

# MAC Layer Abstraction for Simulation Scalability Improvements

Brian Blum, Tian He, Yvan Pointurier  
Computer Science Department  
University of Virginia, Charlottesville  
Charlottesville, VA 22903  
{bblum, tianhe, yvan}@cs.virginia.edu

## 1. Introduction

Advances in the field of computer science will continue to see the movement of processing from large systems to small, ubiquitous, memory and power constrained devices embedded in the environment in which we live. Instead of interacting through GUI interfaces and working on terminals, these embedded devices will utilize sensors and wireless communication to collect and transmit data seamlessly throughout networks of these devices. To achieve the complexity of systems being proposed in current research, Ad-Hoc networks will utilize a combination of fairly complex algorithms to implement protocols that have yet to be established.

For researchers in this field, it is difficult to explore new protocols or algorithms that will run on these proposed networks without physical devices capable of manifesting their proposed work. One method available for validation and analysis of these protocols is mathematical modeling. Unfortunately work in mathematics is limited due to the complex, dynamic, and unpredictable nature of these systems as they scale to hundreds of thousands or millions of devices. Another useful test bed for exploring new protocols for sensor networks are the physical devices or “motes” developed at Berkeley which can be programmed to test new algorithms. Problems occur again with the complexity and scale of these networks. The “motes” that exist today provide a small subset of the functionality and scale that research has envisioned for these devices making work in this medium cumbersome and cost preventive. Without mathematical analysis or physical network implementations, simulation becomes an increasingly important tool for researchers in this field. 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 re-working 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 [13, 14], GloMoSim [5], SSF [17], SensorSim [15]), the scalability of these tools will continue to irk sensor network researchers everywhere. 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 [13] and occupies 412MB memory to simulate 1000 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 current 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.”[16] With this thought in mind our work is an attempt to study the feasibility of MAC layer abstraction as a solution to the problem of scalability. In section 2 we discuss research to date that has pursued similar goals. Specifically we look at different proposed and implemented solutions to scalability and discuss why our solution is different. In section 3 we discuss the rationale behind our abstraction, the basic ideas behind our solution, and an overview of GloMoSim, the base simulator for our work. Section 4 provides detail on our implementation within the GloMoSim framework and section 5 follows with results and analysis. Section 6 looks at potentially unresolved issues that could be re-addressed and potential future work. We conclude in section 7.

## 2. 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 and test our ultimate solution. Prior research in both of these areas are discussed in this section.

## 2.1. 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 [11] and GloMoSim[12, 20] and has proven adequate in speeding up simulations on a limited scale. [11], [12], 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 to solving the scalability problem, model abstraction, is the focus of our work. [1], [4], [6], and [16] 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.

[8] 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 transport level node by node, link by link packet transmission by direct packet scheduling 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 [15], 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. The drawback of this approach is that the performance gain and accuracy is only good for stable traffic and the results of its application to dynamic network environments is unknown.

[10] 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 offline. 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.

## 2.2. 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 [3] can be used to test whether the simulated model reflects the real situation. [6], [7], and [9] 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.[10] Since different networking scenarios will have different impacts on simulated results, a single case comparison is not enough to claim the validation of any model. Model validation, specifically as it applies to abstraction, is left as an area for future research.

## 2.3. Novelty

The originality of our approach lies in three aspects: First, the abstraction model is automatically established on-line by analyzing the data collected from a fine granularity simulation. To the knowledge of our group, all prior work on data collection for abstraction occurs off-line prior to simulation. 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.

## 3. Methodology Overview

Accurately modeling wireless networks on the scale of hundreds of thousands to millions 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.

### 3.1. 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 80~90% of simulation time is spent modeling details of the MAC and radio layer, abstracting these layers can provide a large speedup while preserving the accuracy of upper-layer behavior. In this project we support research being performed on Network, Transport, or Application layer protocols. 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, our project attempts to solve the speedup problem by abstracting out MAC layer implementation detail for the MACA protocol as it was originally implemented in GloMoSim. While the specific abstraction we implement is for the MACA protocol, our concept of MAC layer abstraction can just as easily be implemented across other MAC layer protocols with similar results.

The goals of our project are four fold. Primarily we want to increase the speed of the simulation, in turn increasing the potential for scaling to very large network models. We also need to validate the results of our abstracted implementation against a more detailed and correctly implemented simulator. For this project we assume that the GloMoSim implementation, prior to our modifications, is a valid representation of the MAC layer and therefore the baseline simulation is used as a means of comparison. 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 will allow us to continue to refine our technique and potentially apply it to other layers or on other MAC layer protocols in future research. Finally our fourth goal for this project is to identify additional bottlenecks to scaling simulations in hopes of addressing these bottlenecks in future work.

### 3.2. Model Assumptions

One major assumption that we make in our work is to accept that the existing GloMoSim MACA protocol is accurate and provides a good model of "real" network behavior. 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 consistent network traffic. This assumption is made to ensure that enough time will be spent abstracting MAC layer behavior. Specifics of this assumption are discussed later.

Currently our model assumes that the simulated network is static. With mobility it is difficult to understand the dynamics of Radio propagation and its effect on our abstraction so further work on mobility is left for future work.

Finally our initial design assumes that the MAC layer (and subsequently the Radio layer) is the major simulation bottleneck. Justification and validation of this assumption is discussed in section 5.2.

### 3.3. The GloMoSim Architecture

Before discussing specifically how we added abstraction to the GloMoSim baseline simulator, it is important to understand certain aspects of the GloMoSim architecture. A more in depth explanation of the GloMoSim architecture can be found in [5]

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 (as defined below) which represent events in the simulation.

A message, as defined by GloMoSim, has a multitude of purposes. From the dimension of content, a message's data structure encompasses all of the necessary information pertinent to that message. For example, a message used for simulating communication between the network and radio layer will not only encompass the data that will eventually be sent out into the environment (or at least a value representing the size of this data for simulation purposes) but will also contain any header information that previous layers (application, transport, network, etc..) have attached to this message. 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 will recognize that it is the beginning of a new exchange to the network, so it will initiate several more messages. For MACA a RTS is scheduled along with a timeout message that will handle the case where a CTS is not returned in some specified period of time. 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, a packet being sent out into the network involves the scheduling of two to possibly 20 or more GloMoSim internal messages per simulated node, which when considered on the order of hundreds of thousands of nodes, becomes extremely costly to simulation time.

### 3.4. Architectural Solution

To improve simulation scalability for this described architecture, we attempt to abstract a layer of the previously defined GloMoSim architecture in hopes of reducing 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 assumes that the details of how MAC layer message exchange takes place are unimportant so long as the overall transmission behavior of the model is maintained. Specifically this assumption allows us to abstract the MAC layer and therefore reduce the MAC layer messages being modeled in hopes of speeding up the overall simulation time. For our work we only consider the MACA protocol as implemented by the GloMoSim designers.

Our solution involves three fundamental and one orthogonal change to the GloMoSim simulator. These three fundamental changes are the addition of a Data Collection mode, a Model Abstraction mode, and a mechanism to

monitor and switch between modes. The orthogonal change 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 the GloMoSim architecture. The description of this schematic as well as an overview of our implementation follows.

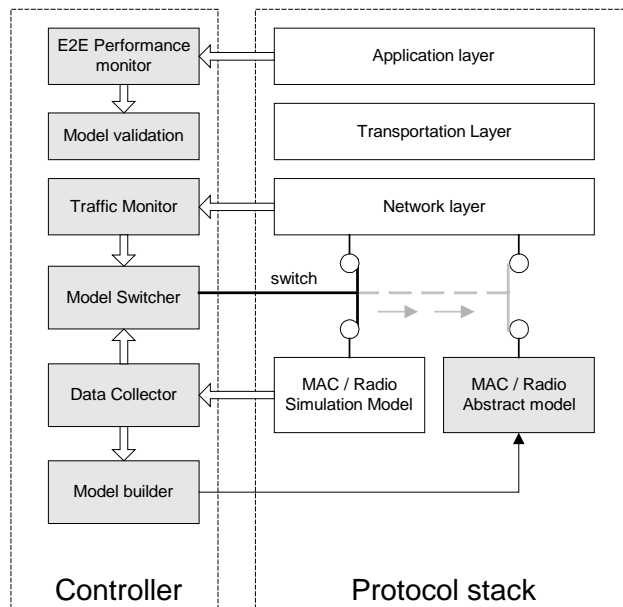


Figure 1: Implemented Architecture

The Data Collection mode is used to monitor and collect information on the detailed and complete exchange of RTS/CTS/DATA packets as specified in the MACA protocol. The Data Collector is actively collecting data while the MAC/Radio Simulation Model is running. 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.

The Traffic Monitor and Model Switcher 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 simple method of deciding when to switch from Data Collection to Abstraction mode and back. Hence forth the combined effort of these modules will be referred to as Toggling.

Upon switching to Abstraction 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 is only capable of implementing a simple model described in section 4.2. Future work could consist of significant enhancements to this module.

Once a switch is made, the Model Abstraction mode utilizes the statistical model to abstract the MACA RTS/CTS message exchange with a predicted delay and corresponding probability of successful RTS/CTS/DATA arrival at the receiving node.

The fourth change in our implementation, data collection for model validation, is implemented across the E2E Performance Monitor and the Model Validation Unit. Currently our E2E Performance Monitor simply collects statistics on end to end message delay and the Model Validation unit has not been implemented. The current solution is discussed further in section 4.4 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 out details of the Radio layer for RTS/CTS/DATA packets. This happens as a result of the packets no longer being sent as a radio signal. Instead, RTS/CTS/DATA packets are directly passed to the receiving nodes' transport layer avoiding both the MAC and Radio layer altogether.

### 3.5. Abstraction Parameters

The model switch can be tuned by changing up to three abstraction parameters. The *initial stride* is the number of packets that will be monitored at the beginning of a simulation during Simulation Mode. When this number of packets has been simulated, the model switches to Abstraction mode (on figure 1 the switch position would be to the right). A *heartbeat*, our second abstraction parameter, is then issued regularly. 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. The switching mechanism that utilizes these parameters is discussed further in section 4.3.

The abstraction parameters; *initial stride*, *heartbeat period* and *threshold*, are static during a simulation run and easily modifiable. Intuitively, a longer initial stride, shorter heartbeat, and smaller threshold will result in a more accurate, and therefore longer simulation. This tradeoff is discussed in section 5.2.

## 4. Implementation

As an aside to the work detailed in this paper, our initial approach to abstracting a MAC layer protocol exposed us to the design and code of various simulators. After

looking into ns-2, DaSSF, GloMoSim, and the tiny-os simulator [18], we eventually settled on 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 Data Collection mode implements the detailed RTS/CTS/DATA packet exchange as previously implemented by GloMoSim. The only change to the previous implementation is the collection of data to be used in the second mode. For this paper we have called this second mode the Abstraction mode and it involves adding functionality that significantly changes the inner working of GloMoSim. The Abstraction 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.

### 4.1. Data Collection

In Data Collection 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 Data Collection 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, and the main body of the simulation and is implemented 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 `SENDINGRTS` and the receiving nodes state as `AWAITINGRTS` (See state transition diagram in Appendix 1). 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 is 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.

## 4.2. Model Abstraction

At some point during the Data Collection portion of our simulation the toggle feature of our implementation will realize 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 abstraction model.

The abstraction 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 Data Collection mode to provide a statistical approach to generalizing data flow through the network. This is possible because we assume fairly regular traffic patterns or sending intervals per node. An example application where this assumption materializes is a temperature sensing application where nodes report temperature readings to a base station every second.

The statistical approach we take for determining send times and whether or not a packet successfully arrives is implemented 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 by RTS/CTS/DATA exchange. This time includes the radio transmission time, radio propagation time, overhead time incurred during message exchange between layers, and any backoff 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 Data Collection mode. 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.

## 4.3. Toggling Between Modes

The problem of switching between Data Collection and Abstraction 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 Data Collection 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 Abstraction model as implemented.

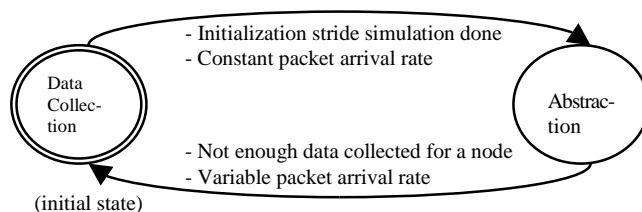


Figure 2: State Transition Diagram for Simulator Mode

Once the simulation is running in Abstraction 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 Data Collection mode.

Another case where our model switches from Abstraction Mode to Data Collection Mode is when the model is in Abstraction 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 Data Collection 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 RTS/CTS/DATA 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 RTS/CTS/DATA packet exchange in Data Collection 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.

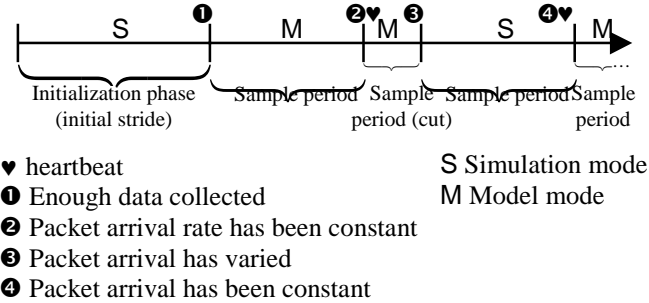


Figure 3. Toggling Behavior

#### 4.4. 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 a simple 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.

### 5. Performance Results

To test the speedup and validation of our abstraction we implement a relatively simple and understandable model with the following attributes:

- Terrain (1000m X 1000m)
- Number of Nodes – 30

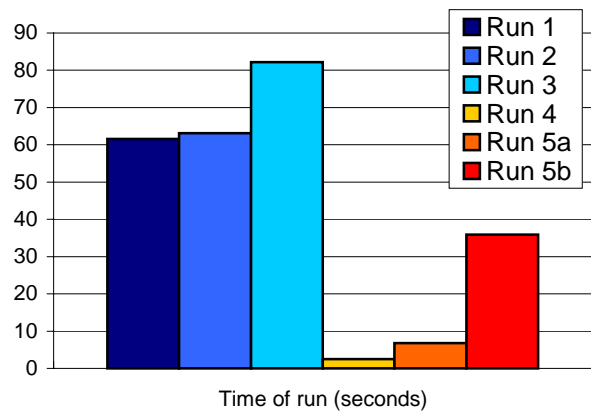
- Network Protocol – IP
- Routing Protocol – Bellman Ford
- MAC layer protocol – MACA
- ATM – CBR (Constant Bit Rate)
- Node Placement – Uniform
- Standard Radio Model (RADIO-ACCNOISE)
- Two Ray Propagation model
- Mobility – None

Additionally the traffic pattern simulated in this model and more details on the model parameters can be found in Appendix 2 and are discussed in more depth below.

### 5.1. Experiment Results

For our test a sample network of 30 nodes are uniformly placed in a field of 1000m x 1000m, in which 10 CBR flows 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 graph 1.



Graph 1: Architectural Comparison of Simulation Times

In a first run, we test the original GloMoSim architecture and measure the overall simulation time. We use

this value as our reference time. For the remainder of our paper we will 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 will never enter our Abstraction 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 set to Abstraction mode. This architecture provides data on the maximum possible speedup.

Finally in a fifth run, we use our fully modified version of GloMoSim as we have intended it to function. We gather end-to-end delays for all packets correctly transmitted at the application level. We will also call this run the “abstracted run”.

*Note:* the simulations were run on a PC AMD Athlon 1.3 GHz with 768 MB of memory.

## 5.2. 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 graph 1, 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 was 9 times faster than the original simulation and only 40% slower than the fastest possible run. 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.

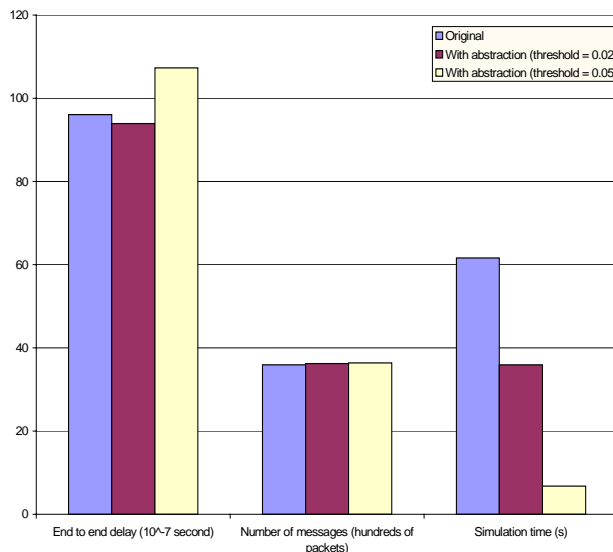
## 5.3. Accuracy vs. time trade-off

We assess the accuracy of our abstraction by monitoring a flow consisting of several hops through the simulated network. We choose the path from node 28 to node

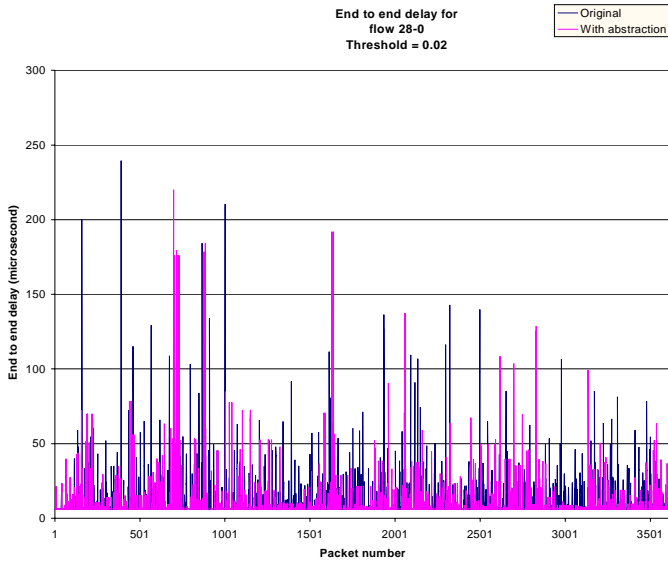
0 (see appendix 3) because it carries a large number of packets and is several hops long. We choose to use multiple hops due to the fact that it allots more room for discrepancy between our simulated abstraction and the original GloMoSim architecture. In addition to assessing simulation times between 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 5% (case 5a) and 2% (case 2b).

## 5.4. Threshold Comparison ( 5% .vs. 2%)

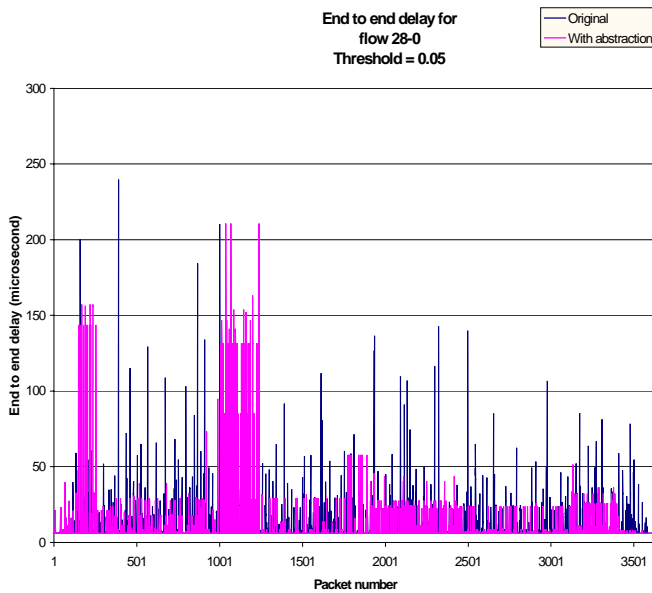
Graph 2 shows us end-to-end delays, the number of messages that arrive along our chosen flow, and the overall simulation time for both the original simulation and our abstracted run (run 2, run 5a, run 5b). We can see that the end-to-end delays viewed by the application layer help to validate our abstraction (less than a 10% difference). From this graph 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 Data Collection mode. This greater threshold results in less switching and therefore a slight decrease in accuracy with a significant increase in speedup. Graphs 3 and 4 depict this behavior by graphing 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 graph.



Graph 2: Model Comparison (run 2, run 5a, run 5b)



Graph 3: End to End Delay for Threshold = 0.02



Graph 4: 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 offline, this issue could be resolved 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 would be only 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.

## 6. Further Research

### 6.1. Open Issues

When analyzing the work we have done, the design we eventually settled on and the results of our abstracted model, we arrive at several open issues that remain for further discussion. We provide a list of questions which we leave open for our readers to ponder:

- Have we looked at enough data/statistics to validate our simulation?
- Have we carefully considered all of the assumptions made in our abstracted design?
- In GloMoSim, upon hearing an RTS or CTS, a node will apply a back-off algorithm and wait a given amount of time before attempting an RTS of its own. During data collection we only record the time between the actual initiation of an RTS and the reception of the DATA packet and although all other backoff time is recorded, we do not consider back-off time due to overhearing.
- Is there a better method of handling messages exchanged during the toggle between simulation and abstraction mode?
- We currently do not handle radio layer header information since our abstraction bypasses the radio layer. This is not a problem for GloMoSim's no-noise or acc-noise modules since there is no additional header information. Do other protocols (not currently implemented) include header information and how could we handle this increase in packet size for varying Radio layer header information?
- The "slow monotonic packet arrival rate variation": Suppose the packet arrival rate slowly increases in the network. More formally, suppose that the ration  $u_{k+1}/u_k$  is always below the threshold ( $u_k$  denotes the packet arrival rate during a period of time indexed by  $k$ ). In this scenario our model cannot detect the change, which can become dramatic after several heartbeats. A possible solution is to record the simulation rate during the previous Data Collection mode and compare all rates against this saved value.

## 6.2. Future Work

The limitations in time and resources have left us with several areas of work that 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. Potential models for future work include:

- Permutations of varying traffic and density
- Large scale tests with thousands of nodes
- Event occurrences (sudden traffic at a specific location in the network)
- Varied routing protocols
- Varied layers (network, radio, transport, etc.) : Specifically how protocols with flow control and congestion control mechanisms like TCP would interact with our abstraction.
- Simulation with mobility or other seemingly non-related features

For our first abstraction 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. Further discussion about this approach is discussed in appendix 4.

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.

GloMoSim was designed and implemented as a parallel simulation language that allows multiple processors to subdivide the simulation and utilize parallel processing to increase speedup. With limited knowledge of GloMoSim moving into this project and an ad-hoc first design, our implementation cuts across modules and eliminates this opportunity for parallelism. With seemingly little effort we should be able to re-package our design to fit within the modular structure of GloMoSim so that the Object Oriented features of the simulator are not violated and our abstraction will work with parallel simulations.

The effect of mobility on our abstraction remains unexplored and is left for future work.

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.

## 7. 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 “simulation mode” during which data are collected, and an “abstraction 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.

On reasonable scale networks (30 nodes), 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 abstraction models and extending our work to include mobility and the more adequate handling of dynamic network behavior.

## 8. References

- [1] J.S. Ahn, P.B. Danzig. “Packet network simulation: Speedup and accuracy versus timing granularity” *IEEE/ACM Transactions on Networking*, 4(5):743–757, October 1996.
- [2] Rajive L. Bagrodia. et. al., “Parallel Languages for Discrete-Event Simulation Models,” *IEEE Computational Science and Engineering*, Apr-Jun 1998, pp. 27-38
- [3] Jay L.Devore et. al., “Probability and Statistics for Engineering and the Sciences”, *5th Edition ISBN 0-534-37281-3 Duxbury Press New York, NY.*
- [4] Deborah Estrin, John Heidemann, et al., “Improving Simulation for Network Research.” *USC Computer Science Department*, March 1999

- [5] GloMoSim - <http://pcl.cs.ucla.edu/projects/glomosim/>
- [6] John Heidemann, et. al "Effects of Detail in Wireless Network Simulation. *In Proceedings of the SCS Multiconference on Distributed Simulation*, pp. 3-11. Phoenix, Arizona, USA, USC/Information Sciences Institute, Society for Computer Simulation. January, 2001.
- [7] John Heidemann, Kevin Mills, Sri Kumar. "Expanding Confidence in Network Simulation" *USC/ISI Technical Report 00-522*, April 2000.
- [8] Huang, Polly. Deborah Estrin, John Heidemann. et al., "Enabling Large-scale Simulations: Selective Abstraction Approach to the Study of Multicast Protocols." *USC/Information Science Institute, University of Southern California. Marina del Rey, CA.* July 1998.
- [9] David B. Johnson "Validation of Wireless and Mobile Network Models and Simulation." *Computer Science Department Carnegie Mellon University Pittsburgh, PA.*
- [10] Jason Liu, David M. Nicol, L. Felipe Perrone, and Michael Liljenstam. "Towards high performance modeling of the 802.11 wireless protocol." *In Proceedings of the 2001 Winter Simulation Conference (WSC 2001).*
- [11] Jason Liu, et. al., "Performance Prediction of a Parallel Simulator." *Proceedings of the Parallel and Distributed Simulation Conference (PADS'99)*, Atlanta, GA 1999.
- [12] Takai. Mineo. et. al., "Impact of Channel Models on Simulation of Large Scale Wireless Networks." *MSWiM*, Seattle WA USA. University of California, Los Angeles, 1999.
- [13] Monarch - <http://www.monarch.cs.cmu.edu/cmu-ns.html>
- [14] ns-2: <http://www.isi.edu/nsnam/ns/index.html>.
- [15] S. Park, A. Savvides and M. B. Srivastava, "SensorSim: A Simulation Framework for Sensor Networks," *In the Proceedings of MSWiM 2000*, Boston, MA, August 11, 2000
- [16] Alex F. Sisti, Steven D. Farr. "Model Abstraction Techniques: An Intuitive Overview." *Air Force Research Laboratory/IFSB.*
- [17] SSF - <http://www.ssfnet.org/homePage.html>
- [18] tiny-os project and simulator - <http://sourceforge.net/projects/tinyos/>
- [19] Jerry Waldorf and Rajive Bagrodia, "MOOSE: A Concurrent Object Oriented Language for Simulation." *International Journal of Computer Simulation*, Vol. 4(2), 1994, pp. 235-257
- [20] Zeng Xiang, Rajive Bagrodia, Mario Gerla. "GloMoSim: a Library for Parallel Simulation of Large-scale Wireless Networks." *Proceedings of the 12th Workshop on Parallel and Distributed Simulations -- PADS '98*, May 26-29, 1998 in Banff, Alberta, Canada

## Appendix 1: State Transition Scenarios

S-Node = Sending Node  
R-Node = Receiving Node

### Possible Problem Scenarios

#### Sending Node

- S-Node receives message before sending scheduled RTS
  - Handle message and delay RTS
- S-Node receives message while sending RTS
  - Ignore message and send RTS as scheduled
- S-Node receives message between finished RTS and received CTS
  - Ignore message and wait for CTS – (Collision?)
- S-Node receives message while receiving CTS
  - Mark sending and receiving node w/ Collision
- S-Node receives message between receiving CTS and sending DATA
  - Ignore message and send DATA
- S-Node receives message while sending DATA
  - Ignore message
- S-Node receives message after DATA sent
  - Ignore message

#### Receiving Node

- R-Node receives message before receiving RTS
  - Handle message appropriately
- R-Node receives message while receiving RTS
  - Mark sending and receiving node w/ Collision
- R-Node receives message between finished RTS and sending CTS
  - Ignore message and send CTS as scheduled
- R-Node receives message while sending CTS
  - Ignore message
- R-Node receives message between finished CTS and receiving DATA
  - Ignore message and wait for DATA
- R-Node receives message while receiving DATA
  - Mark sending and receiving node w/ Collision
- R-Node receives message after DATA received
  - Handle message appropriately

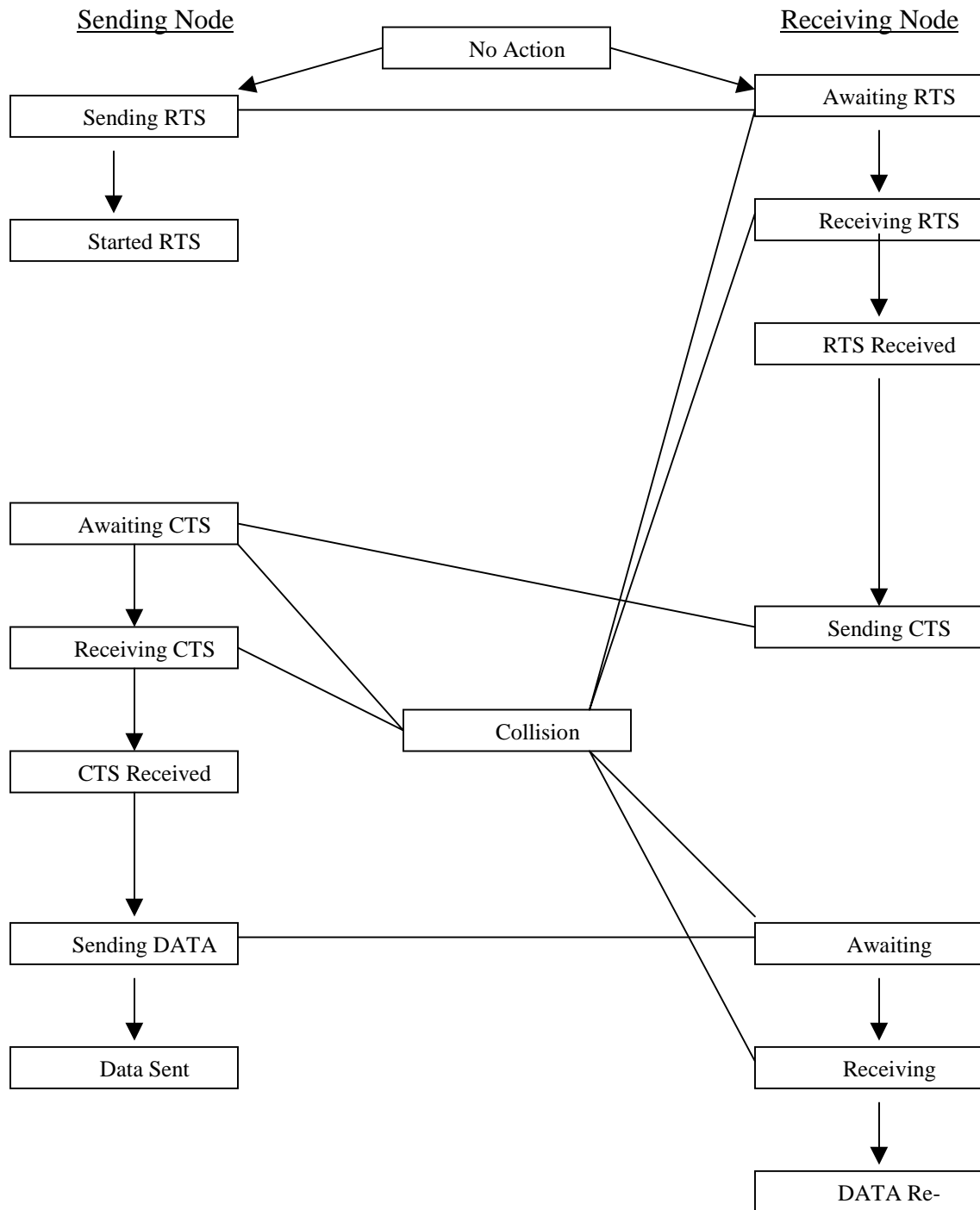
#### Missed Messages

- S-Node does not receive CTS
  - Reschedule RTS
- R-Node does not receive RTS
  - Nothing
- R-Node does not receive DATA
  - Nothing

#### Sending Conflict

- R-Node sends RTS at around the same time as S-Node sends RTS
  - Mark both messages w/ Collision and reschedule both RTS

# State Transition Diagram



## Appendix 2 : Model Configuration File and Traffic Flow

```
# ***** GloMoSim Configuration File *****

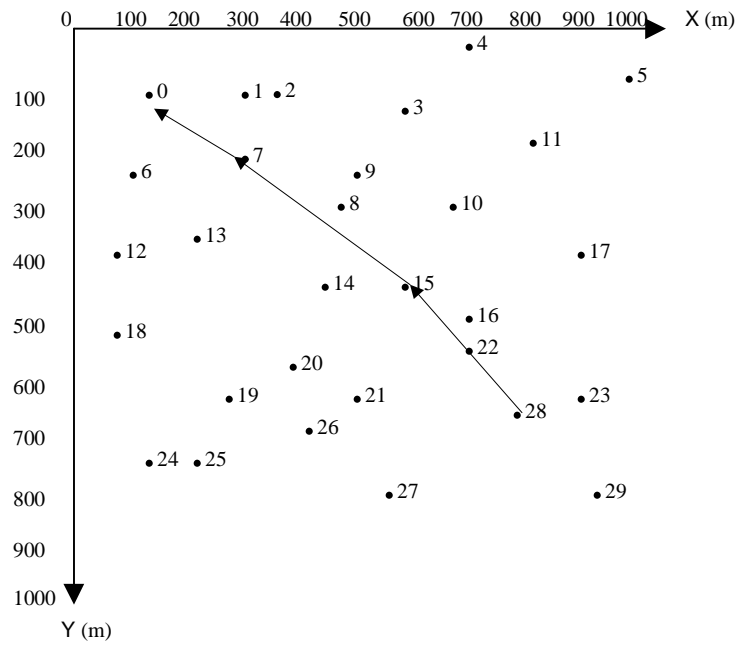
# Glomosim is COPYRIGHTED software. It is freely available without fee for
# education, or research, or to non-profit agencies. No cost evaluation
# licenses are available for commercial users. By obtaining copies of this
# and other files that comprise GloMoSim, you, the Licensee, agree to abide
# by the following conditions and understandings with respect to the
# copyrighted software:
#
# 1.Permission to use, copy, and modify this software and its documentation
# for education, research, and non-profit purposes is hereby granted to
# Licensee, provided that the copyright notice, the original author's names
# and unit identification, and this permission notice appear on all such
# copies, and that no charge be made for such copies. Any entity desiring
# permission to incorporate this software into commercial products or to use
# it for commercial purposes should contact:
#
# Professor Rajive Bagrodia
# University of California, Los Angeles
# Department of Computer Science
# Box 951596
# 3532 Boelter Hall
# Los Angeles, CA 90095-1596
# rajive@cs.ucla.edu
#
# 2.NO REPRESENTATIONS ARE MADE ABOUT THE SUITABILITY OF THE SOFTWARE
FOR ANY
# PURPOSE. IT IS PROVIDED "AS IS" WITHOUT EXPRESS OR IMPLIED WARRANTY.
#
# 3.Neither the software developers, the Parallel Computing Lab, UCLA, or any
# affiliate of the UC system shall be liable for any damages suffered by
# Licensee from the use of this software.
#
# $Id: config.in,v 1.32 2001/04/12 18:35:00 jmartin Exp $
#
# Anything following a "#" is treated as a comment.
#
#####
#
SIMULATION-TIME 15M
#
SEED 1
#
TERRAIN-DIMENSIONS (1000, 1000)
#
NUMBER-OF-NODES 30
#
NODE-PLACEMENT UNIFORM
#
MOBILITY NONE
#
MOBILITY-POSITION-GRANULARITY 0.5
```

```
#
PROPAGATION-LIMIT  -111.0
#
PROPAGATION-PATHLOSS TWO-RAY
#
NOISE-FIGURE  10.0
#
TEMPERATURE  290.0
#
#####
#
RADIO-TYPE      RADIO-ACCNOISE
#
RADIO-FREQUENCY  2.4e9
#
RADIO-BANDWIDTH  2000000
#
RADIO-RX-TYPE    SNR-BOUNDED
RADIO-RX-SNR-THRESHOLD 10.0
RADIO-TX-POWER   15.0
#
RADIO-ANTENNA-GAIN 0.0
#
RADIO-RX-SENSITIVITY -91.0
#
RADIO-RX-THRESHOLD -81.0
#
#####
#
MAC-PROTOCOL    MACA
#
NETWORK-PROTOCOL IP
NETWORK-OUTPUT-QUEUE-SIZE-PER-PRIORITY 100
#
ROUTING-PROTOCOL BELLMANFORD
#
APP-CONFIG-FILE ./scenario.conf
#
#####
#
APPLICATION-STATISTICS  YES
TCP-STATISTICS          NO
UDP-STATISTICS          NO
ROUTING-STATISTICS     NO
NETWORK-LAYER-STATISTICS YES
MAC-LAYER-STATISTICS   YES
RADIO-LAYER-STATISTICS YES
CHANNEL-LAYER-STATISTICS NO
MOBILITY-STATISTICS    NO
#
GUI-OPTION NO
GUI-RADIO NO
GUI-ROUTING NO
```

scenario.conf:

#Flow type	Start	Stop	#elements	Size	Period	Start time	Stop time
CBR	8	17	1000000	435	0.412	0.09	0
CBR	28	0	1000000	271	0.228	0.74	0
CBR	21	9	1000000	437	0.296	0.92	0
CBR	27	19	1000000	471	0.491	0.63	0
CBR	6	4	1000000	470	0.29	0.79	0
CBR	0	4	1000000	220	0.075	1.93	0
CBR	24	25	1000000	360	0.278	0.9	0
CBR	6	17	1000000	220	0.23	1.12	0
CBR	19	0	1000000	421	0.263	1.17	0
CBR	18	8	1000000	474	0.269	1.62	0

### Appendix 3: Example Model Network Topology



## Appendix 4: System ID Model for MAC Layer Simulation

In our project, we applied a statistical model to build our MAC layer abstraction. This model works well with relatively stable traffic. Our assumption of relative stable traffic is reasonable for sensor networks in the sense that researched applications foresee traffic that is generated by a sensor to be periodic. On the other hand, we do want to limit the applicability of our tool in the case of dynamic or sporadic traffic patterns. This appendix briefly discusses our ideas for future work.

In control theory, the process of constructing models and estimating unknown parameters from experimental data is called system identification. System identification is especially suitable for the complex system where interaction is poorly understood and the system is subject to change frequently. Unlike statistical models that predict the future, values based on previous data and system ID models predict the future based not only on previous data but also on current input. These models are more sensitive to changing situations and are potentially applicable to our design.

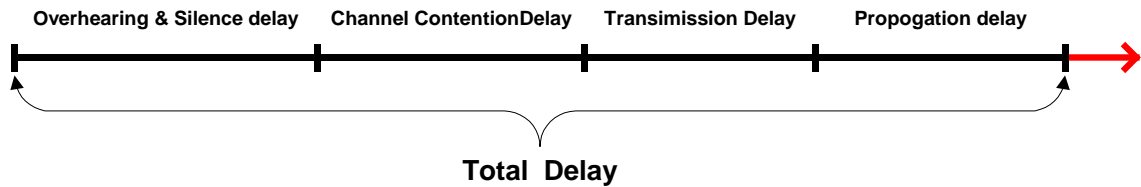
### 1.2 Obtain the System ID Model

First we deem MAC Layer behavior as a multiple-input and multiple-order linear system. Then the dynamic MAC Layer behavior can be modeled as follows:

$$Delay(k) = \sum_{i=1}^{order} A_i Delay(k-i) + \sum_{j=order+1}^{input\ number+order} B_j Input_j(k-1)$$

The  $Delay(k)$  is the time for the MAC layer entity to successfully pass the data-frame to the the receiving MAC Layer entity. This includes the time used in following operation

1. Overhearing & silence delay
2. Contention Delay
3. Transimission delay
4. Propagation delay



All those delays depend on various factors inside the network. For example, the Transmission delay depends on the packet size and bandwidth available. Propagation delay depends on the distance traversed. In wireless networks with open space, contention and noise is much more unpredictable than in traditional wired networks. It is very hard to deduce a general model, which includes every topology and every typical workload inside the network. The main idea behind our approach is to derive the model online for particular network configurations and apply this model to speed up our simulation.

---

Following is the API we implemented for System Identification purposes. It will be used in the model builder component. The Data Collection component also needs to be revised accordingly in order to feed appropriate data to the model builder.

**Function Name:** LSE

**Syntax:** double LSE (FLOAT\_VEC &result,  
FLOAT\_VEC input[ EST\_PARAMS\_LENGTH],  
float parameter[ EST\_PARAMS\_LENGTH] )

**Semantic:** Suppose the model is described by the equation:

$$Delay(k) = \sum_{i=1}^{order} A_i Delay(k-i) + \sum_{j=order+1}^{input\ number+order} B_j Input_j(k-1)$$

The LSE will derive the value of parameters  $A_i$ ,  $B_j$ , according to the experimental data, so that, the equation can estimate the delay with least square errors.

**Parameters:**

FLOAT\_VEC is a vector of type float; the size of the vector is determined by the sample size.  $k$  here is the packet (time) index which goes from 1 to MAX\_NUM.

- **FLOAT\_VEC &result: [in]**

This float vector includes the total delay  $Delay(k)$  values from  $k=1$  to MAX\_NUM

$$result[0][0] = Delay(0)$$

$$result[0][i] = Delay(i)$$

- **FLOAT\_VEC input[ EST\_PARAMS\_LENGTH]: [in]**

This float vector array will include all the factors that affect the total delay. Each element in the array is a float vector and EST\_PARAMS\_LENGTH = order + number of inputs.

input[i]: a float vector ,includes the delay  $Delay(k-1-i)$  from  $k=1$  to MAX\_NUM -i-1

$$input[i][0] = 0;$$

.....

$$input[i][i] = 0;$$

$$input[i][i+1] = Delay(0);$$

$$input[i][i+2] = Delay(1);$$

.....

Input[j]: a float vector , is the input<sub>j</sub>(k-1) ( EST\_PARAMS\_LENGTH > j > order )

$$input[j][0] = Input_j(0)$$

.....

$$input[j][i+1] = Input_j(i+1);$$

$$input[j][i+2] = Input_j(i+2)$$

.....

- **float parameter[ EST\_PARAMS\_LENGTH ] : [out]**

parameter[0] to parameter[order-1] is the order parameter  $A_i$

parameter[order] to parameter[order + # of inputs-1] is the inputs parameter  $B_j$

**Return Value:** returns the goodness of fit. (type double)

$$\sum_{i=0}^{\max\_index} \frac{(observedDelay(i) - EstimatedDelay(i))^2}{Size - 1}$$