in expressing sensing channels in sensor net simulators. Consider any physical phenomenon such as temperature or pressure. We have no easy way of representing the values over the entire terrain and how they change over time. What we need is a compact yet expressive way to represent such channels. We outline here two kinds of sensing channels that have been implemented in sQualNet: *diffusive sensing channels* and *mobility sensing channels*. The diffusive sensing channel assigns some value at each space-time coordinate in simulation and is applicable for phenomenon like temperature, humidity etc where the values change across space and time. In the mobility sensing channels target(s) are mobile and trigger the sensors only they are in the vicinity. This channel is useful for location or angle determination class of sensing applications.

For diffusive channel we have assumed a simple but expressive representation. Consider that the initial value of the property (temperature, for instance) is zero or constant at all the points in the terrain. As the simulation proceeds the four corners will always have the same value but the intermediate locations can show dynamism as determined by a configuration script having the following format:

Location (Time1, Value1) (Time2, Value2) ...
E.g. (100m, 100m) (10sec, 50°C) (40sec, 10°C)
(200m, 150m) (15sec, 0°C) (60sec, 100°C)

The first line will read as “at location 100m,100m the metric has initial value of 0°C which will increase linearly and reach 50°C at 10 second, after which it will go down to 10°C at 40 seconds, where it will remain until the end of the simulation”. To calculate the value at any other point we perform *bilinear spline interpolation* of the four corners and the locations that are specified in this configuration file. For example, if a sensor node located at (50m,50m) requests the value at time 30sec, this value will be calculated using 2-dimensional interpolation of the four corner points at 0°C, (100m, 100m) with

value 30°C and (200m,150m) at value 33.3°C. This illustrates the ease with which a dynamic environment can be expressed with very little configuration effort.

The second channel is mobility channel in which target(s) move about and if they are in proximity of sensors will register an event at the node. A naïve approach is to generate event to each node whenever the target has moved a distance equal to mobility granularity. This method creates an overhead due to large number of events in simulation. Also many sensor nodes do not even consider all the sensing data, they might be sleeping at that time. To reduce the number of events we perform some pre-computations: we calculate using trigonometric functions when the target is going to enter and leave the vicinity of a node and events are generated at these time instances only. This optimization reduces the total number of events by a order of magnitude. We will use this sensing channel model in one of the case studies (sec IV-C).

C. Modeling Battery

An ideal battery is usually viewed as a reservoir of charge from which an amount equal to the load can be subtracted, until the capacity falls to zero. However, this is not an accurate model as a real life battery shows *non-linear discharge* behavior and *recovery* effects. Former implies that the capacity of the battery is dependent on the rate of discharge. Consider an illustrative example of a battery having nominal capacity of 100 mA-hour. If a load of, say, 1mA is applied then the battery is expected to work for 100 hours, however, if 500mA is applied the battery lifetime in practice will be less than 2 hours since the lifetime decreases if the battery is drained at higher rates. The battery can also recover some of the capacity lost earlier, if it is allowed a rest period or discharged at lower rate.

To model these behaviors of the battery we have utilized the model in [6], in which the non-linearity and recovery is explained by process of diffusion of electrolyte. Let $I(t)$ is the current drawn from the battery at time t , then the battery capacity used after time T is given by

$$C(T) = \int_0^T I(t)dt + 2 \sum_{m=0}^{\infty} \int_0^T e^{-\beta^2 m^2 (T-t)} dt$$

The lifetime of the battery is obtained by solving the equation $\alpha = C(L)$. A requirement of this model is that a history of all the loads applied in the past has to be maintained to calculate the lifetime. This has an impact on both memory requirements to store this data and the execution time. The model was primarily designed for user applications running on PDAs where loads profile lengths are 10-100 whereas in sensor networks this history list can go up to 100s of thousands. We worked on two optimization, first to speed up processing and second to reduce the memory requirement.

The first optimization has been published by us previously in [3] but we give a short description here for sake of completeness. The equation above was rewritten as $C = I \times T + L$, where the first term is the ideal discharge and L is the excess discharge. The value of L is maximum, L_{max} , at time 0 which is defined as the time when the load was taken off the battery and decreases in magnitude with time, reaching zero after a sufficiently long time. For the special case of loads applied for milli-seconds level, which is the typical packet reception or transmission time, and for small magnitude of individual loads, we found certain invariants. First, the value of L_{max} is linearly related with I and T , and secondly, the rate of decay of L from L_{max} to 0 is independent of both I and T . This curve can be approximated as a polynomial function. Thus we have transformed the original equation requiring excessive computation into polynomial expressions which greatly reducing the computation effort.

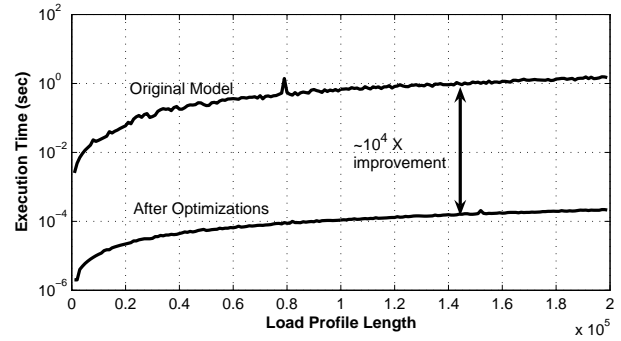


Fig. 4. Execution speed up with optimizations for battery model

The second optimization is to reduce the memory requirement. First, we store data up to H seconds in the past only, that is, we discard all history data that is older than this threshold. This value is derived by studying the curve for rate of decay of L and H is derived when the value goes below a threshold L_{min} . Next, instead of keeping all the events we merge some events that happen in interval Δt into a single entry. Both optimizations reduce the amount of memory that is required to store the load profile. We show the benefit of these optimizations in fig 4 where we can observe about four orders of magnitude improvements in performance while keeping the inaccuracy below 0.1% compared with the original model.

IV. CASE STUDIES

Simulation fidelity is critical for effective evaluation of any design. An inaccurate or highly abstracted model may predict results that are substantially different from real deployments, but more importantly, can display counterfactual trends or behavior of the networks or protocols. We have discussed how sQualNet addresses these fidelity concerns through three components of the design: first, using a state-of-the-art network simulator as underlying base driver for discrete event simulations supports the use of the simulator’s detailed models for radio and channel in sQualNet; second, emulating the sensor operating system facilitates the study of applications and protocols exactly as they would be implemented in the context of the corresponding OS; finally, we have added models for other aspects of sensor networks (e.g., sensing channel, clock drift) that we felt were not adequately represented in the current suite of simulators.

In the following we demonstrate how the modeling inaccuracy can lead to prediction of incorrect trends in the network. We consider three case studies, one each for the three components of sQualnet mentioned earlier. We utilize QualNet’s fast fading channel model to study the impact on a multi-hop network. Advantages of software emulation are shown by studying the effect of timer granularity within a sensor operating system on the applications. We show in third case study how the clock drift model can help in a better evaluation of certain applications. Finally, we also present a case study that models a heterogeneous network, demonstrating the other benefit of sQualNet - the ease with which such networks and scenarios can be realized and studied.

A. Fidelity through QualNet’s models: Impact of Channel Fading

Perhaps the most complex component of a wireless network, from a modeling perspective, is the wireless channel. The channel is commonly modeled with empirical and stochastic models of pathloss, fast fading and slow fading, or with deterministic techniques such as ray tracing. However, the current suite of sensor network emulators have favored a very abstract model of “connectivity graph” channel (e.g. [17]). We challenge the use of such abstract models by demonstrating

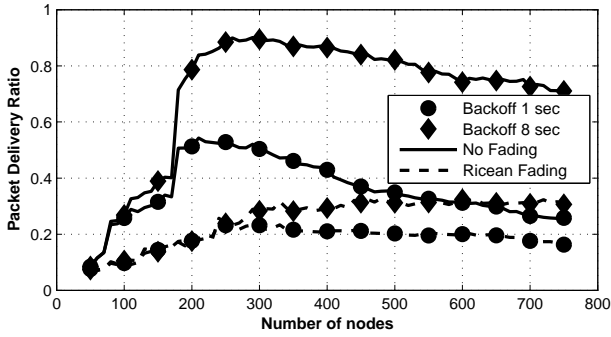


Fig. 5. Effect of fast fading on packet delivery ratio

the impact of fading phenomenon. We consider a very simple network design where nodes periodically obtain readings from sensors and send the raw data to a base station (Surge application) via a tree formed using a tree-routing protocol. This is a characteristic mode of operation of the sensor networks and significant impact of channel fading on the observed network configuration argues for incorporation of detailed channel modeling for sensor networks.

We consider the problem of finding the optimal density of sensor nodes that maximizes the number of packets received at the base station. In our simulations the radio range was about 35 meters and we configure a terrain of $350m \times 350m$, thus the network diameter is about ten hops. A base station is placed at the center and additional nodes are deployed at random locations. The *surge* and *tree-routing* modules directly supported by the SOS distribution were emulated. CSMA was used as the MAC protocol. We recorded the percentage of packets received at the base station under two experimental conditions: one where there is no fading and the channel is affected by ‘Two-ray’ pathloss and interference only, and second with *Ricean* fading. Influence of a protocol parameter – initial random backoff – was also studied in conjunction with fading. Both tree routing and surge applications periodically broadcast packets and in order to minimize the probability of collision each node starts the periodic transmission phase after an initial random backoff. This backoff is a random number in the range $[0, \text{MaxBackoff}]$, where the default value of MaxBackoff is 1 second.

The packet delivery ratio at the base station for the various cases is shown in fig 5. For sake of brevity we show results for Ricean K parameter equal to 5, which we consider as moderate fading. First consider the two plots for the case of no channel fading. These are the results we might expect from any simulator that utilizes a highly abstract model of the channel. In this case we observe that the PDR is low for small number of nodes since they might not be connected and the network may be partitioned, improves as the node count is increased, but then begins to decrease as the node count is increased further. The latter effect has been attributed to an increase in hidden terminal collisions. These results will lead us into believing that there is an optimal density of sensor nodes and anything above or below this optimal value with result in performance degradation. Such an observation, if it becomes a part of common folklore, will drive future research into areas dealing with configuration and deployment of the networks. For example, a suggestion to increase the density to improve, say network lifetime, may be discouraged based on these results. However, if we examine the more accurate simulation case of channel with fading, we observe different behavior. Ignoring the absolute values but rather focussing on the trend, we see that the PDR increases and then stabilizes, at least for the number of nodes shown in graph. This is the key result that we would like to highlight: using

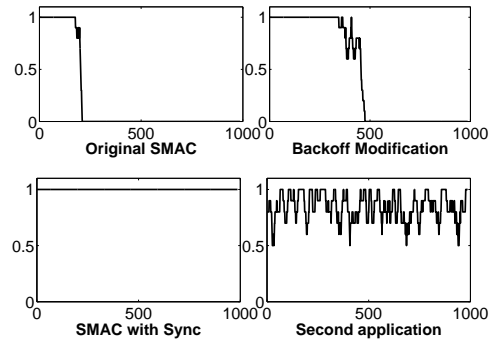


Fig. 6. Effect of SOS timer management on SMAC

abstract models can predict incorrect trends in the network.

We also observed that increasing the backoff interval improves the PDR. This parameter has an impact on the probability of collisions of transmissions. For a channel with no fading we see an improvement of 250% which is actually inflated compared to the more conservative value of 150% predicted by the fading channel model. Thus, using accurate models can also provide significant insights into the relative importance of specific types of optimizations and modifications being evaluated for network design.

We recognize that ours is not the first attempt to caution against using inaccurate models in general and channel models in particular. The larger point we hope to make is that even for such a relatively simple sensor network configuration the effects of model accuracy are far reaching and need to be recognized.

B. Fidelity through Emulation: Impact of SOS Timer Management

In this section we demonstrate the benefits of evaluating applications in the context of an emulated sensor operating system. Detailed operation of operating systems are not faithfully modeled in network simulators and these details can have an impact on evaluation results. In this study we will consider the effects of timer management and timer granularity at the operating system.

A simple network configuration of only two nodes is considered with one node sending packets periodically to the other. We have implemented the SMAC [7] protocol in SOS and compare the results of the study using the SMAC implementation as part of the SOS emulation with that of a study using a simulation model of SMAC. In both cases, the nodes alternate between sleep and listening states. In the experiments, the nodes sleep for 1024ms after which they wake up and listen to the channel for 300ms. The transmitter will do a random backoff in the interval $[0, 63ms]$ before sending one packet and then transition to the listen state. A node can optionally send a ‘SYNC’ packet to synchronize the time with its neighbors. In our experiments the two nodes are perfectly synchronized, that is, we have disabled the clock drift model.

For our initial investigation, we synchronized the sleeping schedules of the two nodes at the start of simulation and subsequently disabled transmission of SYNC packets. This is reasonable as the clocks are perfectly synchronized and the nodes should wake up and go to sleep in perfect unison. The throughput observed at the receiver is shown in fig 6 as a function of time. Note the unexpected result with the emulated SMAC (graph titled Original SMAC), where the throughput appears to relatively quickly go down to zero. The results from the simulated SMAC show the expected result of 100% reception. The packet traces with the emulated SMAC revealed that the sender and receiver were in fact out of sync in the latter portion of the experiment, with the receiver sleeping at those instants when the sender was transmitting. This was unexpected – the nodes are

synchronized at the beginning and the clocks are perfect, so what caused the skew? It required delving into the SOS kernel to identify the cause: the minimum timer granularity. The default value of this parameter is 10ms, that is, if an event is scheduled for a shorter interval, it would still be triggered only after 10ms has elapsed. Now the unexpected behavior with the emulated SMAC can be explained: suppose the nodes wake up simultaneously at time T . At this time, the transmitter does a backoff of, say, 3ms, sends a packet and sets a timer for 297ms so that it may transition to the sleep state at $T + 300$ ms, same as the other node. However, due to the setting of the minimum timer granularity in the SOS kernel, the first timer is actually fired after 10ms and the second after 297ms, so the node will actually finish its listening state at $T + 307$, and this creates a skew of 7ms! After a sufficiently long duration, this skew will accumulate to completely offset the alignment of listening states of the two nodes.

Thus even with perfect clocks, a skew can occur due to the interaction with other OS components. We avoided the above problem with a simple modification: requiring that backoff be in the interval [10ms, 63ms). Unfortunately this modification also failed to correct the problem and the reason this time was the *maximum* timer granularity. The kernel has a maximum timer granularity (250ms by default in SOS) and any timer greater than this value is divided into multiple timers, each sub-interval no greater than 250ms. In this case consider that the protocol made a backoff request of 48ms and subsequently listen state timer of 252ms. The latter timer is translated as two timers of 250ms and 2ms. The second timer will be fired after 10ms for reasons mentioned above and the node will be skewed by 8ms.

With these two observations we fall back to the option of using sync packets. This packet was sent at every 10^{th} listening period as advised in the original design. With this addition we found that both problems are taken care of since the sync packets make sure that the clock skews do not accumulate. But here also we encountered an even more interesting case. A second application was running at the receiver that was generating a periodic timer of 40ms and the co-existence of this application resulted in degradation of throughput. The explanation for this behavior is that the timers from all applications are handled at a single place in the kernel. The interplay of timers from this application and the SMAC protocol created scenarios similar to above that resulted in clock skew. (Note again that the second application did not affect the simulated SMAC experiment).

From the preceding examples, we conclude two key points: first, the effects we have seen are artifacts of interactions with other operating system modules that are typically not modeled in simulations and thus *cannot be observed in simulation model of the same application*. Second, while it is possible to model this behavior into simulators, it may not be advisable for two reasons: first, this observation is specific to SOS, other operating systems may have similar behavior but it may be manifested differently. Also, there might be implications due to other features of the OS that are not modeled. Our experience has been that emulation of the operating systems rather than attempting to model them is a better approach.

C. Fidelity through *sQualNet* models: Impact of Clock Skew

Several applications of sensor networks require synchronization among nodes. We consider one such application that calculates the line of bearing of a target by sensing the acoustic signal. The details of the algorithm and implementation is presented in [8], here we provide a brief overview relevant to our experiments. The target transmits an acoustic signal at time t_0 , which will be heard by different sensor

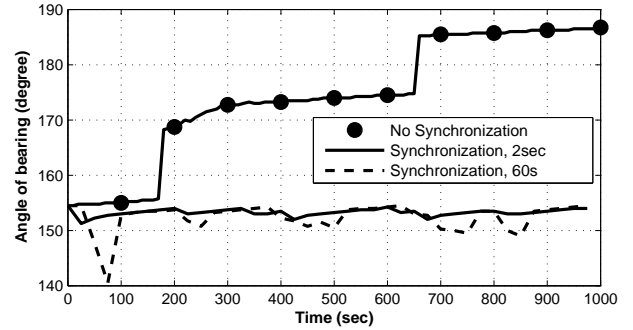


Fig. 7. Impact of clock drifts on the acoustic angle-of-bearing algorithm nodes in a cluster at t_i , for different sensor node i . These time instances will be slightly offset with each other due to difference in distance between the nodes and the target. These time-stamps are input to the algorithm that calculates offsets of time at which the signal was received at the different nodes which derives the angle of bearing of the target. For our purpose, it is important to note that if there is some skew in the clocks the time-stamps will be different and the algorithm will predict different, and possibly, incorrect results.

The preceding algorithm was ported to SOS which was configured to collect data from three sensor nodes. The mobility based sensor channel is used to trigger events at the sensors every 10 seconds. For the clock drift model, we randomly selected drift between 0 and 5 $\mu\text{s}/\text{sec}$ with no initial clock skew. We have also assumed a linear drift model as against the quadratic model to ease the analysis of the results. The algorithm was executed for two scenarios: when the nodes do not attempt to synchronize time among themselves and secondly, when they synchronize using the RATS [9] protocol. The RATS algorithm, which is part of the SOS distribution, offers various controls to fine tune its performance. For our experiments we consider the impact of synchronization interval. Fig 7 shows the angle of target that is computed by the algorithm as the time progresses for one sample set of simulation parameters. The target itself is immobile and is located at about 154° from the cluster.

Consider, first, the performance of the algorithm in absence of any synchronization protocol. Here we observe an interesting property of the algorithm – the accuracy is not linearly sensitive to the clock skew, instead there are discontinuities or “jumps”. Looking at this execution run it can be concluded that the nodes must not allow clock drifts to accumulate more than 200 seconds. Now let us examine the improvements that the RATS protocol can offer. We show in the graphs the results when the algorithms attempt to synchronize by sending beacons every 2 seconds and every minute. In both cases, since the clock skew is kept within bound, the predicted results do not deviate too much from the correct value. When the beacons are sent once per minute, the variance of the predicted angle is 2.64° ; this variance can be reduced to 0.64° but at the cost of sending thirty times more traffic. A tradeoff can be met between the accuracy desired from the algorithm and the amount of traffic in the network based on the application requirements.

The purpose of this case study and our investigations is to bring out few key points of the evaluation framework. First, simulation offers a very controlled environment. Multiple sets of experiments with different parameters can be executed with identical clock drifts which is difficult to achieve in physical testbeds. Second, this example has demonstrated an important observation, which is that the computation accuracy may not have linear relation with clock skew. Even though this application is highly sensitive to timing offsets, there may be other applications that need to be executed for very long durations

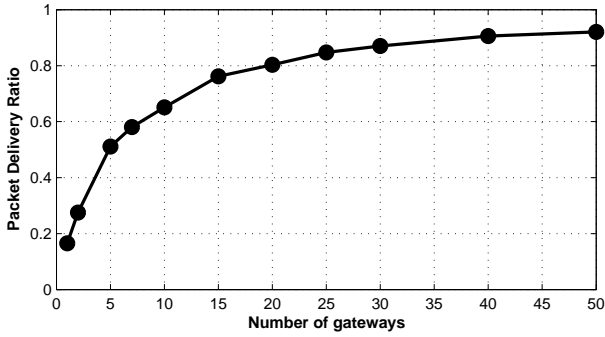


Fig. 8. Packet delivery ratio in the sensor subnet

to ensure that such discontinuities do not arise.

D. Modeling Heterogeneous Networks

The final case study demonstrates another advantage of the sQualNet framework – the ability to model diverse networks and scenarios consisting of heterogeneous components. We consider a two-tier sensor network. The lower layer is that of low end sensor nodes while the higher tier has more powerful nodes that run 802.11 radios and IP protocol. Some of the nodes in the second tier act as gateways or as a bridge between the two networks. The sensor nodes sense the environment and send reports to the nearest gateway. These gateway nodes transfer the information from the sensor network to IP network, where they do multi-hop routing to send the data to a node designated as the monitoring node.

This hybrid network is modeled by *emulating the sensor network* and *simulating the IP network*. The gateway nodes, which must communicate in both networks, are modeled as nodes with two interfaces (refer sec II-E). In our experiments, 1000 sensor nodes are emulated and 50 IP network nodes are simulated in a terrain of 700m × 700m. The sensor nodes run surge application and tree routing protocol that has been modified to create multiple trees in the network rooted at different gateways. The higher tier nodes are equipped with 802.11b radios, use AODV for multipath routing and UDP for end-to-end communication. We also built a custom application for these nodes. A gateway stores the packets received from the sensors and dispatches a message in the IP network only when it has accumulated K number of sensor packets, where K was varied between 1 and 20 in our experiments. We also vary the number of IP nodes that can act as gateways.

Fig 8 shows the packet delivery ratio in the sensor network subnet for different number of gateway nodes. As expected this ratio increases with higher number of gateways. More gateways imply there will be multiple trees which decreases the average number of

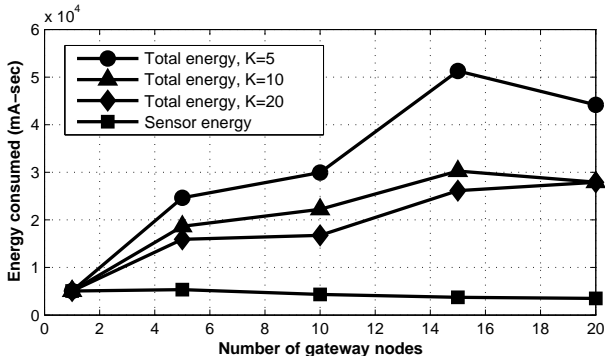


Fig. 9. Energy consumed in the network

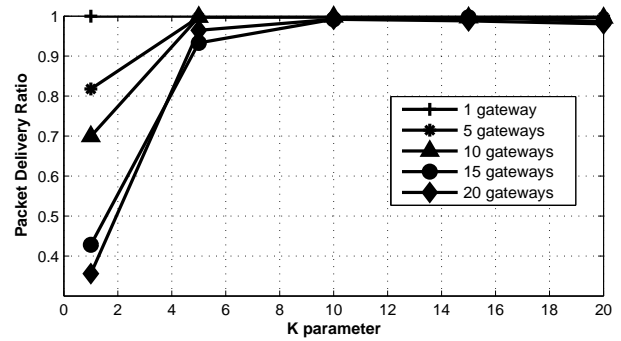


Fig. 10. Packet delivery ratio in the IP subnet

hops required to reach a gateway. Fewer hops, in turn, implies fewer transmissions in the network, hence a decreased probability of packet loss due to collision.

In the IP network, we consider the impact of the number of accumulated entries (the parameter K) in a packet. This has an affect on the packet size as well as the number of packets in the network. Fig 10 shows the ratio of packets received at the monitoring station as a function of the parameter K . Note that the graph only plots the PDR for the IP network. Note also that when only one gateway is used, the monitoring station is the gateway so all IP traffic is local and hence the PDR is 1. As the number of gateways increase, the PDR generally degrades due to the collision between simultaneous transmissions from the multiple gateways. Further, as the value of K increases, the PDR generally improves as it reduces the amount of IP traffic and hence the possibility of collisions. In particular, for $K = 1$ we find that lot of packets are lost in the network, particularly when there are many gateways in the network. The packet losses decrease as we increase the value of K or decrease the number of gateway nodes, as both have the effect of decreasing the traffic volume in the network.

Finally, fig 9 shows the energy consumed in the entire network as well as in the sensor subnet alone. Only radio transmission and reception energy is considered for purposes of energy consumption. The sensor radio has power consumption of 20mA and 18mA (Zigbee radio specifications) for transmission and reception, respectively. The equivalent figures for 802.11 radio were assigned as 300mA and 200mA. We find that almost all of the energy budget goes to the IP network. In sensor network the energy consumed decreases with larger number of gateways, again owing due to fewer transmissions in the network. However, this decrement is offset by the increase in the energy at the IP subnet.

This case study demonstrates how a part of the network can be evaluated via emulation while another with simulation. This aspect of sQualnet framework allows the software modeling fidelity for sensor networks via emulation as well as rich set of protocols and applications for other kinds of networks.

V. RELATED WORK

Existing efforts in sensor network evaluation can be broadly divided in three categories: network simulators enhanced with sensor models, sensor emulators enhanced with network models and instruction cycle level emulators. Earlier work in sensor simulation belonged to first category wherein sensor specific models were added to a popular network simulator. This design was attractive as does not require migrating away from the existing and familiar network simulators. SensorSim [10] extended the ns-2 [11] simulator with models of sensor channels, battery and power consumption. SWAN

[12] was extended to TOSSF [14] to emulate the TinyOS code. However, the approach taken was to translate the TinyOS syntax into C++ using either scripts or source-to-source compilers. This approach has a lot of room for introducing bugs in the source code in this translation phase. Our approach, on the other hand, is to run the sensor code-base without any modifications either to programming syntax, compiler or the binaries. This has been achieved by executing the sensor as a separate processing instance which interacts with the base simulator through a fixed set of APIs.

The most popular evaluation paradigm for sensor networks is sensor emulation with networking support. The sensor code is compiled into a desktop class of computer and the hardware API is abstracted with models that were added. TOSSIM [17] is an emulator for TinyOS for the mica motes. The emphasis of this tool is to be able to develop and debug the applications targeted for motes. Emstar [18] takes a similar approach but is targeted towards higher end sensor nodes such as iPAQs. EmTos [19] provides a hybrid framework for TOSSIM and EmStar simulations. This approach suffers from various limitations. All these emulators have their own implementation of the event scheduler which is typically not as efficient as the state-of-the-art scalable network simulators. Also, the models for various aspects of sensor network, in particular the wireless channel, are highly abstract and not accurate. This can have significant impact on the network evaluations as we have seen in the case studies. These emulators are also not able to emulate sensor software other than TinyOS or EmStar, that is, they do not offer flexibility in extending the frameworks for any OS. Also these tools are typically limited in representing heterogeneous architectures. For instance, to model the network described in case study in sec ref will require porting or rewriting the 802.11, IP, AODV etc into the framework. Even with these drawbacks, we consider that emulation is a very promising approach for sensor network evaluations and our work is an extension to these efforts by making the emulation framework more accurate, scalable and expressive.

The final class of evaluation methodology is emulation at the instruction cycle granularity. Avrora [20] and Atemu[21] are two examples of this approach. The code is cross-compiled for the sensor node architecture and is executed on an emulated processor. This provides the greatest measure of software modeling fidelity, but this approach is not suitable for large scale network evaluations. Emulating a processor is computation intensive and it becomes intractable simulating even a moderate size of network. However, it should be noted that the purpose of these simulators is different. They are more focussed on evaluating the correctness of implementation. We feel that these and the sQualNet framework are complimentary. The designer should first verify the correctness of a small scale network in these emulators and then move on to sQualNet for large scale evaluation with diverse conditions of channels, clock drifts etc.

VI. CONCLUSIONS AND FUTURE WORK

In this paper we have presented the design and implementation of sQualNet – a scalable and accurate evaluation framework for sensor networks. Our design philosophy was to enhance a network simulator with detailed and efficient models for sensor networks and provide a capability to emulate sensor node software. This design offers benefits in fidelity and scalability through three key features: first, using a network simulator allows us to use its detailed models of channel, radio, mobility etc and we can share the rich protocol suite for other kinds of networks to model heterogenous sensor networks. Second, through emulation we offer software modeling fidelity that is the ability to evaluate applications directly in the context of their

operating systems. Finally, we have enhanced the simulator with models for clock drift, sensing channel and battery usage. The models were optimized to make the simulations scalable.

We have also demonstrated through several case studies how these features can provide a comprehensive framework for accurate evaluation of sensor networks. Another aspect is flexibility or expressiveness in modeling networks. We demonstrated this with a hierarchical network with different network protocol stacks. Our design allows to emulate part of the network and simulate the rest.

The design approach that has been adopted in the development of sQualNet maintains extensibility to diverse sensor network Operating Systems. We are in the process of integrating TinyOS into this framework. Once this is done, we believe this will be a unique tool that can emulate both tinyos and sos in a single network. We are also extending the models to specifically represent actuation in sensor nodes.

REFERENCES

- [1] <http://www.scalable-networks.com>, “Qualnet.”
- [2] M. Takai, J. Martin, and R. Bagrodia, “Effects of wireless physical layer modeling in mobile ad-hoc networks,” *International Symposium on Mobile Ad Hoc Networking and Computing (Mobihoc)*, October 2001.
- [3] M. Varshney and R. Bagrodia, “Detailed models for sensor network simulations and their impact on network performance,” *Proceedings of ACM MSWIM*, October 2004.
- [4] S. Han, R. Rengaswamy, R. S. Shea, E. Kohler, and M. B. Srivastava, “A dynamic operating system for sensor nodes,” *International Conference on Mobile Systems, Applications and Services (Mobisys)*, 2005.
- [5] “Stochastic model estimation of network time variance,” *White Paper, Symmetricom*.
- [6] D. Rakhmatov and S. Vrudhula, “Energy management for battery-powered embedded systems,” *ACM Transactions on Embedded Computing systems*, 2002.
- [7] W. Ye, J. Heidemann, and D. Estrin, “An energy-efficient mac protocol for wireless sensor networks,” *Proceeding of Infocom*, June 2002.
- [8] V. Tsiatsis, R. Rengaswamy, and M. Srivastava, “Computation hierarchical for in-network processing,” *Mobile Networks and Applications*, pp. 505–518, 2005.
- [9] S. Ganerwal, D. Ganesan, H. Shim, V. Tsiatsis, and M. B. Srivastava, “Estimating clock uncertainty for efficient duty-cycling in sensor networks,” *ACM Conference on Sensor Networking Systems (SenSys)*, 2005.
- [10] S. Park, A. Savvides, and M. Srivastava, “Simulating networks of wireless sensors,” *Winter Simulation Conference*, 2001.
- [11] S. McCanne and S. Floyd, “Network simulator ns-2,” <http://www.isi.edu/nsnam/ns>.
- [12] J. Liu, D. Nicol, F. Perrone, M. Liljenstam, C. Elliot, and D. Pearson, “Simulation modeling of large-scale ad-hoc sensor networks,” *European Interoperability Workshop*, June 2001.
- [13] J.Liu and D. Nicol, “Dassf 3.1 user manual.”
- [14] L. F. Perrone and D. Nicol, “A scalable simulator for tinyos applications,” *Winter Simulation Conference*, 2002.
- [15] S. Sundresh, W. Kim, and G. Agha, “Sense: A sensor, environment and network simulator,” *37th annual Simulation Symposium*, April 2004.
- [16] G. Chen, J. Branch, M. J. Pflug, L. Zhu, and B. Szymanski, “Sense: A sensor network simulator,” *Advances in Pervasive Computing and Networking*, 2004.
- [17] P. Levis, N. Lee, M. Welsh, and D. Culler, “Tossim: Accurate and scalable simulation of entire tinyos applications,” *ACM Conference on Embedded Networked Sensor Systems (SenSys)*, November 2003.
- [18] L. Girod, J. Elson, A. Cerpa, T. Stathopoulos, N. Ramanathan, and D. Estrin, “Emstar: a software environment for developing and deploying wireless sensor networks,” *Proceedings of USENIX General Track*, 2004.
- [19] L. Girod, T. Stathopoulos, N. Ramanathan, J. Elson, D. Estrin, E. Osterweil, and T. Schoellhammer, “A system for simulation, emulation, and deployment of heterogeneous sensor networks,” *ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2004.
- [20] B. Titzer, D. Lee, and J. Palsberg, “Avrora: Scalable sensor network simulation with precise timing,” *In Proceedings of IPSN’05*, 2005.
- [21] J. Polley, D. Blazakis, J. McGee, D. Rusk, J. S. Baras, and M. Karir, “Atemu: A fine-grained sensor network simulator,” *Conference on Sensor and Ad Hoc Communications and Networks (SECON)*, 2004.