ATEMU: A Fine-grained Sensor Network Simulator

Jonathan Polley, Dionysys Blazakis, Jonathan McGee, Dan Rusk, John S. Baras
Center for Satellite and Hybrid Communication Networks
Department of Electrical and Computer Engineering & Institute for Systems Engineering
University of Maryland, College Park, MD 20742, USA
{jpolley, dblaze, mcgee, ruskd, baras}@umd.edu

Manish Karir
Networking Research and Development
Merit Network Inc. Ann Arbor, MI 48105
mkarir@merit.edu

***Abstract*— In this paper we describe the design and implementation of ATEMU, a fine grained sensor network simulator. ATEMU is intended to bridge the gap between actual sensor network deployments and sensor network simulations. We adopt a hybrid strategy, where the operation of individual sensor nodes is *emulated* in an instruction by instruction manner, and their interactions with each other via wireless transmissions are *simulated* in a realistic manner. A unique feature of ATEMU is its ability to simulate a *heterogeneous* sensor network. Using ATEMU it is possible to not only accurately simulate the operation of different application on the MICA2 platform but also a complete sensor network where the sensor nodes themselves maybe based on different hardware platforms. In addition we also describe our implementation of XATDB, our front-end debugger/GUI for ATEMU. XATDB provides an excellent educational tool for people to start learning about the operation of sensor nodes and sensor networks, without requiring the purchase of actual sensor node hardware. The accuracy and emulation capabilities provided by ATEMU ensure that when and if actual hardware is used, the software will already have undergone rigorous testing and debugging on an accurate platform. This would provide the sensor network deployment community with a much more accurate estimate of the performance of various algorithms and protocols in realistic scenarios and platforms.**

I. Introduction

One of primary challenges facing the research community in the area of sensor networks is translating theoretical research into actual deployable protocols and systems. Traditionally, simulation has been the means of achieving this goal. Various strategies have been proposed and implemented for simulating sensor networks. However, no single tool has been able to achieve the goal of providing accurate results that can be translated directly into guidelines of how an actual live sensor network might operate. While some current approaches abstract too many important details out of the simulation framework [1], others are not flexible enough to allow for experiments with heterogeneous nodes [2]. ATEMU (ATmel EMUlator) is our attempt at providing a crucial missing piece in making sensor networks a reality.

ATEMU provides low-level emulation of the operation of each individual sensor node. It emulates the operation of the various components on a sensor node, such as the processor, timers, and the radio interface. These emulations of individual sensor nodes are then tied together via their interactions with each other to form an emulation of an entire sensor network. In order to achieve this goal ATEMU provides an extensible model of the "air" to simulate operation of the wireless medium. Although, in its current form ATEMU only contains support for the MICA2 hardware platform, the architecture is general enough to allow for other hardware platforms to be easily supported. The emulation approach towards simulating sensor networks leads to results that are much more detailed as the actual hardware platforms themselves are being modeled and not simply some abstraction of the platforms.

Our primary goal is to provide the sensor network research community with a scalable and extremely high fidelity platform, that can be used as a pre-deployment tool for sensor networks. Even a small scale deployment is difficult to manage unless the software has been extensively tested, as the logistics of gathering, reprogramming and then re-deploying even a 30 node network are difficult and time consuming. With ATEMU, the goal is to delay using the actual hardware platforms till the last step, and still be able to maintain some level of confidence regarding the operation of a live network.

A secondary goal of ATEMU is to provide an efficient front-end that will make it easier to debug various sensor networking applications, algorithms, as well as embedded operating systems such as TinyOS. We have implemented XATDB as a graphical front-end to ATEMU to provide exactly this functionality. Using XATDB we can monitor the operation of an individual sensor node instruction by instruction. We have the ability to single step through either assembly instructions or at the C instruction level. In addition, XATDB also provides the ability to specify breakpoints and watchpoints to aid in the debugging process.

However, we note that such high fidelity emulation comes at the expense of high processing requirements. While we do

0-7803-8796-1/04/$20.00 (c) 2004 IEEE.

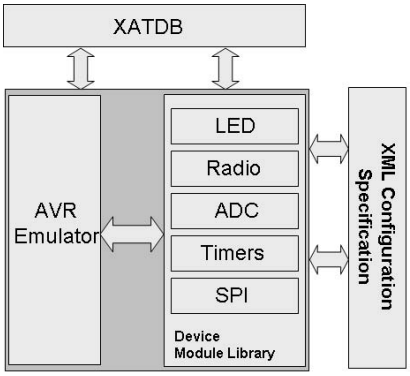

Fig. 1. ATEMU Components Architecture

attempt to optimize the emulation environment to extract the best performance possible, we are limited by our design goal of providing an extremely high level of detail which is required in order to provide high-fidelity and accurate results. We argue that the higher quality of the results from our emulator and the increased confidence users can have in them are worth the extra computational costs. We are actively investigating methods to optimize the performance of the emulator even further on highly parallel cluster computing systems which are becoming increasing popular.

Finally, it should be noted that ATEMU is not simply a simulator for TinyOS specific applications. ATEMU models the operation of the hardware platform, and not simply the software, it can even be used to develop alternate operating systems for these sensor platforms. Currently it seamlessly runs TinyOS and most applications built for the highly popular MICA2 sensor nodes developed at UC Berkeley.

ATEMU has been publically released via our website [3] and is actively being used by the sensor networking research community. We are engaging the sensor network research community in the further development of ATEMU via a public mailing list as well as a public bug reporting and tracking website. Already we have received valuable bug fixes and patches from the community that have been incorporated into the newer releases.

The rest of this paper is organized as follows: Section 2 provides an overview of the software architecture of ATEMU, section 3 provides a description of XATDB, our debugger frontend to ATEMU, Section 4 provides a description of some of the validation experiments and simulations we have run to verify the correct operation of ATEMU, and finally section 5 provides some conclusions and outlines our future development plans.

II. ATEMU

ATEMU at its core is a software emulator for AVR processor based systems such as the MICA2. Along with support for the AVR processor, it also includes support for other peripheral devices on the MICA2 sensor node platform such as the radio. ATEMU can be used to perform high fidelity large scale sensor network emulation studies in a controlled environment. In addition, the ATEMU package can also be used in an educational environment to facilitate experimentation with sensor networks, without requiring the purchase of expensive sensor node hardware. It offers a solution around the logistical difficulties of conducting experiments with large numbers of physical devices. As ATEMU is binary compatible with the MICA2 hardware, it can be directly used by developers of TinyOS related software. Although the current version only includes support for MICA2 hardware, it can be easily extended to include other sensor node platforms. It allows for the use of heterogeneous sensor nodes in the same sensor network. The ATEMU distribution consists of two components: the ATEMU emulator core, and the XATDB graphical debugger. The architecture of the various software components of ATEMU are shown in Figure 1.

The ATEMU emulator core can simulate arbitrary numbers of nodes each of which may be configured to run a different sensor networking application. It can model their execution and the interactions between them, such as radio communications in extremely fine detail. It offers nearly complete emulation of the MICA2 hardware platform and as a result provides results that are closer to real life operation of a distributed sensor network. Currently only d-squared propagation is modeled, however this will be improved with future versions. ATEMU uses the same binary that is loaded onto the MICA2 node and uses its AVR processor emulation engine to model the execution of the code on each sensor node in step with each other. As very few details of the actual operation of a sensor node are abstracted out, it provides an excellent platform to perform unbiased comparisons of various sensor networking protocols and the results are significantly more realistic than other simulators.

In the following sections we describe the software architecture and design of ATEMU. This can help people understand the underlying philosophy behind its operation. In particular it is of use not only to people who wish to use it for their experiments but is also useful for developers who wish to extend ATEMU.

A. CPU Emulation

The most important part of ATEMU is its AVR CPU emulation core. Each AVR instruction is decoded and executed according to the specifications in Atmel's instruction set documentation [4].

While the instruction set remains the same, different hardware platforms might choose to use different versions of Atmel's CPU's. We allow the user to specify such variables on a per node basis via a config file. These include the sizes of the SRAM and flash, size of the program counter, and the symbolic names for all of the IO peripheral devices that have been attached to the CPU on the hardware board. This allows us to emulate sensor networks where different nodes have different hardware platforms and peripheral devices.

B. Hardware Emulation

A sensor node hardware platform consists of not only the CPU, but various other peripheral devices that are attached to it. In keeping with this design, ATEMU provides various device modules that can be attached to the CPU to completely specify a particular sensor node platform. For example the MICA2 sensor node platform includes the the Atmel ATmega 128L CPU,

```
<?xml version="1.0" encoding="utf-8"?>
<configuration title="Common MICA2 features"
       xmlns="http://www.cshcn.umd.edu/research/atemu/">
    <!-- Include the AVR Atmel128L processor description -->
    <include href="atmel128">
        <!-- Frequency of main system clock -->
        <param name="oscillator" value="7.3728MHz" />
        <!-- Frequency of TIMER0's asynchronous clock -->
        <param name="asynccrystal" value="32kHz" />
    </include>

    <!-- Include the LEDs -->
    <message text="LED: Red Green Yellow" />
    <device id="Red" module="led">
        <register name="direction" ref="DDRA" bits="0x04" />
        <register name="port" ref="PORTA" bits="0x04" />
        <register name="pin" ref="PINA" bits="0x04" />
    </device>
    <device id="Green" module="led">
        <register name="direction" ref="DDRA" bits="0x02" />
        <register name="port" ref="PORTA" bits="0x02" />
        <register name="pin" ref="PINA" bits="0x02" />
    </device>
    <device id="Yellow" module="led">
        <register name="direction" ref="DDRA" bits="0x01" />
        <register name="port" ref="PORTA" bits="0x01" />
        <register name="pin" ref="PINA" bits="0x01" />
    </device>

    <!-- The Radio -->
    <device id="radio" module="radio">
        <!-- Connect the normal registers -->
        <register name="ddrd" ref="DDRD" bits="0xD0" />
        <register name="portd" ref="PORTD" bits="0xD0" />
        <register name="pind" ref="PIND" bits="0xD0" />
        <!-- Connect with the other devices -->
        <!-- NOTE: Consider this interface deprecated as it may change once
          - XCONF is completed -->
        <connect name="spi" ref="spi" />
        <connect name="adc" ref="adc" />
        <!-- Parameters, will be deprecated once XRUN is completed -->
        <!-- Standard deviation of the noise to add to the RSSI reading.
          - TinyOS will halt communications if it is not noisy. -->
        <param name="rssinoise" value="0.1" />
        <!-- Noise level for calculating bit error rate in dBm -->
        <param name="radionoise" value="-117" />
        <!-- Constant attenuation to tack onto radio transimissions to correct
          - range problems.  In dBm. -->
        <param name="attenuate" value="-37" />
    </device>

</configuration>
```

Fig. 2. MICA2 Hardware Specification

as well as a CC1000 radio chip, three LEDs, ADC, EEPROM, SPI, Timers, and external sensor boards. In some cases the devices might be physically present on the CPU or the radio chips but as they are logically different, we have implemented them separately. ATEMU provides implementations of all peripheral devices needed for the emulation of the MICA2 in the form of plug-in libraries, and are loaded at run time by the emulator.

A MICA2 hardware node model is completely specified by including a directive to link to the appropriate CPU definition followed by a series of device definition sections. Each device section contains all the register definitions that define how that device communicates with the CPU. Figure 2 shows specific details of how the MICA2 device is specified using ATEMU. One can easily see that it is trivial to emulate for example a MICA2 that could have 4 LEDs. Though that is a trivial change it demonstrates that power of using the emulation approach. Given that the CPU we are attaching to can support it it is even possible to build in software a MICA2 platform that can have 2 different radios, but attaching 2 different CC1000 chips to the CPU. In this way ATEMU demonstrates its flexibility by encouraging experimentation with newer hardware platforms.

1) Radio Emulation: Aside from the CPU the radio device is the most critical component on a sensor node platform. In this section we describe some details of our current implementation of the radio interface. In ATEMU, the radio back-end is composed of three primary components: an emulator of the CC1000 chip, an interface between the Atmel processor and the CC1000, and a radio propagation model for communications between different motes. Dividing up the wireless interface in this way allows us the flexibility in the future to implement radio interfaces based on other radio chips. For example our emulation of the CC1000 could be replaced with an emulation model of the CC2420, while keeping the interface with the Atmel CPU and the radio propagation models the same. However, for the rest of this section we will focus on the radio interface implemented by the MICA2 architecture.

Each sensor node which has a CC1000 is responsible for taking a bit stream from the chip, FSK modulating it to a software-controlled frequency, and sending it onto the air. Packet reception is the inverse of this process. The software control of this chip may be a strength in hardware design and systems integration, but makes it difficult to effectively emulate. As the software is free to utilize an near-infinite number of frequencies for communication, simply modeling the air as a bit stream would not allow the user to test software which uses these features of the CC1000, such as algorithms that make use of multiple bands. This flexibility of the CC1000 makes the task of emulating its behavior that much more difficult.

In order to simplify this process we implement each mote as a receiver/transmitter pair. The transmitter is modeled as having a single fundamental frequency and associated power. The receiver's tuner has a band for each bit value (zero or one), and when receiving, each band tallies the power from all transmitters after being attenuated by distance in space and frequency (-10dB/600kHz). The difference between the two powers is applied to the error function:

$$P(P_0, P_1) \quad = \quad \frac{1}{2} e^{\frac{1}{4} \frac{|P_1 - P_0|}{\text{Noise}}}$$

where P_0 and P_1 are the power of the zero and one bands and the final resultant bit is given to the CC1000 emulator for transfer to the CPU as the received bit value.

In the current implementation the ATEMU simulation of the behavior of air is based on the standard d-squared radio propagation model:

$$P_R \quad = \quad \frac{P_T G_R G_T \lambda^2}{(4\pi)^2 d^2}$$

Where P_R is the received power, P_T is the transmitted power, G_R and G_T are the receive and transmit gains, and d is the distance between the receiver and the transmitter. For simplicity, both transmit and receive antenna gains are assumed to be 1.0 and λ is calculated as the wavelength in vacuum. We apply a constant coefficient to the expression above to bring the maximum radio range in line with real-world observed values.

As each node can potentially be interfered with by all other nodes in a network, keeping track of received power turns into a highly inefficient n-squared algorithm. In the current version of ATEMU we make the following attempts at optimizing these calculations.

The first optimization is to cache the computed values of transmitted power. Each time a change is made to a transmitter object, a global timestamp is incremented. Each receiver object, by comparing its cache timestamp to the global timestamp, can determine if any change has occurred in air communications. Each transmitter object also contains the time it was

last modified. By iterating through the cache, each receiver can determine exactly which transmitters have changed and only recalculate them. So if a given transmitter hasn't shifted to a new frequency or power, the previously calculated value can be used when determining the next bit to receive.

The second optimization we have implemented is to cache the various attenuation coefficients. In a given transmission during emulation, it is likely that only the frequency and transmit power will change. For this, the power formula breaks down into:

$$P_R = \frac{P_T G_R G_T c^2}{(4\pi)^2 d^2 f^2} = (P_T)\left(\frac{G_R G_T c^2}{(4\pi)^2 d^2}\right)\left(\frac{1}{f^2}\right)$$

The second term has been turned into a constant, so it can be cached and only needs to be recalculated if a node moves.

However, the optimizations described above do nothing to change the inherent n-squared scaling of the power computation algorithm. This is not a problem with the approach taken by ATEMU alone, ns-2 has also been shown to suffer from the same problem [5]. In [5] a new method was proposed to improve the scalability of modeling wireless communication between nodes by limiting how transmitters affect receivers by using the fact that in reality a transmission beyond a certain range has attenuated so much that it can be ignored by receivers beyond that range. We are currently investigating incorporating this modification into ATEMU.

C. XML Configuration Specification File

There is currently a lack of any common method of specifying the various configuration parameters of a sensor network for a simulation/emulation tool. One of the primary contributions of ATEMU is to provide a simulator agnostic method of specifying the configuration specifications of a sensor network. XML has emerged as a widely accepted method of specifying common interfaces among different tools. Therefore, we use XML to define a simulator agnostic configuration specification of a sensor network. Using this basic specification, one can then generate various other input formats required by other simulation tools. This can provide a common base for researchers in sensor networks who can define their network scenarios independently of the choice of network simulator.

Though our work is still in its infancy we have already been able to achieve significant benefits from this approach in terms of its flexibility and logical structure. Figure 3 shows how XML can be used to specify various parameters for a simple two-node network in ATEMU. A unified input file specification format would be of great use in comparing and contrasting the performance of various simulation platforms. In the following description we focus on how the various XML tags are used to specify some very basic attributes of a sensor network. ATEMU directly uses the XML configuration specification to obtain the values of various global and local parameters.

For small topologies the XML input file can be generated by hand, but for larger topologies scripts or tools can be written to automate the process. For ATEMU, a minimum configuration specification input file would need to specify the hardware configuration, software executable binary image, and the physical location for each node.

```
<?xml version="1.0" encoding="utf-8" ?>
<motelist
            title="Minimal example of run file operation"
            xmlns="http://www.cshcn.umd.edu/research/atemu/">
        <mote id="rfmcounter" model="mica2">
            <flash href="apps/CntToRfm/main.exe" />
            <param name="position" value="0,0,0" />
        </mote>
        <mote id="rfmblinker" model="mica2">
            <flash href="apps/RfmToLeds/main.exe" />
            <param name="position" value="5,0,0" />
        </mote>
</motelist>
```

Fig. 3. An Example XML Configuration Specification File for a 2 Node Sensor Network

The syntax of a valid configuration file is relatively simple. After the xml header line, the file must contain a motelist tag. This element will contain the list of motes to be run. This is used to describe some attributes that are global and apply to the entire sensor network. Some of the important ones that are currently defined are:

- *title* - a descriptive title of the runfile
- *xmlns* - defines the xml namespace

This is followed by the *motelist* tag which contains a description of each individual sensor node in our sensor network. For each node we can define several specific attributes. The ones listed below are required for a valid config:

- *id* - a unique name for this node
- *model* - specifies the sensor node hardware, in ATEMU this is used to specify the hardware specific emulation core that is used to emulate the operation of this node, such as the MICA2

Each sensor node element may take a few sub-elements. Some important ones are:

- *flash* - contains an href to the program to load
- *param* - contains the name and value of a parameter to set for the device. In our example in Figure 3, we use it to set the position for the radio.

One of the more important valid tags that we are implementing next is the *event* tag. This tag can be used to specify events that occur as the emulation/simulation progresses. Using the event tag we could associate an action with a time, for example we would be able to specify that at time = 30 seconds, a specific sensor node should be powered off. This can also be used to simulate other artificial conditions in a sensor network emulator, such as at time = 50 seconds, the value being received from a specific sensor node's light sensor suddenly becomes much larger.

Using the XML configuration specification format allows us a lot of flexibility in improving the usability of ATEMU. We are currently still in the process of defining and implementing this feature in more detail.

D. The Operation of ATEMU

The basic flow of control of ATEMU is fairly straightforward. After loading and initializing all of the simulated nodes and their components, each of the nodes is advanced by one cycle, executing an instruction if appropriate, interrupts are polled, and clocked devices receive a tick.

Using pseudo-code the high-level operation of ATEMU can be represented as follows:

```
for each node
    if node running and not sleeping
        step cycle
    if interrupt
        PC = interrupt vector address
    for each device
        tick device clock
```

This loop continues until either the user interrupts it, or some control structure such as a breakpoint or watch-point causes the simulation to halt.

III. XATDB

Included in the ATEMU distribution is XATDB, XATDB is a graphical front-end to the ATEMU sensor network emulator. XATDB provides users a complete system for debugging and monitoring the execution of their code. Using XATDB, users can run code built for the MICA2 platform, and debug efficiently using the ability to set breakpoints, watchpoints, as well the ability to single step through either assembly or high level C code. XATDB is particularly powerful in its ability to provide a debugging interface to multiple nodes in a sensor network.

A. Symbolic Debug

XATDB derives its symbolic debugging capabilities from Stabs [6][7]. Stabs is a format developed at the University of California at Berkeley, that that can be used to describe a program to a debugger. When the -g option is used with GCC, additional debugging information is carried from the compilation process into the final executable. This debugging information describes features of the original source file, such as line numbers, the types and scopes of variables, function names, and parameters.

XATDB differs from traditional debuggers like gdb, in that it executes the sensor node binary(s), on the ATEMU emulator and not on the host computer. In terms of functionality however, XATDB duplicates most of the features of gdb. In addition to the core functionality of being able to run, debug and watch a binary execution on the ATEMU emulator, XATDB also has the capability to monitor the execution of several different emulations.

During initialization XATDB reads in the sensor network configuration file from a XML based sensor network description specification file such as the one shown in Figure Figure 3. It passes this information in an appropriate form to ATEMU which in turn initializes separate emulation state machines for each sensor node that is specified in the description file. In the user interface, a separate tab is created for each sensor node, which displays the assembly or C-level source code of the binary that is being run on that node. Figure 4 shows an example of a 6 node sensor network emulation in progress. The window on the left displays a separate tab for each emulated sensor node. When a particular sensor node tab is selected, the window in the right displays registers, stack information, and local and global variables and their values. The window on the bottom displays various events, debugging information, breakpoints,

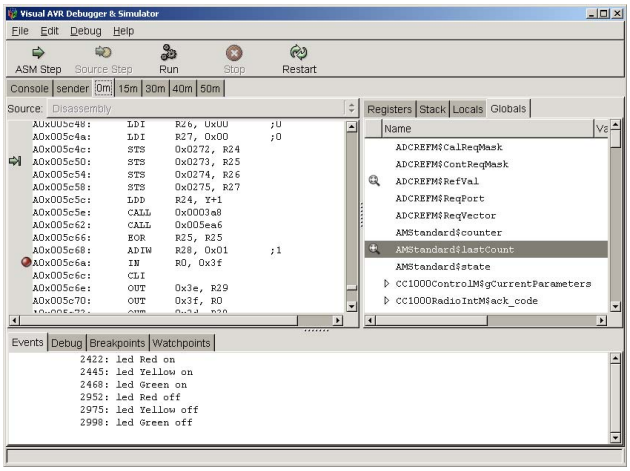

Fig. 4. XATDB: Example of a 6-node sensor network

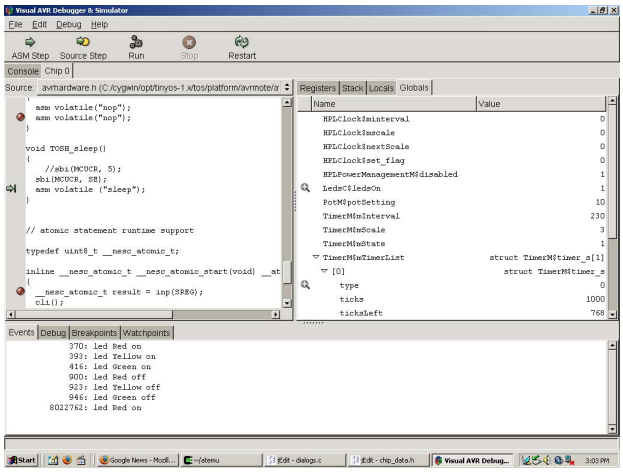

Fig. 5. XATDB: Example of C-level Debugging

and watchpoints. It is possible to examine and step through either assembly level code or at the C-level. Figure 5 shows an example where XATDB is being used to single step through C-level code. The red dots indicate where breakpoints have been set, and the window on the right is showing various global variable data structures, and their values. The screen at the bottom is displaying the status of the LEDS on the emulated sensor node.

IV. Validation and Experiments

In order to verify the proper operation of ATEMU and to demonstrate its capabilities we ran a series of simple experiments. In order to be able to validate the operation of the emulator, we have to use it initally in simple scenarios where the results are intuitive and well understood. This also provides us with a usage scenario from which we can extract the various operational parameters of the emulator such as how it scales in terms of memory and cpu usage.

A. Emulating a Heterogeneous 2-node Sensor Network

After performing basic testing and validation tests on ATEMU consisting of simple well known TinyOS applications

such as Blink, and CntToLeds, we proceeded to run a test that would demonstrate ATEMU's ability to emulate a *heterogeneous* sensor network. The simple 2-node network consisted of one node running the CntToRfm application, and the second node running the RfmToLeds application. These are both well known TinyOS test applications. The first simply transmits an increasing counter value over the radio interface, and the second application listens on the radio interface and displays the counter values it receives on the LEDs.

We started by first separately compiling both application binaries as if they were going to be used on actual MICA2 hardware. Next we use the sample XML configuration specification file shown in Figure 3 to specify our 2-node sensor network. This configuration file designates one node as the node that will run CntToRfm and the other node as the node that will run RfmToLeds. We then proceed to load this XML configuration file into XATDB. XATDB displays a "Finished Loading" message after ATEMU has initialized the nodes. Now we are able to run our sensor network by simply pressing the "Run" button. The console messages in the bottom window display the status of the LEDs and by observing the changing values we are able to verify the correct operation of our emulated network. We are also able to stop the running emulation, inspect the source code for either one of the running applications, set breakpoints, single-step through the execution of the emulated sensor network and examine any register values, global and local variables and their values.

Though simple in nature this experiment allows us to demonstrate some of the basic features and capabilities of ATEMU.

B. Evaluating a Simple MAC Protocol

The first protocol we decided to study was an example of a very simple collision avoidance scheme. The experimental scenario consists of a single master node surrounded by various numbers of transmitter sensor nodes. We want to study a sensor network where the master node sends out periodic beacons to all the transmitters, which respond by sending back a reply. We study the behavior of two different MAC protocols in this scenario. In the first protocol, the transmitter nodes all attempt to transmit a single packet back to the master node at the same time, in the second, the transmitters pause for a random amount of time (between a min and a max) before they attempt to transmit.

We first generate two different types of application binaries for this project. The first binary image will be run on the master node and is responsible for sending out the beacons. The second binary image will be installed on the transmitter nodes and is responsible for replying to the beacons transmitted by the master. In order to run this type of an experiment using TOSSIM [2] both these functionalities would need to be combined into a single application then nodes would select which subset of the functionality they are required to run.

Intuitively, from the description of the two protocols, we would expect the jittered transmit protocol to have a much lower number of retransmission attempts, as the transmissions are staggered and therefore there are fewer collisions. However just how big a difference does it make is still an important question to answer.

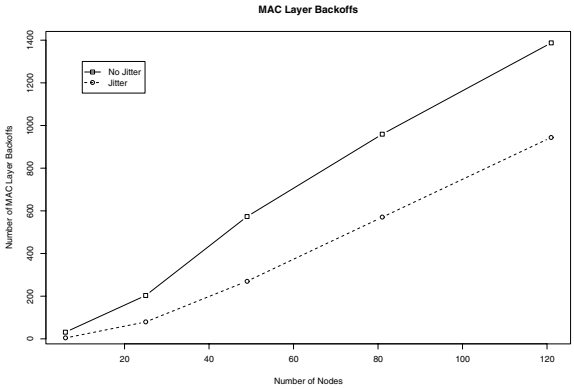

Fig. 6. Number of Backoffs

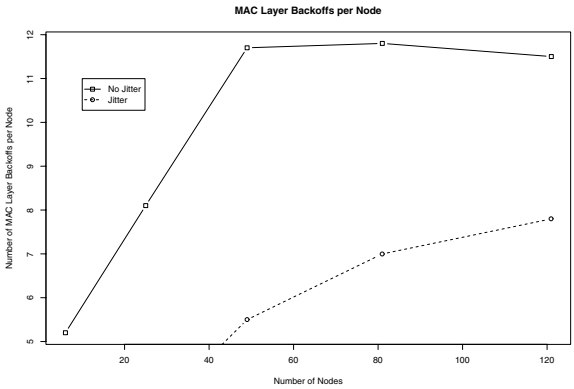

Fig. 7. Number of Backoffs per Node

Figure 6 shows the total number of backoffs reported by ATEMU as the number of sensor nodes increases. As expected the staggered transmit protocol outperforms the simpler direct transmission approach. Also the performance gap increases as the number of nodes increases. Figure 7 shows how the average number of backoffs per node varies as the number of nodes is increased.

Figure Figure 8 shows the memory used by ATEMU to emulate our topology with different number of nodes. As expected, due to the detailed emulation of each individual node memory consumption is linear with respect to the number of nodes in a sensor network.

Figure 9 shows the number of real-time seconds it takes per second of simulated time as a function of the number of nodes in a sensor network. The graph shows that the time it takes to emulate a single second of a sensor network in ATEMU increases exponentially with respect to the number of nodes that are present in the network. This behavior is largely due to the n-squared nature of the emulation of the wireless component of a sensor node. Each transmission has the potential to interfere with every other node in the network, and therefore we end up with a n-squared algorithm to keep track of transmissions and received power at each node. Recently there has been some interesting work in addressing the same problem in ns-2 [5]. Our next step will be to attempt to apply the simple approach of limiting the influence of transmissions to only nodes that are within

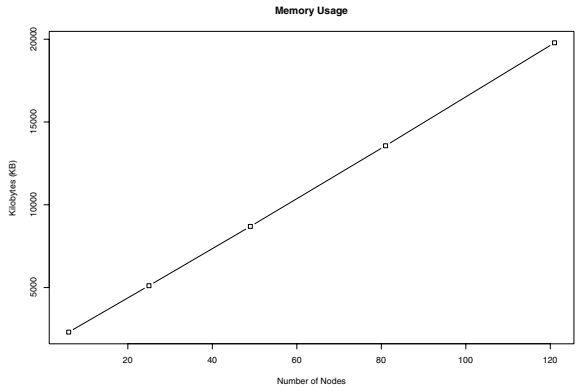

Fig. 8. Memory Utilization

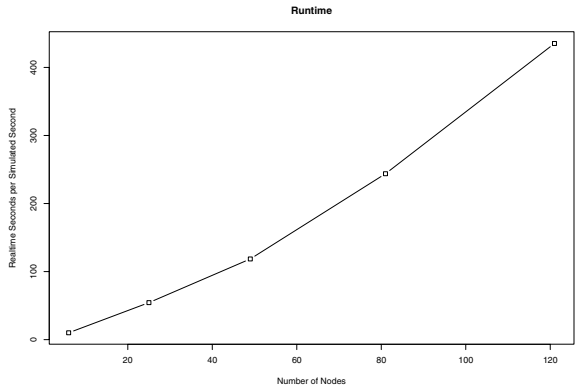

Fig. 9. Runtime

a certain range of each transmitter to ATEMU.

V. Related Work

While there has been quite a significant amount of work in the general area of sensor networks, there has been substantially less work in the area of simulation tools for studying them. Here we describe some of the most relevant work in an effort to demonstrate and motivate the need for its development.

Modifications to ns-2 [1] have been proposed in order to make it suitable for modeling sensor networks. However, ns-2 operates at a much coarse level of detail, and therefore only provides results that are inadequate for some studies. In particular it lacks the ability to accurately model specific hardware details that might have a significant impact on performance metrics such as power. This coarse grained nature of ns-2 is what make it flexible enough for all types of studies ranging from sensor networks to satellite networks to MPLS and fiber optic networks.

TOSSIM [2] has been proposed as a method to simulate the operation of TinyOS based sensor networks. However, TOSSIM has several limitations that restrict its use to wider scenarios. In particular TOSSIM by virtue of its design is tied to TinyOS, and therefore cannot be used to perform studies where an alternate application is being used. TOSSIM is limited by only being able to run a single binary image on all sensor nodes. This does not allow users the flexibility they need in order to be able to study the design and performance of *heterogeneous* sensor networks.

The emulation approach itself is not new. Embra [8] was developed as an emulator for the MIPS R3000/R4000 processor. However, we believe ours is the first attempt to emulate multiple nodes in a network, as well as provide an extremely useful debugging tool to aid in the development of embedded software for the MICA2 platform. Recently, EmStar has been proposed as a means of studying wireless sensor networks that incorporate both motes as well as iPAQ based microservers [9]. However, this too fails to provide the level of detail provided by ATEMU. The SensorSim project which provided a simulation framework for sensor networks similar to ns-2 has been discontinued is no longer available to researchers [10].

In terms of features that they provide, both ns-2 and TOSSIM can be extremely useful. Sensor network developers and researchers would simply pick the right tools depending on the level of detail which was required. ATEMU serves to complete the sensor network research toolchest by providing an extremely high-fidelity *emulation* environment of sensor node hardware. In its current implementation it is able to emulate the operation of the MICA2 sensor node hardware platform, including the processor, radio interface, as well as other peripheral devices. ATEMU is particularly useful in studies that require an extremely high level of detail, such as those which would like to monitor power consumption, or those that involve heterogeneous (both in terms of software as well as in terms of hardware) nodes in the same network. Due to its emulation approach it is much closer in operation to actual hardware sensor nodes.

VI. Conclusions and Future Work

In this paper we have described the design and implementation of ATEMU, a sensor network emulation platform. ATEMU differs from other existing simulators in that it has the ability to perform extremely low-level emulation of the sensor node hardware. Though the current version includes support for the MICA2 sensor node, ATEMU can be easily extended to include support for other hardware platforms as well. The low level emulation capabilities of ATEMU make it an ideal complement to the existing set of simulation tools. Scenarios that require extremely high-fidelity results even at the expense of longer simulation times can benefit immensely from ATEMU. As ATEMU emulates the hardware of the MICA2 including the processor, radio interface, ADC, LEDs, and other peripheral devices, it is possible to run the same application binary image that is run on actual hardware in the emulator further eliminating any simulation artifacts that running a different binary format might introduce.

The ATEMU platform includes XATDB, a debugger front-end which provides an excellent learning and debugging tool for large sensor networks. Using XATDB, it is possible to single step through code, set breakpoints and watchpoints, examine data structures and variables, and observe the status of peripheral devices.

The ATEMU platform relies on the use of XML based configuration specification files. Using XML, we hope to be able to develop a common sensor network definition specification

framework that can be used to specify the various parameters of a sensor network in a simulation tool agnostic format. This specification format can then either be used directly as input into to various simulation tools, or can be converted into simulator specific configuration files.

ATEMU has been publically released and can be downloaded from our website [3]. A public mailing list and bug tracking system are also provided to encourage feedback from the sensor network research community. Bug fixes and suggestions from the community are being taken into account and incorporated into the newer releases. The ATEMU platform provides valuable functionality to both the sensor network research as well as the sensor network deployment communities.

ACKNOWLEDGEMENT

This paper was funded and prepared through collaborative participation in the Communications and Networks Consortium sponsored by the U. S. Army Research Laboratory under the Collaborative Technology Alliance Program, Cooperative Agreement DAAD19-01-2-0011; by the Space and Naval Warfare Systems Center - San Diego under contract number N66001-00-C-8063; and by NASA under award number NCC8235. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory, the U. S. Government, the National Aeronautics and Space Administration, or the Space and Naval Warfare Systems Center.

REFERENCES

[1] I. Downard. Simulating Sensor Networks in NS-2. http://nrlsensorsim.pf.itd.nrl.navy.mil.

[2] P. Levis, N. Lee, M. Welsh, and D. Culler. Tossim: Accurate and scalable simulation of entire tinyos applications. *Proceedings of the First ACM Conference on Embedded Networked Sensor Systems(SenSys 2003)*, November 2003.

[3] ATEMU. Sensor Network Emulator/Simulator/Debugger . http://www.isr.umd.edu/CSHCN/research/atemu.

[4] ATMEL. 8-bit AVR Instruction Set. http://www.atmel.com/.

[5] V. Naoumov and T. Gross. Simulation of large ad hoc networks. *Proceedings of the Sixth ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, September 2003.

[6] Julia Menapace, Jim Kingdon, and David MacKenzie. The stabs debug format . http://citeseer.ist.psu.edu/342295.html.

[7] P.B. Kessler. Fast breakpoints. *Proceedings of the Conference on Programming Languate Design and Implementation*, pages 78–84, 1990.

[8] E. Witchel and Rosenblum M. Embra: Fast and flexible machine simulation. *Proceedings of ACM Conference on Measurement and Modeling of Computer Systems*, 1996.

[9] L. Girod, Elson J., Cerpa A., Stathopoulos T., Ramanathan N., and Estrin D. Em*: a sofware environment for developing and deploying wireless sensor networks. *Proceedings of USENIX*, 2004.

[10] S. Park, Savvides A., and Srivastava M. Sensorsim: A simulation framework for sensor networks. *Proceedings of MSWiM*, Aug 2000.