MAC Layer Abstraction for Simulation Scalability Improvements in Large-scale Sensor Networks [1]

Tian He[†], Brain Blum[*], Yvan Pointurier[*], Chenyang Lu[‡], John A. Stankovic[*], Sang Son[*]
[*]Department of Computer Science, University of Virginia
[†]Department of Computer Science and Engineering, University of Minnesota
[‡]Department of Computer Science and Engineering, Washington University in St. Louis

***Abstract*— It is risky to implement a large-scale wireless sensor system without predicating its behavior beforehand through simulation, an indispensable step toward the final system. Since in wireless simulation, over 90% simulation time is spent on the characterization of the MAC layer behaviors, abstraction at the MAC layer becomes the key to the scalability of wireless sensor network simulations. In this work, we employ an abstract simulation mode, established by online data collection, to describe the MAC behaviors efficiently. Observing that the abstract simulation mode becomes inaccurate when network traffic changes significantly, we adopt a hybrid approach. We switch between the abstract simulation mode and full simulation mode in response to the traffic changes. Statistics are collected online under full simulation mode to update the abstract simulation mode to be used. In evaluation, we demonstrate that our approach is 9 times faster than the original simulation with a loss of accuracy less than 10% in the end-to-end delay and 1% in the number of messages successfully transmitted across the network.**

I. Introduction

With the advance of sensor network research, numerous protocols have been proposed recently. Validation and analysis of these protocols are very important for system designers to make right decisions to meet application requirements. One method available for validation and analysis is mathematical modeling. Unfortunately work in mathematics is limited due to the complex, dynamic, and unpredictable nature of these systems as they scale to thousands of devices. One can also explore new protocols for sensor networks directly with the physical devices (e.g., mica2). Problems occur again with the complexity and scale of these networks. The sensor devices that exist today provide a limited subset of the functionality and scale. Due to the practical limitations of mathematical analysis or physical network implementations, simulation is an important tool for researchers in this field, at least during the initial stage of the system design. Aside from just providing a means of representing models where it was previously impossible or just not practical, simulation also allows fast evolution of protocols without the significant effort of reworking a mathematical analysis or reloading code onto thousands of physical motes. The need for simulators capable of modeling these large, complex, and seemingly limitless networks grows every day. Although many simulators have been built to date (ns-2 [8], GloMoSim [20], SSF [18], SensorSim [16], TOSSIM [10], SENSE [3], J-Sim [6], ATEMU [15], SENS [19], TOSSF [14]), the scalability of these tools is not satisfactory. For example ns-2, currently the most popular network simulator, suffers an order of magnitude performance penalty by invoking the Tcl interpreter during a simulation run and occupies approximately gigabytes memory to simulate thousands of nodes. This limitation in scalability is a problem as networks being tested grow and simulators are left with the task of modeling the incredibly complex behavior of these large systems.

One method of improving the scalability of network simulators is through model abstraction. *The goal of model abstraction is to sufficiently reduce the complexity of a model, without suffering (too great) a loss in accuracy*. With this thought in mind, our work is an attempt to study the feasibility of using abstraction as a solution to the problem of scalability. Specifically, we are targeting to MAC layer abstraction, because based on our empirical profiling results, over 90% simulation time is spent on the characterization of the MAC layer behaviors.

The originality of our approach lies in three aspects: First, to the best of our knowledge, all prior work on data collection for abstraction occurs off-line prior to simulation. In contrast, our abstraction is established on-line by analyzing the data collected from a full simulation mode. This approach provides a high fidelity than off-line static abstractions. Second, we dynamically switch the granularity of the simulation based on the feedback from the traffic rate monitor. This approach embraces the accuracy gain from the detailed simulation and speedup from high-level abstraction. Third, our approach is independent of upper layer implementations. Different network protocols and applications can run on top of our implementation seamlessly. we demonstrate that our approach is 9 times faster than the original simulation with a loss of accuracy less than 10% in the end-to-end delay and 1% in the number of messages successfully transmitted across the network.

The rest of paper is organized as follows. In section II, we discuss research to date that has pursued similar goals. Specifically we look at different solutions to scalability and identify the uniqueness of our solution. In section III we discuss the rationale behind our abstraction and the basic ideas behind our solution. Section IV provides detail on our implementation within the GloMoSim framework and section V follows with results and analysis. Section VI inspects the unresolved issues that needs to be addressed in the future work. We conclude in section VII.

II. State Of The Art

In search of a solution to augmenting scalability in wireless network simulators, we looked at prior work in two general areas of network simulation research. The first, achieving simulation scalability, gave us insight into prior strategy and potential solutions to help guide us towards our ultimate attempt at a MAC layer abstraction. The second, simulation validation, helped us understand the important aspects and techniques to validate our ultimate solution. Prior research in both of these areas are discussed in this section.

A. Scalability

To solve the scalability problem, modern research on network simulation has taken two major approaches: parallel simulation and simulation abstraction. The first, parallel simulation, has been implemented in architectures such as DaSSF [18]and GloMoSim [20] and has proven adequate in speeding up simulations on a limited scale. [11], [13] and [20] discuss the application and results of implementing parallel and distributed simulation across several processes to achieve speedup. The biggest advantage of parallel simulation is that speedup is gained without sacrificing the accuracy and granularity of the simulation. This research on parallel simulation environments and techniques is complementary to our work.

The second approach, called *model abstraction*, is the focus of our work. [1], [4] and [17] outline fundamental goals and tradeoffs to consider when simulating a network layer abstraction. Additionally these papers provide important insights into understanding the effects of model abstraction and testing for possible simulator invalidation. [5] looks at two abstraction techniques: centralized computation and abstract packet distribution. Centralized computation saves memory consumption and time by centrally computing protocol states to reduce the workload and complexity of performing these computations for every simulated node. Abstract packet distribution avoids link-by-link packet transmission by scheduling packets directly at the receiving end. This second technique is similar to our work in that it eliminates potentially unnecessary details during packet transfer. In our work we implement a similar, but less aggressive technique that results in a more detailed and hence more accurate abstraction. Hybrid simulation, as implemented in SensorSim [16], is another approach to achieving speedup through abstraction. This approach utilized data collected from a real system to apply statistical data to simulation models. Our approach is different from this technique in the sense that we use adaptive simulation granularity from fine detailed packet-level simulation and apply it to coarse abstract-level simulation. [1] uses a fluid-based approach to speedup simulation. In this solution, speedup is achieved by coarsening the representation of network traffic from a packet level granularity to a flow level granularity where closely related packets are substituted with a single packet. [12] abstracts details of an 802.11 MAC Protocol by substituting probabilistic packet arrival data and statistical packet arrival times to remove the details of the 802.11 MAC protocol contention phase. This research most closely resembles our own with the exception that data collection occurs off-line. This limitation prevents researchers from applying statistical data to situations that have not been previously encountered and makes dynamic traffic patterns difficult to model. Our research addresses this limitation by performing online data collection by monitoring traffic patterns to determine when simulated abstraction is appropriate.

B. Validation

Aside from the work designing and implementing our abstraction, we also studied model validation to better understand the overall effect and validity of our research. Validation tests such as the chi-square goodness of fit test [9] can be used to test whether the simulated model reflects the real situation. [7] and [4] discuss issues and techniques for model validation and analysis. Although various methods and techniques are proposed and discussed in these papers, our primary focus is on comparing the simulated results of our abstraction with the results of the high granularity simulation [12]. Since different networking scenarios have different impacts on simulated results, a limited set of case comparisons are not enough to claim the validation of any model. Rigorous model validation, specifically as it applies to abstraction, is left as an area for future research.

III. Methodology Overview

Accurately modeling wireless networks on the scale of hundreds of thousands of nodes has been and will remain a challenging problem for research. Despite the effects of Moore's law it remains impractical to handle simulations of large-scale networks over reasonable periods of simulation time to collect precise experimental data.

A. Design Goals

The array of simulators developed to date provide varying levels of detail, accuracy, scalability, modularity (flexibility), etc. Unfortunately the more detailed a simulation becomes, the less capable it is of scaling to large complex networks. It is crucial to decide what level of detail can be sacrificed in exchange for simulation speed and scalability. Since in the wireless scenario, nearly $90 \sim 95\%$ of simulation time is spent modeling details of the MAC and below, abstracting these layers can provide a large speedup while preserving the accuracy of upper-layer behavior. The higher level protocols are ideal for our abstraction because they are not concerned with MAC layer detail so long as the packet transmission delay remains consistent with that of a detailed model's behavior. More specifically, this work attempts to solve the speedup problem by abstracting out MAC layer implementation detail for the MAC protocols (e.g.MACAW [2]) in GloMoSim [20]. While the specific abstraction we implement is under GloMoSim, our concept of MAC layer abstraction is generic and can be applied across other simulators.

The goals of our work are four fold. 1) Primarily we want to increase the speed of the simulation, in turn increasing the potential for scaling to very large network models. 2) We also need to validate the results of our abstracted implementation against a more detailed and correctly implemented simulator. 3) Slightly less important but along the lines of validation is the need to fully understand the effects of our abstraction on the model being simulated. The importance of understanding both the positive and negative impacts of our abstraction allows us to continue to refine our technique and potentially apply it to other layers or on other MAC layer protocols in future research. 4) Finally our fourth goal is to identify additional bottlenecks in simulations in hopes of addressing these bottlenecks in future work.

B. Model Assumptions

We assume that the current wireless simulators (e.g. GloMoSim [20], ns-2 [8]), prior to our modifications, is a valid representation of the MAC layer and therefore the baseline simulation is used as a means of comparison. Although the success of our abstraction does not hinge on a valid initial implementation, we assume a valid model so that similar implementations on other validated simulators can expect similar results. For our design we also impose the assumption of relatively consistent network traffic. This assumption is made to ensure that enough time is spent abstracting MAC layer behavior. We discuss the relaxation of this assumption later. Currently our model assumes that the simulated network is static or with a low mobility. With a high mobility, it is difficult to characterize the dynamics of radio propagation and its effect on our abstraction, so further investigation on high mobility is left for future work.

C. Overview of Detailed MAC Layer Simulation

Before discussing specifically how we added abstraction to the MAC layer simulation, it is important to understand certain aspects of the MAC layer simulation. To illustrate the simulation process, here we use GlmoSim [20], a typical discrete wireless simulator, as an example. We note this paper is described in the context of GloMoSim, however the design idea can be applied in other discrete simulators as well. GloMoSim was developed as a modular library of components that contribute to an extensible, robust, and dynamic simulator for wireless networks. By isolating nodes' communication layers into independent modules, GloMoSim allows the researcher to "plug and play"different protocols (i.e. protocols that they develop and implement) without concern for the inner workings of other architectural layers. To handle the overall organization of the simulator, GloMoSim implements a main module responsible for instantiating and organizing nodes, scheduling messages between modules, and tracking the exchange of messages within the simulation. The most important responsibility of the main component is to invoke calls to specific modules as appropriate to control the overall sequential flow of events throughout the simulation. This flow of control is handled by scheduling and passing *messages* which represent events in the simulation. A message, as defined by GloMoSim, has a multitude of purposes. For example, a message used for simulating communication between the network and radio layer not only encompasses the application payload, but also contains any header information that previous layers (application, transport, network, etc..) have attached to this payload. For communication messages, this mimics the way packets are packaged in the OSI seven-layer architecture. Aside from their obvious use for communication, messages are used to schedule node timeouts, handle mobility, or provide any form of communication between modules during simulation. To better understand the use of messages within GloMoSim we provide a simplified example of a packet being sent into the simulated environment. When node A's network layer has a packet to send over a single hop, the network layer schedules a message (encompassing the packet and upper layers' header information) to the MAC layer. The simulator core stamps this message with a time of delivery and returns to the main sequential flow in the main module. When the simulation time for this message arrives, the main module looks at its type and passes it to the appropriate node's MAC layer. Upon receiving the message, the MAC layer further decodes the type of MAC layer message so that it can be handled appropriately. In this situation, the MAC layer recognizes that it is the beginning of a new exchange to the network, so it initiates several more messages. These messages are also scheduled to take place at specific simulation times and are later handled by the main module when it becomes appropriate. The subsequent message scheduled is a notification of message propagation from the Radio layer. This message results in *every node* within the simulator receiving the propagated message and scheduling messages for overhearing, responding when appropriate, backing off, etc. As you can see from this significantly simplified example, due to the broadcast nature of wireless communication, a packet being sent out into the network involves the scheduling of two to possibly 20 or more GloMoSim internal messages per simulated node. This overhead in terms of simulation time escalates when a network has hundreds of thousands of nodes.

D. Architectural Solution

To improve simulation scalability for this described architecture, we attempt to abstract a layer of the previously defined GloMoSim architecture to reduce the number of messages exchanged during a simulation. Due to the fact that many sensor network simulations attempt to understand network, transport, or even application layer behavior, our abstraction argues that the details of how MAC layer message exchange takes place are not quite important so long as the overall transmission behavior of the model is maintained. Specifically this allows us to abstract the MAC layer and therefore reduce the MAC layer messages being modeled to speed up the overall simulation time. As shown in Figure 1, our solution adds four essential modules and one orthogonal module to the GloMoSim

simulator. The four essential modules are a Data Collector module, an Abstraction Model Builder, a Traffic Monitor and A Mode Switcher. The orthogonal module is simply the addition of data collection for model validation. To better understand these changes, Figure 1 shows an architectural schematic of how our changes fit into a simulator architecture.

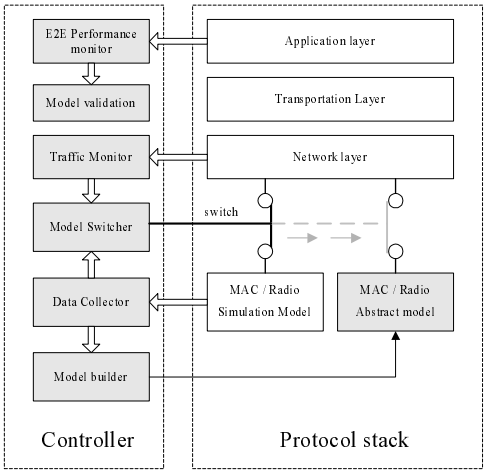

Fig. 1. Architecture for MAC layer Abstraction

The description of this architecture is as follows.

- **Data Collector:** In Figure 1, the Data Collector module is used to monitor and collect information on the detailed and complete exchange of packets as specified in the protocol. The Data Collector is actively collecting data when simulator is in the Full Simulation Mode. During this mode the simulator functions as previously implemented by the GloMoSim developers. By collecting MAC layer message data during the simulation, we attempt to use this collected data to apply a statistical model that removes this MAC layer message exchange.
- **Model Builder:** Upon switching to Abstract Simulation Mode, the Model Builder utilizes information previously collected by the Data Collector to determine an applicable statistical model for our abstraction. Currently the Model Builder implements a simple statistical history-index model, described in section IV-B. Once a switch is made, the Model Abstract Simulation Mode utilizes the statistical model to abstract the message exchange with a predicted delay and corresponding probability of successful packet arrival at the receiving node. This abstraction significantly reduces the simulation time.
- **The Traffic Monitor and Mode Switcher:** they are provided to actively monitor and respond to simulated traffic flow during a simulation. These modules monitor packet rates in the network and utilize a threshold-based method of deciding when to switch from Full Simulation Mode to Abstract Simulation Mode and back. The combined effort of these modules is referred to as *Toggling*.
- **Model Validation Module:** The fifth module in our implementation, data collection for model validation, is implemented across the End-to-End Performance Monitor and the Model Validation Unit. Currently our End-to-End Performance Monitor collects statistics on end-to-end message delay and the Model Validation unit has not been fully implemented. The current solution is discussed further in section IV-D and work with online Model Validation and the addition of a feedback mechanism for dynamic optimization of abstraction parameter is left for future work.

It is important to note that our MAC layer abstraction subsequently abstracts the details of the radio layer for all packets. This happens as a result of the packets no longer being sent as a radio signal. Instead, packets are directly passed to the receiving nodes' routing layer avoiding both the MAC and radio layer altogether.

E. Abstraction Parameters

The mode switch can be tuned by changing up to three abstraction parameters. 1) The *initial stride* is the number of packets that are monitored at the beginning of a simulation during Simulation Mode. When this number of packets has been simulated, the mode switches to Abstract Simulation Mode (on figure 1 the switch position would be to the right). 2) The *heartbeat period* controls the frequency of traffic monitoring. The "packet arrival rate"is the rate at which (unicast) packets are transmitted from the network layer to the MAC layer for all simulated nodes. Each time a heartbeat is issued, we compare the arrival rate in the time period that just finished with the arrival rate in the previous time period. If (the absolute value of) this ratio exceeds a *threshold* (our third abstraction parameter), the packet arrival rate is considered fresh and we no longer consider past data applicable to the current abstraction and a switch back to full simulation mode is needed. The abstraction parameters: (i) initial stride, (ii) heartbeat period and (iii) threshold, are static during a simulation run and easily modifiable. Intuitively, a longer initial stride, shorter heartbeat, and smaller threshold result in a more accurate, and therefore longer simulation. This tradeoff is discussed in section V-A.

IV. Implementation

Our initial approach to abstracting a MAC layer protocol expose us to the design and code of various simulators. After looking into ns-2 [8], GloMoSim [20], SSF [18], and the TinyOs simulator [10], we eventually choose GloMoSim due to its ease of understanding, relatively good performance, and modularization of layers.

As stated before, our MAC layer abstraction toggles between two different modes during a simulation. The Full Simulation mode implements the detailed packet exchange as previously implemented by GloMoSim. The only change to the previous implementation is the collection of data to be used in the Abstract Simulation Mode. The Abstract Simulation Mode reduces the number of messages exchanged during simulation and therefore has a vast impact on the overall simulation time.

To switch between modes we implemented what we call a Toggle feature. The final piece of our implementation involves data collection for analyzing the effect of our abstraction on end-to-end delay and packet loss for the models chosen. These four components are described in detail below.

A. Data Collection

In Full Simulation mode the simulation runs exactly as it had previously been implemented by the GloMoSim designers. For the purposes of this paper we are going to assume that this original design was correct and can therefore be used as the baseline for our analysis. The only difference in Full Simulation mode is that we have inserted several data structures in combination with a fair amount of logic to eavesdrop and collect data for the detailed simulation as it runs. This eavesdropping takes place in both the MAC layer, Radio layer. For example, the main body of the simulation ruling MACAW protocol [2] goes as follows : When the network layer has a packet to be sent to its next hop, the MAC layer is notified of this packet and a RTS is scheduled. At this point we collect the time in which the RTS was initiated and the node in which the RTS is destined and we set the sending nodes state as SENDING RTS and the receiving nodes state as AWAITING RTS. From here on the Data Collection component tracks the exchange of packets between nodes, updates the node's state accordingly, and sets variables within the Data Collection structure that records how long the exchange took and whether or not the exchange was successful. Any collisions with other RTS, CTS, Data packets or other noise are handled by the simulation and recorded as appropriate. Finally when this MAC layer exchange has completed the nodes state is reset and its collection data index is updated as appropriate.

B. Model Abstraction

At some point during the Data Collection portion of our simulation, the toggle feature of our implementation realizes that enough data has been collected and it is time to switch to abstraction. The implementation of the toggle feature follows this description of the Abstract Simulation model. The Abstract Simulation Mode is used to significantly reduce the number of messages sent during the simulation and therefore its goal is to reduce the overall simulation time. We do this by applying the data collected during the Full Simulation mode to provide a statistical approach to generalize data flow through the network. This is possible under fairly regular traffic patterns or sending intervals per node. An example application is a temperature sensing application where nodes report temperature readings to a base station every second. We statistically determine the send times and whether or not a packet successfully arrives by generating a random number in the range of the number of data exchanges previously collected. Using this random number, we index the data structure where this information resides. If the index into our data structure points to a successful exchange then we use the time of this exchange and directly send our current packet to the network layer of the receiver with a set delay appropriate to the time it would have taken (statistically speaking) for this packet to arrive. This time includes the radio transmission time, radio propagation time, overhead time incurred during message exchange between layers, and any back-off that occurred due to collisions in the network. If our index points to a data packet that had previously been dropped, we drop this packet accordingly. These dropped packets statistically model packet loss in the model. While our abstraction methodology is fairly simple, it works for several reasons. Primarily we assume fairly constant traffic which allows us to look at behavior from the past to determine current behavior. Although this assumption seems stringent, it is relaxed by applying strict monitoring of aggregate packet rate by our toggle component which can be modified by changing the abstraction parameters. As you will see, the worst case scenario for our simulation is when the smooth traffic assumption is relaxed and nodes sporadically send messages changing the networks aggregate packet rate. In this scenario our model realizes that the changing rate requires fresh data and continues to toggle back to Full simulation mode with Data Collection. When this happens the simulation runs as it had prior to our implementation with the exception that we incur the additional cost of data collection and therefore see slight performance loss.

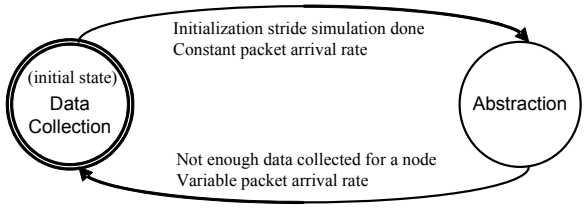

Fig. 2. Phase Transition

C. Toggling Between Modes

The problem of switching between Full Simulation mode and Abstract Simulation Mode is handled by a toggle component that is responsible for monitoring traffic flow and data collection. The state transition diagram for our Toggling component is shown in figure 2 and a visual representation of the Toggling behavior is provided in figure 3. Specifically the toggle component functions as follows. While in Full Simulation mode the toggle component monitors the total number of data packets collected. When the total packets collected reaches some threshold, (the abstraction parameter previously discussed) the toggle component modifies a state parameter which tells the MAC layer to use our Abstract Simulation Model as implemented.

Once the simulation is running in Abstract Simulation Mode, the toggle component begins to monitor the aggregate packet transfer rate provided by the network layer of every node in the network. The heartbeat counter is used to periodically check the aggregate rate of messages sent by comparing the total number of messages sent between the current time and the

last heartbeat with the total number of messages sent between the two prior heartbeats. Because the time between heartbeats remains constant this value provides a rate. At each heartbeat these two rates are compared and if the difference between them exceeds some threshold (one of the abstraction variables previously mentioned) then the aggregate send behavior is determined to have changed significantly enough to warrant switching back to Full Simulation mode. Another case where our mode switches from Abstract Simulation Mode to Full Simulation Mode is when the model is in Abstract Simulation Mode and a node for which we do not have enough statistical data wants to send a packet. At this point we switch modes so that statistics for that node can be appropriately collected. Note that this deeply impacts the performance of the simulation as all nodes switch to Full Simulation Mode simultaneously. In our design we choose to implement the Toggling feature as a network parameter (as apposed to a node parameter) since nodes in abstraction do not send packets and as a result would invalidate data collected for other nodes simultaneously. One additional challenge we face in our implementation is when the toggling mechanism is invoked during a packet exchange in Full Simulation mode. Due to the complexity of a more thorough solution, our current implementation simply drops the packet in transit. For our simulation, these lost packets are rare and as a result do not have significant impact on our results.

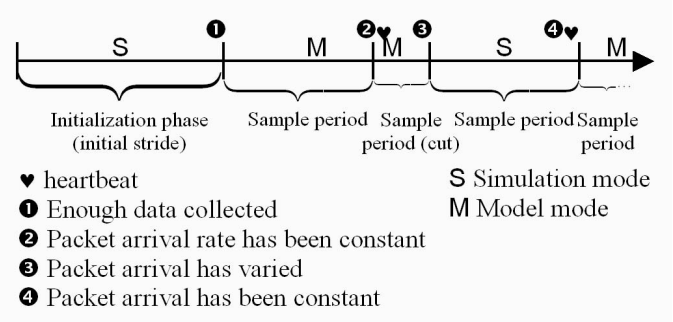

Fig. 3. A hybrid Approach with Toggling

D. End-to-End Data and Model Validation

The final modification to GloMoSim is building a mechanism to collect information for comparison and validation purposes. GloMoSim provides information specific to each layer implemented in the simulation but unfortunately this information is not detailed enough to satisfy our needs. Our modification involves an addition to a component of the simulator that collects the end-to-end delay for all packets successfully sent through the network. By collecting the end-to-end delay for each packet between specific nodes, we are not only able to get the average and standard deviation of the end-to-end transfer time, but the node to node data loss (for our CBR implementation) allowing us to ensure that the MAC layer appropriately handles collisions and therefore congestion in the network. Since the end-to-end delay statistics are a necessary component for validating our abstraction, but are not necessary for the abstraction itself, we take measure to ensure that this additional code does not affect our speedup analysis. Fortunately since this code is necessary for our comparison between models, we implement these changes in both the original and abstracted simulators to provide consistent results.

V. Evaluation

In the evaluation, nodes in a sample network are uniformly placed in a field of 1000m x 1000m, in which 10 CBR flows between 30 nodes are transmitted during 900 seconds. Because these flows have random start times (0 to 2 seconds), random starting and end points (fixed for the duration of the simulation), and random rates (2 to 1000 packets per second), some parts of the network are more active than others and the load on each node changes with time. By using several CBR flows with randomness in their parameters, we create a more general load for the network. To test the effects of our abstracted simulator on the model previously described we run the model several times with each simulated run taking place on different architectures of the GloMoSim simulator. We maintain a seed value of 1, allowing pseudo random numbers generated during the simulation to be consistent between each architecture simulated. The simulation times for the architectures described below are compared in Figure 4.

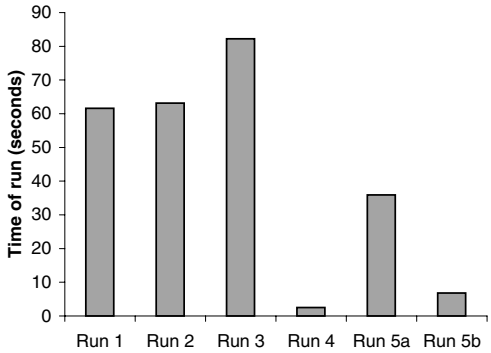

Fig. 4. Simulation Time under Different Runs

In a first run, we test the original GloMoSim architecture and measure the overall simulation time [1] We use this value as our reference time. For the remainder of our paper, we refer to this and the subsequently described runs as run 1, run 2, etc. In a second run, we use a modified version of the GloMoSim architecture that includes end-to-end delay data collection for all packets correctly transmitted at the application level. This architecture is also addressed as the "original simulation". In a third run, we use our fully modified version of GloMoSim with Toggling deactivated so that the simulator never enters our Abstract Simulation Mode. This simulation is used to determine the cost of online data collection. In a fourth run, we use our fully modified version of GloMoSim with Toggling and Data Collection deactivated and the simulator initially

[1] the simulations were run on a PC AMD Athlon 1.3 GHz with 768 MB of memory.

set to Abstract Simulation Mode. This architecture provides data on the maximum possible speedup. Finally in a fifth run (run 5a and run 5b), we use our fully modified version of GloMoSim as we have intended it to function with two different thresholds. We gather end-to-end delays for all packets correctly transmitted at the application level. We also call this run the "abstracted run".

A. Abstraction vs. implementation trade-off

To understand the effect of our abstraction on simulation time, we have run the five architectures previously described to obtain time comparisons for each architecture. As we can see from Figure 4, if the overhead of the end-to-end delay collector (run 2) remains low (2%), the overhead due to the data collector (run 3) is relatively high (30%). Fortunately, our assumption that most of the time spent in simulation is a result of MAC and radio layer details proves to be true. If messages from these layers were delivered directly from the network layer of a node to the network layer of the receiving node, simulation time would be 24 times faster. As a result, the overhead introduced by our implementation can be counterbalanced by the speedup incurred through abstraction. For our abstraction the simulation (run 5b) was 9 times faster than the original simulation and only 40% slower than the fastest possible run (run 4). This speedup occurred with a loss of less than 10% accuracy in the end-to-end delay and a 1% loss in the number of messages successfully transmitted across the network.

B. Accuracy vs. time trade-off

We assess the accuracy of our abstraction by monitoring a flow consisting of several hops through the simulated network. We choose to use multiple hops due to the fact that it allots more room for discrepancy between our simulated abstraction and the original GloMoSim architecture. In addition to assessing simulation times between architectures, we also run our model over varied threshold values as previously described. In these cases the initialization stride is 2000 packets and the heartbeat period is 2 seconds with the threshold varied at 2% (case 5a) and 5% (case 5b).

C. Threshold Comparison

Figure 5 shows us end-to-end delays, the number of messages that arrive along our chosen flow, and the overall simulation time for both the original simulation and our abstracted run (run 2, run 5a, run 5b). We can see that the end-to-end delays viewed by the application layer help to validate our abstraction (less than a 10% difference). From this figure we can also see a tradeoff between end-to-end delay accuracy and simulation time as a result of the different values for our threshold parameter. This is the result of an increased threshold allowing more variance in traffic rate prior to switching back to Full Simulation mode. This greater threshold results in less switching and therefore a slight decrease in accuracy with a significant increase in speedup. Figure 6 and Figure 7 depict this behavior by plotting end-to-end packet delay as a function of the packet sent for both threshold values. The packet sent can also be looked at as a time parameter for this figure.

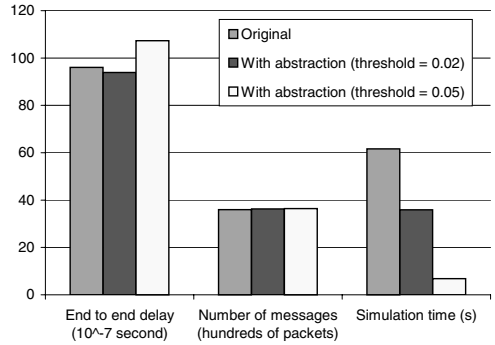

Fig. 5. Model Comparison (run 2, run 5a, run 5b)

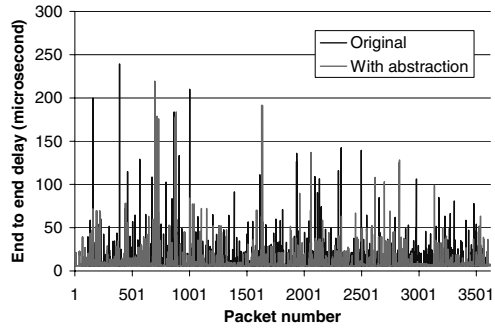

Fig. 6. End-to-End Delay for Threshold = 0.02

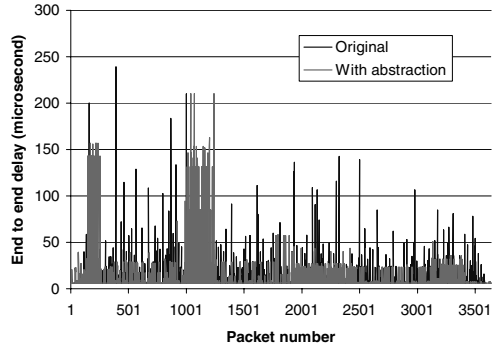

Fig. 7. End-to-End Delay for Threshold = 0.05

From these results we can see the importance of fine tuning our three abstraction parameters. These parameters allow for the adaptation of the simulator to more appropriately fit the model in simulation. Although we currently modify these parameters off-line, this issue could be re-solved by profiling the simulator or implementing an auto-profiling and auto-adaptation mechanism to handle parameter modification online. To do this the simulator would run an initial phase during which it tries to find optimal parameter values. The simulator could then change these parameters dynamically to adjust the quality of the abstraction. Ultimately, the end user could provide a single parameter (i.e. a value between 0 and 10) with 0 meaning

optimize for speed and 10 meaning optimize for accuracy at the expense of speed. This could be implemented using a feedback control loop in the simulator with the speed and accuracy references as inputs. Note this is a second level of auto-adaptation as we already provide a mechanism for the modified simulator to respond automatically to load variations. Further discussion of online parameter optimization is left for future work.

VI. Future Work

Due to space constraints, several directions remain both unexplored and under explored. The following ideas and issues are left for future work:

- In our implementation we have only tested/validated our abstraction for small models implemented on MACA using CBR (constant bit rate). For further validation and to test the true success of our model we must continue to run tests on larger, more complex, and more dynamic models to better understand the effects of our abstraction.
- For our first Abstract Simulation Model we simply used a random index into an array of collected data to determine send time and the probability of a successful data exchange. This simple model abstracts data when traffic flow remains fairly constant and switches to data collection when traffic flow changes. The loss of speedup incurred by switching back to data collection could be reduced by applying a system identification model that more aptly handles dynamic traffic flow in our abstraction.
- Automatic model validation is another area left open for future work. In our implementation we used simple mathematical validation and manual control of abstraction parameters to determine the accuracy and effect of our abstraction. It is conceivable that feedback control and dynamic adjustments could be implemented to manipulate the abstraction parameters and optimize speedup online.
- Finally, our first iteration involved extending the MACA protocol as implemented by GloMoSim. Future work could involve similar abstraction to other MAC layer protocols such as 802.11 or MACAW to determine the extensibility of our design.

VII. Conclusion

Research on protocols for wireless ad-hoc sensor networks continue to reach new limits. To fully understand the impact of these protocols researchers must be able to simulate networks of hundreds of thousands to potentially millions of nodes. A major problem of current wireless network simulators is their inability to simulate networks of this scale. Research to date has taken several approaches to solving this problem. In this paper we present the novel design, implementation, and results of an online algorithm to switch the simulator between a *Full Simulation Mode* during which data are collected, and an *Abstract Simulation Mode*, which uses the collected data to abstract the simulator's MAC layer. This innovative design is implemented on the GloMoSim simulator to specifically abstract the MACA protocol and can easily be ported to work across other protocol layers on a variety of simulators. Our design can be utilized to speed up future simulations studying transport, network, or application layer protocols. In the evaluation, we achieved 9X speedup at the cost of only 10% loss in end-to-end delay accuracy combined with a negligible difference in the number of messages correctly exchanged through our abstraction. While these results are promising they are by no means the limit of this design. Future work on optimizing our design includes implementing alternate abstract simulation models and extending our work to include mobility and the more adequate handling of dynamic network behavior.

References

[1] J. Ahn and P. Danzig. Packet network simulation: Speedup and accuracy versus timing granularity. In *IEEE/ACM Transactions on Networking*, pages 743–757, 1996.

[2] V. Bharghavan, A. Demers, S. Shenker, and L. Zhang. Macaw: A media access protocol for wireless lan. In *the SIGCOMM '94 Conference on Communications Architectures*, 1994.

[3] G. Chen, J. Branch, M. J. Pflug, L. Zhu, and B. Szymanski. SENSE: A Sensor Network Simulator. *Advances in Pervasive Computing and Networking,Springer: 249-267*, 2004.

[4] J. Heidemann and et. al. Effects of Detail in Wireless Net-work Simulation. In *SCS Multiconfer-ence on Distributed Simulation*, pages 3–11, 2001.

[5] P. Huang, D. Estrin, and J. Heidemann. Enabling Large-scale Simulations: Selective Abstraction Approach to the Study of Multicast Protocols. In *USC/ISI, USC*, 1998.

[6] J-SIM. *The J-SIM Project*. Available at `http://www.j-sim.org/`.

[7] D. B. Johnson. Validation of Wireless and Mobile Network Models and Simulation. In *DARPA/NIST Workshop on Validation of Large-Scale Network Models and Simulation*, 1999.

[8] Kevin Fall and Kannan Varadhan. *The ns Manual*. Available at `http://www.isi.edu/nsnam/ns/`.

[9] J. L.Devore and et.al. Probability and Statistics for Engi-neering and the Sciences. In *5th Edition ISBN 0-534-37281-3 Duxbury Press New York, NY*, 2003.

[10] P. Levis, N. Lee, M. Welsh, and D. Culler. TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications. In *First ACM Conference on Embedded Networked Sensor Systems (SenSys 2003)*, November 2003.

[11] J. Liu. Performance Prediction of a Parallel Simulator. In *PADS'99*, 1999.

[12] J. Liu, D. M. Nicol, L. F. Perrone, , and M. Liljenstam. Towards high performance modeling of the 802.11 wireless protocol. In *In WSC 2001*, 2001.

[13] T. Mineo and et. al. Impact of Channel Models on Simu-lation of Large Scale Wireless Networks. In *MSWiM*, 1999.

[14] L. F. Perrone and D. Nicol. A Scalable Simulator for TinyOS Applications. In *Proceedings of the Winter Simulation Conference, 2002*, 2002.

[15] J. Polley, D. Blazakis, J. McGee, D. Rusk, J. S. Baras, and M. Karir. ATEMU: A Fine-grained Sensor Network Simulator. In *SECON'04*, 2004.

[16] A. S. S. Park and M. B. Srivastava. SensorSim: A Simulation Framework for Sensor Networks. In *MSWiM 2000*, 2000.

[17] A. F. Sisti and S. D. Farr. Model Abstraction Tech-niques: An Intuitive Overview. In *Air Force Research Laboratory/IFSB*, 2000.

[18] ssfnet. *The SSFNET Manual*. Available at `http://www.ssfnet.org`.

[19] S. Sundresh, W. Kim, , and G. Agha. SENS: A Sensor, Environment and Network Simulator. In *Proceedings of 37th Annual Simulation Symposium, 2004*, 2004.

[20] X. Zeng, R. Bagrodia, and M. Gerla. Glomosim: a library for parallel simulation of large-scale wireless networks. In *the 12th Workshop on Parallel and Distributed Simulations*, 1998.