MAGICWEAVER

An Agent-Based Simulation Framework for Wireless Sensor Networks

Sovrin Tolia
University of Maryland
Baltimore County
1000 Hilltop Circle
Baltimore, Maryland
stolia1@cs.umbc.edu

Anupam Joshi
University of Maryland
Baltimore County
1000 Hilltop Circle
Baltimore, Maryland
joshi@cs.umbc.edu

Timothy Finin
University of Maryland
Baltimore County
1000 Hilltop Circle
Baltimore, Maryland
finin@cs.umbc.edu

ABSTRACT

Distributed sensor networks have been the focus of considerable research during the past few years. Current research on wireless sensor networks can involve a number of issues, including hardware constraints, communication and routing issues, data management problems, and software engineering principles. To clearly understand their characteristics, we require development environments to design them, explore architectural alternatives, and simulate their performance. MagicWeaver is an agent-based simulation framework for wireless sensor networks. It provides abstract models for sensor tasking and communication, device constraints, network topologies and the physical environment. It also supports a special class of applications characterized by streaming data-feeds. MagicWeaver illustrates ways in which agents paradigm extends it's flexibility to incorporate teamwork and coordination between sensor agents. In this paper we discuss MagicWeaver's underlying design rationale and provide it's current implementation status. We also report on results obtained while performing scalability tests using the JADE agent platform.

Keywords

Sensor Networks, Simulation, Test-bed, Performance issues

1. INTRODUCTION

A sensor node is a small form-factor hardware device typically consisting of a microcontroller, radio module and environmental sensors. We envision these to span from extremely resource-constrained "Smartdusts"[14] type devices to resource-rich devices with sufficient memory and compute power. A sensor network is a collection of such devices entrusted with the task of sensing and communicating events from the environment. There is often a distributed set of resource-rich base stations that perform information aggregation and advanced decision making.

Typical characteristics of wireless sensor networks include a large number of nodes deployed to gain sufficient coverage, susceptibility to failures, non-replenishable energy sources and communication using wireless links. These characteristics pose numerous problems that have to be solved before sensor networks become socially viable. Since these devices are battery-powered, energy efficiency becomes an important goal for our algorithms. Their sheer numbers makes scalability one of the key design issues. Noisy and loss-prone wireless environments warrants data propagation schemes that ensure that the utility[6] of the network is not lost. These challenges, once solved will create a lot of application areas where they can be applied.

To answer some of these challenges, we will have to experiment with different algorithms and techniques. For this, we require development environments to design them, explore architectural alternatives, and simulate their performance. It is important to evaluate these systems using simulations as research on the hardware devices is still not mature and hardware devices are not easily available. Moreover, their cost makes them prohibitive to purchase in huge numbers and evaluating on actual devices would be really difficult. MagicWeaver is an agent-based simulation framework towards this end. It provides abstract models for sensor network tasking and communication, device constraints, network topologies and the physical environment. It also supports a special class of applications characterized by streaming data-feeds. MagicWeaver illustrates many ways in which agents paradigm extends it's flexibility to incorporate teamwork and coordination between sensor agents. In this paper we discuss MagicWeaver's underlying design rationale and provide it's current implementation status. We also report on results obtained while performing scalability tests using the JADE agent platform.

In the context of this framework, we have used the weak notion of agency as defined in [19]. Accordingly, agent-based systems are characterized by the properties of autonomy, social ability, reactivity and pro-activeness. Based on this, we find a sensor is like an agent. Both are designed to operate without direct involvement of humans or others and have control over their state, thus exhibiting autonomy. Both need to communicate and coordinate with their counterparts to achieve the desired goal, thus exhibiting social ability. Both are designed to take some action in response to cer-

tain environmental events, thus exhibiting reactivity. Sensors often balance the trade-off between conserving available resources and sensing events, thus exhibiting pro-activeness. Further we find that communication and coordination are two key research aspects, common to both. This high degree of conceptual correlation between sensors and agents makes the agent paradigm suited for this kind of a framework.

In the next section we survey related research work. In section 3, we discuss the design and implementation of MagicWeaver. In section 4 we provide experimental results obtained while performing the scalability tests on the framework. In section 5 we discuss the potential benefits of using MagicWeaver. Section 6 concludes the paper with the features that we plan to implement in future.

2. RELATED WORK

SensorSim[16] is a simulation framework for sensor networks built by providing extensions to ns2 [9], event-driven network simulator. It provides power usage model and supports hybrid simulations. SensorSim is not designed for extensibility as much as MagicWeaver is.

SensorWare[4] is a middleware platform providing mobile code support for dynamic management of a sensor node. It's focus is to enable dynamic runtime configuration of sensors whereas MagicWeaver provides a flexible design-time environment for sensor network simulation.

Our work has been motivated by various hardware and software research projects in sensor networks. The Hardware research comprising of WINS project[5], PicoRadio project[18], Smartdust project and Motes Project[13] helped us in defining abstract models for sensor device constraints. Study of various Data Propagation Schemes developed ([7],[8], [12],[2]) gave us concrete ideas for building models for data propagation.

The Telegraph project[15] deals with query processing over data-feeds that originate from adhoc sensor networks. It deals with design of query operators and a proxy-based architecture for handling huge data streams. In contrast, MagicWeaver supports hierarchical processing of streaming data-feeds, based on the availability of resources on the data-path.

Software Agents paradigm has been applied to build simulation frameworks in the past [11, 17] but to the best of our knowledge, agents paradigm has not been specifically applied to model and simulate wireless sensor networks.

3. MAGICWEAVER

3.1 Overview

It is a common observation that sensor network consists of five modular units: sensor tasking, data collection, data dissemination, data aggregation and data analysis. There is a wide spectrum of algorithms that can be applied to realize these modules. To flexibly simulate these heterogeneous techniques, a framework is required that provides abstract definitions and models for them. MagicWeaver provides such a framework.

It deploys a multi-agent system where each software agent acts as a sensor device. These agents coordinate and communicate with each other to pass sensed information back to the distributed base stations. The framework in itself does not provide any particular model to be used. The designer of the sensor network is expected to define his/her own models and the framework will use these user-defined instances to simulate the network.

Sensor Networks of considerable size can generate huge data streams. For such networks, data transmission to the base station on an as-is basis for storage, processing and analysis is a problem. To solve this problem, MagicWeaver provides a hierarchical scheme to do in-network data processing, data reduction and fusion, based on the availability of resources at the nodes in the data path.

In addition to providing abstract models and support for streaming applications, it provides helper components for carrying out the simulations effectively. These include, SensorView, a graphical tool for displaying the network topology and the condition of each sensor node, DataView, a graphical tool for displaying the sensed data that is available for processing and QMView, a quantitative measure of energies and data transmission statistics at each node in the network.

3.2 Sensors As Agents

Use of the agents paradigm extends it's flexibility to incorporate teamwork and coordination between sensor agents. MagicWeaver illustrates several ways in which it provides flexibility to the sensor network designers. The object orientation in the language Java, provides flexibility in terms of code modularity. It helps define abstract models and clean interfaces for designers to implement their own basic schemes.

We have sensor devices like WINS Sensoria nodes[1] that are computationally powerful and have large memory. As hardware research progresses, we will have devices with more powerful processors, large storage capacities and long battery lives. This will create sensor networks in which sensors are intelligent, adaptive, dynamically configurable and reactive. They will be able to take intelligent decisions about occurrence of abnormal conditions. They will be able to alter their functioning mode based on resource availability and external conditions. Further, they will be able to switch between tasks when instructed and will be able to take actions when abnormalities are detected. To realize such sophisticated sensor networks, we need to simulate their behaviour to carefully understand various engineering trade-offs. Since these properties have already been studied in the context of software agents, MagicWeaver leverages this work in it's application to sensor networks.

Teamwork between sensor agents would be important to answer non-trivial queries. For example, there might be queries of the form, "Give me average temperature in the region for the next 1 hour?". For answering such queries, sensors would have to team up and exchange information. MagicWeaver allows such teamwork concepts to be flexibly built into the sensor agents.

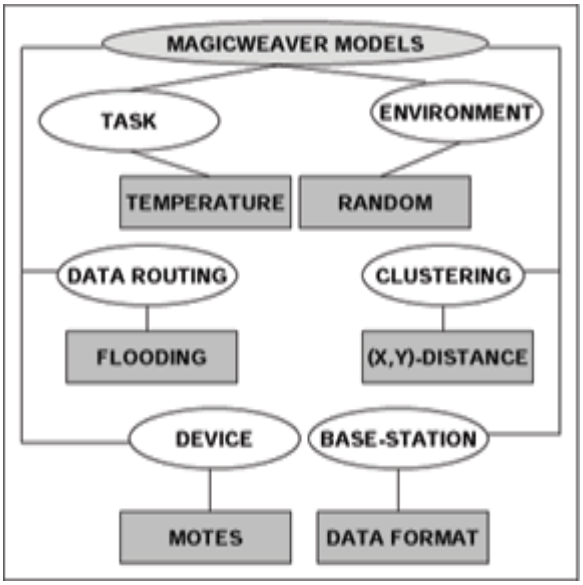

Figure 1: Abstract Models and Example Instantiations

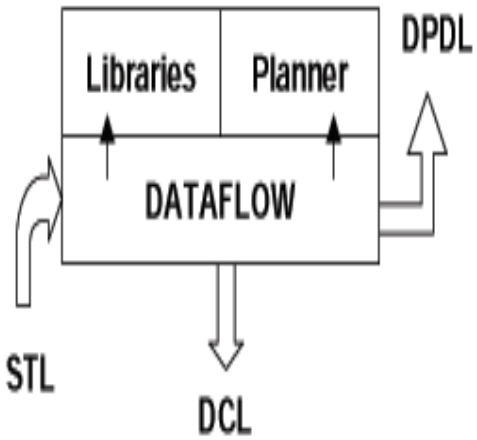

Figure 2: Data Processor for Streaming Applications

Sensor would have to coordinate their actions in the network to conserve energy. For example, we will have sensor networks with motion and video sensors for detecting movement and capturing images of moving objects. When there is no movement in the region, the video sensor would waste a lot of battery and bandwidth in sensing and transmitting these images. However, when the motion sensors detects movements, it will actuate the video sensors to start capturing and transmitting images. Thus, coordination between sensors would help conserve energy.

We envision that in future, sensor devices will have the capability to run agents on them and this would help us in creating actual agent-based sensor systems. Since this is not a viable alternative presently, MagicWeaver provides an agent-based simulation framework to experiment with such advanced agent-oriented techniques.

3.3 General Purpose Models

Based on the modular units mentioned earlier, MagicWeaver provides abstract models for sensor tasking, data collection and data dissemination. All these models were derived by observing the hardware characteristics and limitations of sensor devices, the type of sensing they do and way the they communicate within a network.

The abstract models currently provided by MagicWeaver include the device constraint model, data propagation definition model, sensor task and tasking model, environmental model, cache model, energy consumption model, location model, base station data model and network topology model. Figure-1 gives the various models supported by the MagicWeaver.

The device constraint model defines the hardware limitations of sensors. Memory, processing power and battery life are the typical constraints that are modelled. The data propagation definition model provides means to define data routing schemes. The sensor task and tasking model defines attributes and execution policies associated with the tasks. The environmental model incorporates definitions to generate external data events. It is designed to simulate the actual physical environment that the sensors are expected to sense.

The cache model provides an abstract definition of the caching strategy used by sensor nodes. The energy consumption model provides quantitative measures for energy consumed by different sensor actions. The location model provides means to associate location information for sensor nodes. The base station data model defines a format for publishing data. It is used to interface external applications with the sensor network. The network topology model provides a means to define node-to-node connectivity within the sensor network.

The sensor network designer is expected to define specific instances of these models for the environment that he is trying to simulate. MagicWeaver uses these user-defined instantiations to simulate the behaviour of the network.

3.4 Support for Streaming Applications

A large sensor network can generate huge streams of data. Persistent storage of those streams can be extremely costly. Moreover, the wireless network will not be able to handle such huge streams, data transmission will drain the sensor's energy and it might not be possible to do real-time data processing at the computational resource.

To address these problems in streaming environments, MagicWeaver provides support for in-network data processing, data reduction and data fusion. It is based on resource-based hierarchical processing of data as it travels through the network. Each node in the network does data reduction or data transformation based on the available resources. This scheme leverages the resources encountered on the data path in reducing data streams.

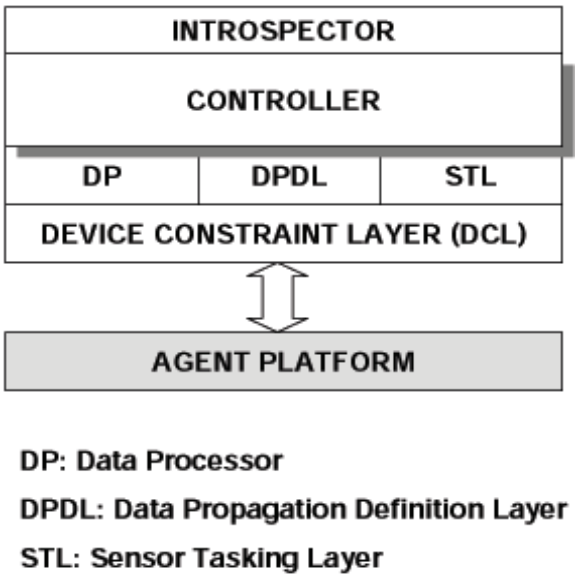

Figure 3: Sensor Node Architecture

For example, when there is a fire in the building, the fireman would like to know the heat distribution in the different floors. Computing the heat distribution requires solving partial differential equations and this is highly processor intensive. The sensors sense the temperature and can do a local coordination to transmit only the representative values of the sensed data-set. The fireman's PDA, upon receiving the data, will compute the distribution parameters and correlation matrix and transmit it to the cluster of resource-rich machines. They compute the heat distribution in the building and send it to the fireman. This scenario illustrates how each device in the data-path does data transformation based on resource availability. It helps in reducing data streams, while ensuring that the utility of the data is not lost.

To realize the above scenario, MagicWeaver simulates a network of heterogeneous nodes. A specific instance of the environment model is used to generate events at a high data rate. The data is processed at the individual nodes by the data processor component. It consists of three modules. The Library, that stores processing routines that can be applied to the data. The Planner, that specifies the plan for data processing and maintains resource requirements for different data processing steps. The Dataflow, that links the planner with the library for carrying out the data transformation and record the resources consumed during the process. Figure-2 provides the representation of the data processor component.

3.5 Agent Components

The three prime components in the MagicWeaver framework are the sensor agent, the environmental agent and the network topology agent. The helper components include the logger agent and the base station agent.

3.5.1 Sensor Agent

Figure-3 shows the design of a sensor agent. It has a layered architecture with each layer working under the constraints imposed by some of the models. The agent platform hides all the low level transport details from the sensor agent. The device constraint layer(DCL) imposes resource restrictions, based on which all sensor agent actions are taken.

The data propagation definition layer(DPDL) embodies the data propagation definition model and the cache model. The message passing primitives are drawn from the underlying agent platform. Moreover, it uses the services of the network topology agent to establish node-to-node connectivity.The sensor tasking layer(STL) embodies the task and tasking policy model. The agent performs sensing actions based on the definitions provided by these two models.

Controller is analogous to the microcontroller present in the hardware device. It coordinates the actions of the sensor agent and takes dynamic decisions to switch between node states(active and parked) and switch between sensing tasks based on the tasking policy. The introspector monitors the internal state of the sensor agent and updates the logger agent.

3.5.2 Network Topology Agent

The network topology agent embodies the location model and the clustering model. It is responsible for assigning unique locations to the sensor agents in the network and running a clustering algorithm to find out agent's neighbour connectivity.

3.5.3 Environmental Agent

The environmental agent embodies the environment model. It simulates the actual physical environment, regularly generating events for the sensor agents to sense. It is configured for generating events of different types based on the policy definition.

3.5.4 Base Station Agent

The base station agent expects data to be published to it by the sensor agents, in the format specified by the base station data model. It provides a graphical tool, DataView for displaying the data gathered by the sensors from the network.

3.5.5 Logger Agent

The logger agent receives messages about the internal state of the sensor agents. It provides a graphical tool, SensorView, to display the network graph and reflect the state data for the sensor agents. The QMView tool in it, provides simulation data in a suitable format for analysis.

MagicWeaver provides a centralized architecture for network topology management and maintaining a view of the sensor network. The centralized architecture does not impose any limitation on the framework, as the organization of the network topology is done only once, during startup. The dynamic reconfiguration of the network topology would be handled by the nodes themselves and does not require support from a central network topology agent. The communication and coordination between sensor agents follows a completely distributed architecture. In essence, a sensor network is modelled as a multi-agent system in which agents are communicating and coordinating with each other to achieve

the desired goal. With this architecture, it would be very flexible to test the performance of different algorithms pertaining to different facets of sensor networks.

MagicWeaver has been implemented using Java for model definitions and Java Agent Development Environment[3] as the underlying agent platform. It exposes an object-oriented application programming interface that the sensor network designers can use to incorporate their models into the runtime system. XML is used for all policy definitions and inter-agent communication. JADE is a FIPA compliant agent development platform that hides all the low level transport details from the agent programmer. The cooperatively scheduled behaviour-oriented paradigm of JADE greatly helps in modelling autonomous actions of the sensor agents.

3.6 Runtime Environment

To illustrate the use of MagicWeaver, we have created specific instances of various models required by it. The sensor is modelled as a Berkeley Motes[13] device. The snippet below shows a sample device configuration.

```
<?xml version=''1.0'' encoding=''UTF-8''?>
<sensordeviceprofile>
   <devicename>Motes</devicename>
   <devicetype>Wireless Sensor</devicetype>
   <memory>
      <primary>
          <maxmemory>512</maxmemory>
          <memoryunit>B</memoryunit>
      </primary>
      <secondary></secondary>
   </memory>
   <cpu>
      <cputype>AMD Processor</cputype>
      <cpuspeed>20</cpuspeed>
      <cpuspeedunit>MHz</cpuspeedunit>
   </cpu>
   <battery>
      <batterytype>Ni-CD</batterytype>
      <batterymaxenergy>30</batterymaxenergy>
      <batteryenergyunit>Joules</batteryenergyunit>
   </battery>
   <networksupport>
      <wireless>
         <communicationrange>10</communicationrange>
         <commrangeunit>feet</commrangeunit>
         <bandwidth>19.2</bandwidth>
         <bandwidthunit>Kbps</bandwidthunit>
         <tranceiverpower>
            <receiving>0.0135</receiving>
            <transmitting>0.036</transmitting>
            <idle>0.000015</idle>
         </tranceiverpower>
         <amplifierpower>
            <receiving>0.0</receiving>
            <transmitting>0.0</transmitting>
            <idle>0.0</idle>
         </amplifierpower>
      </wireless>
    </networksupport>
</sensordeviceprofile>
```

Flooding and Gossipping represent the underlying data propagation schemes. In Flooding, sensor nodes sends the data-item to all its neighbours, which in turn does the same action. In Gossipping, the sensor node sends the data-item to only one of its neighbours, chosen randomly.

The simulation is divided into two phases. During the first phase, each sensor agent contacts the network topology agent for location and neighbor connectivity information. The network topology agent generates neighbor information using a distance-based clustering algorithm. Randomly it flags some sensor nodes as having base station connectivity and assigns the environmental agents to be polled.

During the second phase, the sensors start sensing based on the tasking policy. The sensing action is simulated as a message send and receive between the sensor agent and the environmental agent. When the sensor senses some event, the event data-item is stored in the cache. Periodically it sends out these sensed values based on the underlying data propagation scheme. The cache follows the caching strategy outlined in the cache model instance. To maintain information about the network and sensor agent states, the introspector component in each of the sensor agents sends internal state messages to the logger agent.

3.7 Application Areas

MagicWeaver simulation framework will be useful in simulating different aspects of Sensor Networks. The designer could potentially compare the performance of different data propagation schemes like Flooding, Gossipping, SPIN and Direction Diffusion. The framework allows investigation of varying sensor network lifetimes based on different sensing strategies, data transmission, device constraints and cache optimizations. The coupling between these aspects can also be studied to determine the optimum configuration for a particular deployment of sensor network.

For example, to compare the performance of MagicWeaver with ns-2, we are simulating the following sensor environment. A network of sensors is deployed in a building. Each sensor has the capability to sense temperature and pressure. They are configured to use either Flooding or Gossipping as the underlying data-propagation scheme. The frequency of sensing and data transmission is externally configurable. Each sensor device is modelled as a Berkeley Mote. We have used a simple distance based clustering algorithm to determine the neighbor connectivity in the network. A subset of nodes in the network are configured to have base station connectivity.

During a fire(detected by abnormal increase in temperatures), the sensors become active and start sending their sensed data values using multi-hop routing. When the node receives a data-item, it either forwards it to its neighbors or sends it to the base-station(if it has that connectivity). The DataView monitoring tool displays all the information received and displays the number of active agents sending information to it. Intermittently, the introspector component in each of the sensor agents sends internal state messages to the Logger Agent. Vital pieces of information include, the energy levels, bytes sent, bytes received and its neighbour list. SensorView graphical tool displays a graph view of the network and displays this state information.

This kind of a network setup would help determine the temperature and pressure readings in different regions of the building and the internal state information would be useful in finding out the lifetime of the sensor network. An obvious

enhancement to this scenario would be introducing failures in the network. The snippet below shows an example of the tasking policy used in the simulation.

```
<?xml version=''1.0'' encoding=''UTF-8''?>
<taskingpolicy>
    <task>
        <taskname>Temperature</taskname>
        <taskclass>TemperatureSensorTask</taskclass>
        <taskschedule>
            <frequency>30000</frequency>
            <unit>milliseconds</unit>
        </taskschedule>
        <access_mode>PUSH</access_mode>
    </task>
    <task>
        <taskname>Pressure</taskname>
        <taskclass>PressureSensorTask</taskclass>
        <taskschedule>
            <frequency>45000</frequency>
            <unit>milliseconds</unit>
        </taskschedule>
        <access_mode>PUSH</access_mode>
    </task>
</taskingpolicy>
```

4. EXPERIMENTAL ANALYSIS

4.1 Overview

To have wide geographical coverage and to provide resiliency from node failures, sensors are deployed in large numbers in the network. Further, this will generate high network traffic. This requires it's simulation frameworks to be scalable both in terms of number of devices that can be simulated and the data traffic that can be handled.

MagicWeaver is built on top of JADE, leveraging it's messaging and deployment infrastructure. JADE consists of a Main Container or the Platform that provides mandatory FIPA[10] compliant services and peripheral containers that are connected with the platform. Both the platform and the main containers are capable of hosting agents. The underlying messaging infrastructure in JADE has three types of messaging. Two agents within the same container exchange messages using events. Two agents in different containers exchange messages using Java RMI and two agents in different platforms communicate using protocols like HTTP/IIOP.

The number of sensor devices that can be simulated by the framework depends on the footprint imposed by sensor-specific behaviour of these agents and the agent platform. The network traffic that is handled by the framework depends on the capability of the underlying message transport protocol. To investigate the constraints imposed by the underlying agent platform, we carried out scalability tests on the platform and report the results here.

4.2 Test-Bed

In these experiments we are investigating two issues. Given a fixed number of messages per minute, per agent, how many agents can a system support? Secondly, given a fixed number of agents, what is the message complexity that the system handles reasonably, beyond which substantial message losses are encountered or the system breaks.

Hosts	Messaging	Stable	Unstable	Breakdown
1	Inter-Container	500	750	1000
1	Intra-Container	750	1000	1500
3	Inter-Container	1000	1250	1500
3	Intra-Container	1500	2000	2500

Table 1: Maximum Agents Threshold

We used two configurations, one in which all agents are deployed on a single host and the other in which all agents are equally distributed on three hosts. We restricted the number of hosts to three, as our goal is to make MagicWeaver scalable using the least number of hosts. These hosts were a part of the beowulf cluster. Each node in the cluster has a Pentium III 598 MHz dual processor and 1 GB memory. The nodes ran Linux 2.4.7-10 and were connected using dual-bonded 100 Mbit ethernet links.

The simulation of agents in our tests was based on JADE deployment architecture. Each peripheral container was configured to host 50 agents and these were distributed equally across the available hosts. To ensure that messages are sent uniformly during the time period (one minute intervals), a clock skew was introduced during the startup of agents. We configured each sensor agent to have 4 neighbours, to which it sends messages whenever it senses events. Forwarding of data messages was disabled to keep the number of messages sent bound to the theoretical message complexity.

The experiments were carried out for intra-container and inter-container messaging. The threshold values obtained were categorized into three states, Stable, Unstable and Breakdown. In the stable state, atleast 95% of the agents in the system are active and the message loss is less than 5%. In an unstable state the number of active agents ranges between 90% and 95% and the message loss is between 5% and 15%. The breakdown state is reached when the number of active agents are less than 90% or the message loss is greater than 15%. To determine these boundary values, we carried out initial stress tests. These tests revealed that the system within the given boundary conditions, tends to remain in the same state with no performance degradation. For the purpose of this simulation, an Active Agent is one that is sensing and reporting data according to the configured specifications.

4.3 Maximum Agents Threshold

Each sensor was configured to generate a sensor event once per minute and this was sent to it's 4 neighbours. This frequency was carefully chosen to ensure that the network is not overwhelmed with messages as the number of agents increases. Each simulation was allowed to run for 20 minutes. The total number of active agents and messages sent and received by the them were accumulated for analysis and averaged to get the final thresholds. The performance of the system was evaluated, by monitoring the number of agents that were active during the simulation and by comparing the actual messages sent and received with the theoretical bounds.

Table-1 provides the results obtained through the simula-

tion for both inter-container and intra-container messaging. The agents were increased in the system in multiples of 250. A single host setup with inter-container messaging was stable with 500 agents. With 750 agents, the number of active agents decreased and message losses increased. At 1000 agents, only 12% of them were active. With intra-container messaging, there was a 50% increase in the number of agents supported by the system in different states. The 3-host setup with 1250 agents went into the unstable state as it experienced both messages losses and decline in agents that were active. With 1500 agents using inter-container messaging, only 16% of the agents were active. With 2000 agents, the system went into an unstable state and there was a complete failure with 2500 agents.

In the unstable state, containers got disconnected from the platform. This led to the execution of a connection recovery phase in which additional messages were sent, posing additional message overhead on the platform.

We find that intra-container messaging provides support for greater number of agents(50%) in the system as compared to inter-container messaging. During the simulations we observed that the minimum amount of memory available on the hosts was 400MB. This clearly indicates that performance degradation is happening due to the increasing number of messages in the system. Thus, both the number of agents and the number of messages are important parameters in testing the scalability of the system.

4.4 Message Complexity Threshold

Based on the results of the previous experiment, we carried out this test with 300 and 600 agents in the system for both intra-container and inter-container messaging. The goal was to find the threshold values for the number of messages that the underlying platform and containers can support. The messages were increased by a factor of 4 messages per agent per minute. Table-2 and Table-3 summarizes the results obtained during the simulations. In these tables, values for the stable, unstable and breakdown state are expressed as **per agent per minute**.

From the results, we find that increasing the number of messages decreases the number of active agents in the system and increases the messages losses, taking the system from stable to breakdown state. As the number of hosts increases, the number of messages that are handled also increases. Again in this case, intra-container messaging performs better than inter-container messaging. Further, as the number of agent increases, the system is able to support lesser messages per agent per minute, though the total messages per minute is more.

In this experiment also, we found that the nodes had enough memory available(minimum 400MB) during the simulations. Hence the real bottleneck is created due to the large number of messages passed in the system. In the burst mode (messages are sent at the same time), the system performance degrades much faster.

5. DISCUSSION

Our implementation demonstrates the flexibility for incorporating various models in the simulations. Sensor network designers can try different combinations of schemes to get the most optimal performance for their algorithms. The framework acts as a runtime environment linking various models to simulate the network. Traditional network simulators provide complex models for users to adapt in order to simulate their environment. MagicWeaver leaves this complexity to the designer.

Hosts	Agents	Stable	Unstable	Breakdown
1	300	12	16	24
1	600	8	12	16
3	300	16	24	40
3	600	12	16	20

Table 2: Message Complexity Thresholds with Inter-Container Messaging

Hosts	Agents	Stable	Unstable	Breakdown
1	300	16	14	40
1	600	12	16	20
3	300	24	32	40
3	600	16	20	24

Table 3: Message Complexity Thresholds with Intra-Container Messaging

Based on the results obtained through the simulations, MagicWeaver will be able to provide a robust runtime environment configuration for sensor agents. It will be able to create an optimized deployment of agents for the clustering policy specified by the user. The thresholds for the number of agents and message complexity supported by the JADE platform are encouraging. This, coupled with the message transmission optimizations by MagicWeaver, would enable it to simulate large sensor networks with huge data streams. To counteract some of the scalability limitations imposed by the underlying agent platform, we would be modifying the platform to include just the components neccessary for the MagicWeaver framework. Some of the components that are not required include Security modules, mobile code support modules and communication modules like Corba IIOP and Java RMI.

An agent-oriented approach to building such a framework provides a foundation for incorporating agent theories like Teamwork and Coordination between sensor agents. It would be difficult to include such principles in an Object-Oriented approach for building such simulation tools.

6. CONCLUSION

In this paper, we have described MagicWeaver, an agent-based simulation framework for wireless sensor networks. The key feature of MagicWeaver is flexibility, both in terms of incorporating user-defined models and agent-based properties like teamwork, adaptiveness and intelligence in sensors.

We are working on simulating the performance of hierarchical data processing scheme for streaming applications. The goal is to compare sensor network lifetime under this scheme with the one in which all the processing is done at the central node. To demonstrate the correctness of the quantative

measures provided by the framework, we are comparing it's performance with ns-2 and Sensorsim.

On the modelling side, MagicWeaver is to be extended to provide support for querying the sensor network. Specifically we are looking at query routing and dynamic teamwork formation between sensor agents to answer queries. The framework supports Directed Diffusion as one of the underlying data routing schemes.

7. REFERENCES

[1] G. Asada, M. Dong, T. Lin, F. Newberg, G. Pottie, W. Kaiser, and H. Marcy. Wireless integrated network sensors:low power systems on the chip. *Proceedings of the 1998 European Solid State Circuits Conference.*

[2] M. Bawa and M. Anant. Locating objects over a wireless sensor networks.

[3] F. Bellifemine, A. Poggi, and G. Rimassa. Developing multi agent systems with a fipa-compliant agent framework. In *Software - Practice And Experience*, number 31, pages 103–128, 2001.

[4] A. Boulis, A. Savvides, and M. Srivastava. Sensorware: A middleware supporting mobile distributed computing for sensor networks.

[5] K. Bult, A. Burstein, D. Chang, M. Dong, and W. Kaiser. Wireless integrated microsensors. In *Proceedings of Conference on Sensor s and Systems (Sensors Expo). Anaheim, CA, USA*, pages 33–38.

[6] J. Byers and G. Nasser. Utility-based decision-making in wireless sensor networks. Technical Report 2000-014, 1 2000.

[7] D. Estrin, L. Girod, G. Pottie, and M. Shrivastava. Instrumenting the world with wireless sensor networks. In *International Conference on Acoustics, Speech and Signal Processing ICASSP 2001*, 2001.

[8] D. Estrin, R. Govindan, and C. Intanagonewawat. Directed diffusion, a scalable and robust communication paradigm for sensor networks. In *International Conference on Mobile Computing and Networking*, pages 56–67, august 2000.

[9] K. Fall and K. Varadhan. Ns notes and documentation. http://mach.cs.berkeley.edu.

[10] FIPA. The foundation for intelligent physical agents. http://www.fipa.org.

[11] S. Giroux, P. Marcenac, S. Calderoni, D. Grosser, and J. Grasso. A report of a case study with agents in simulation, 1996.

[12] W. R. Heinzelman, J. Kulik, and H. Balakrishnan. Adaptive protocols for information dissemination in wireless sensor networks. In *Mobile Computing and Networking*, pages 174–185, 1999.

[13] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for networked sensors.

[14] J. M. Kahn, R. H. Katz, and K. S. J. Pister. Next century challenges: Mobile networking for "smart dust". pages 271–278.

[15] S. Madden and M. J. Franklin. Fjording the stream: An architecture for queries over streaming sensor data. In *ICDE*, 2002.

[16] S. Park, A. Savvides, and M. B. Srivastava. Sensorsim: A simulation framework for sensor networks. In *The Third ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, 2000.

[17] S. G. Pierre. Modelling and simulating self-organised critical systems.

[18] J. Rabaey, J. Ammer, J. da Silva Jr., and D. Patel. Picoradio: Ad-hoc wireless networking of ubiquitous low-energy sensor/monitor nodes. In *Proceedings of the IEEE Computer Society Annual Workshop on VLSI (WVLSI'00).*

[19] M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.