SIMULATION ENVIRONMENT SHOWING ENERGY CONSERVATION FOR AGILLA MIDDLEWARE

[1]Dr. G. Mahadevan, [2]Prof. Ms.Nirmala.S, [3]Pradeep N
[1]Prof., [2]Research Schola, [3]4th Semester, M-Tech,
Dept. of CSE,AMCEC , Bangalore ,Karnataka India
Email:[1]g_mahadevan@gmail.com, [2]esthernirmala70jc@yahoo.co.in, [3]pradeepnmurthy8@gmail.com

Abstract: Wireless sensor networks are turning out to be progressively attractive to researchers and industries, due to extensive variety of applications. Middleware serves to overcome the gap between the high-level requirements of the application programs and underlying operations of WSN. Agilla is a portable agent middleware that encourages the rapid deployment of adaptive applications in wireless sensor networks (WSNs). Agilla allows users to create and inject special programs called mobile agents that coordinate through local tuple spaces, and relocate across the WSN performing application-specific tasks. Agents can alterably enter and exit a network and can autonomously clone and migrate themselves in response to environmental changes. AgentInjector can be used to inject agents into a network. A simulation is performed on the mobile agents running on Agilla middleware intended for sensor networks. The simulations are performed using TOSSIM assuming that Agilla middleware is installed on the sensor nodes which are running TinyOS operating system. Simulated is performed on different agents corresponding to various functions and the time taken for agent software to run in the simulated environment is measured. The results of migration delay and reliability in the simulation of agents is calculated.

Keywords: Wireless sensor network, Middleware, Agilla, Simulation, Mobile Agents

1. INTRODUCTION

Wireless sensor networks (WSNs) comprises of small sensors deeply embedded within the environment. WSNs must deal with highly dynamic environments. For instance, while a fire tracking network deployed in a forest may remain inactive most of the time, a wildfire may break out and spread unpredictably, rapidly triggering numerous network activities. Therefore, WSN applications need to be highly flexible and adaptive, which places an additional burden on the application developer.

Middleware solutions are developed as the link between application and low level operating system to solve many wireless sensor network issues, or the loss of coverage occurs. There are diverse sorts of middleware's developed for distinctive purposes. Middleware's can be classified considering their programming approaches, as database approach, virtual machine approach, adaptive approach and agent based approach.

The Database methodology regards the entire sensor deployment as an appropriated database. Generally it has a simple interface to use, like SQL that make enquiries to gather target information. It is great at consistently queries, but does not support actual time applications, so at times it just gives estimated results.

Virtual machine middleware methodology is utilized to reducing general force and asset utilization. The system comprises of virtual machines and translators. Designers compose applications into little modules, and infusing and

conveying modules through system, and finally virtual machines interpret the modules to implement application

For some predefined reason, application specific middleware could alter system setups as per application necessities. It has a structure that deliveries different system measures by picking suitable protocol in its network protocol stack. Diverse sensors combination and system setups give distinctive performance of QoS to meet related application requirements.

Agent based approach mobile agents, support migration and use a local shared memory to provide local communication. The main feature of mobile agent middleware is the applications are treat as modules for injection and distribution through the network using mobile codes.

Since mobile agent middleware has not been transferred to the simulation environment, software (Agent Injector) is used to perform simulation of Agilla middleware on TOSSIM [10] which is a simulation platform of WSN with nodes running TinyOS [11]. These agents, running on the Agilla middleware which is installed on the motes, are allowed to perform different tasks and the performance of the system when running this software is evaluated.

2. RELATED WORK

A traditional approach for WSN adaption is to reprogram it over the wireless network. Systems that enable this can be divided based on what is reprogrammed, that is, native code, interpreted code, or both. Two systems that reprogram native code are Deluge [Hui and Culler 2004] and MOAP [Stathopoulos et al. 2003]. They are designed to transfer large program binaries, enable the network to be arbitrarily reprogrammed, but incur high overhead and latency.

To address this, SOS [Han et al. 2005], Contiki [Dunkels et al. 2004], and Impala [Liu and Martonosi 2003] are systems that enable partial reprogramming of binary code by providing a micro-kernel that supports dynamically linked modules. Since modules are relatively small, the cost of reprogramming is lower.

Systems that reprogram interpreted code include Mat´e [Levis and Culler 2002], Application-Specific Virtual Machines (ASVM) [Levis et al. 2005], Melete [Yu et al. 2006], and SensorWare [Boulis et al. 2003]. In Mat´e and its successor ASVM, applications are divided into capsules that are flooded throughout the network. Each node stores the most recent version of a capsule and runs the application by interpreting the capsules using a Virtual Machine (VM).

The reprogramming systems share a common feature: The decision on when and where to reprogram the network is determined centrally at a base station, often by a human operator. In contrast, Agilla provides a fundamentally different programming model based on mobile agents and tuple spaces that are especially well suited for self-adaptive applications in WSNs. Mobile agents can make adaptation decisions locally and autonomously within the network via migration (i.e., moving and cloning). Since network nodes are directly exposed to the environment, they can more quickly detect changes and
better determine when software adaptation is necessary.

Mobile agents have been used in the Internet .These systems are designed to run on Internet servers, efficient resource utilization is not their main focus. Mobile agents have also been used in wireless ad hoc networks. Systems that provide this include LIME [Murphy et al. 2006], Limone [Fok et al. 2004], and Smart Messages [Kang et al. 2004]. All three were designed for relatively resource rich devices and are thus not appropriate for WSNs. For example, LIME supports tuple spaces that span multiple hosts incurring transactional costs [Carbunar et al. 2004], while Limone allocates a tuple space for each agent, rendering the cost of migration untenable. Smart Messages only supports a single thread of execution per node, meaning it cannot support multiple applications.

Mobile agents have previously been considered for use in wireless sensor networks. For example, mobile agents can perform certain operations like data integration [Qi et al. 2003, 2001a, 2001b] and tracking [Tseng et al. 2004a] better than traditional client/server mechanisms, and can be made energy efficient [Marsh et al.

2005; Tong et al. 2003]. While these efforts promote the use of mobile agents in wireless sensor networks, they were not actually deployed or evaluated in a real network. Instead, they were evaluated theoretically, in simulation, or using networks consisting of relatively resource-rich devices.

Agilla is the first system to bring the mobile agent programming model into a real wireless sensor network. By integrating the mobile agent and tuple space programming models, Agilla enables applications to be locally and autonomously self-adaptive. The Agilla programming model and middleware architecture meet the challenges unique to wireless sensor networks, for example, severe resource constraints and unreliable wireless connectivity. The novel specialization of the mobile agent and tuple space programming models combined with a careful engineering effort resulted in the working of Agilla.

3. MOBILE AGENTS

Mobile agents autonomously sense the environment and respond accordingly. Thus agents are very suitable to be deployed in the sensor network environment where many methodologies were proposed to solve the recognized problems in sensor networks by using agents. Mobile agents used in WSNs are classical software agents, which are specifically written to run separately in adaptive sensor environments. In a WSN, different mobile agents may be constructed for individual tasks, which work cooperatively to achieve the main objective of the network application

When the agent chooses to migrate to another node in the WSN, it logically transports its state and code to the destination. The mobile agent middleware framework embedded within a sensor network is the fundamental agent-based programming platform of the network. This framework mainly provides the software migration between the sensors. Agilla [8] is the most popular agent-based middleware framework for WSNs.

4. AGILLA

Agilla is a middleware which supports mobile agents on WSN. Mobile agents are dynamic, localized, intelligent programs that can move or clone themselves across the nodes to perform a specific task. They are used when it is advantageous to move software all over the place so that the sensor nodes may perform different tasks without the need to reload new programs on them. After being injected into the network, the mobile agent performs autonomously and executes its instructions upon reaching a sensor node. Mobile agents can also interact with other agents and through their use, network flexibility is considerably increased.

Agilla is initially developed for Mica2 motes where each mote in the network is considered to be a node. Agilla middleware is then loaded on the nodes and mobile agents are injected to work on this middleware. There is a neighbor list and a tuple space on
each node which are maintained by Agilla. Neighbor list consists of the addresses of one hop neighbors and the tuple space stores data to be shared by the local and remote agents (agents which reside on other nodes). Multiple agents may also reside on different nodes simultaneously and they can migrate among the nodes. Migrating agents can carry their codes and execution states, but they cannot carry tuple space of the node. Mica2 motes use TinyOS operating system which is specifically
designed for sensor nodes [17].

Portable agents running on Agilla have numerous points of interest, for example, adjustment to ecological changes and remote reconstructing which are the two essential difficulties for WSN. To delineate this circumstance, expect that a WSN is basically sent for interruption recognition in a building. Common guard powers may need to reinvent the system to identify fire or gas spill in a crisis circumstance. Introducing every one of these applications immediately is not adaptable, reasonable or versatile. Portable agents middleware address this issue. It gives element reconstructing of WSN by permitting new agents to be infused and permits old agents to

pass on. Subsequently, portable agents middleware bolster flexibility and versatility.

Since multiple agents can exist on a node simultaneously, mobile agent middleware support coexistence of multiple applications on a node.

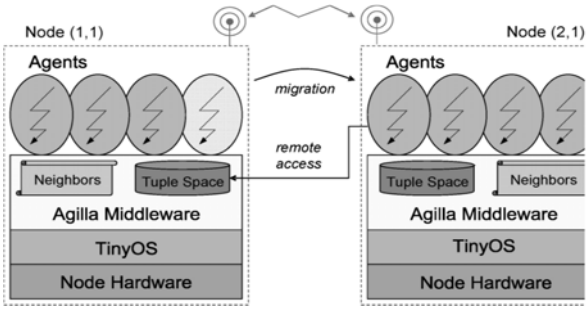

Fig 1: Agilla Model

Fig 1 shows the Agilla architecture. There are three layers: TinyOS, Agilla middleware, and mobile agents. TinyOS is the lower layer of the architecture which Geographic routing, network stack and sensor components are managed by that. Middle layer of the architecture is the core component of the Agilla middleware. Five managers together with the Agilla engine compose the middleware. Each of these is implemented as a TinyOS component and a separate process. These managers are tuple space manager, reaction manager, agent manager, context manager and code manager. Mobile agents constitute the top layer of the architecture. A mobile agent is composed of a stack, heap, various registers and code.

5. SIMULATION METHOD

TOSSIM is used to simulate a WSN with agents loaded on each mote. The motes have TinyOS running on them. We have also used TinyViz, a visualization tool for TOSSIM simulator.

To setup the simulation environment, some software must be installed. These are: Sun's Java Development Kit (JDK) version 1.4.x, Sun's javax.comm package. Later, TinyOS, Agilla, TinyViz and AgentInjector can be installed. mobile agent middleware such as Agilla has not been transferred to the simulation environment, Thus Agent Injector can be used to inject agents into a network.

First goal in conducting the simulations was to measure performance of important Agilla instructions. The simulations are performed on a 25-node network arranged in a 5*5 grid. The term "migration" consists of moving and cloning of an agent. Moving of an agent is the transferring of the agent from one node to another node, whereas cloning is copying an agent from one node to another node. Special instructions are used to move or clone one agent from one node to another.

Cloning instructions are sclone and wclone. Moving instructions are smove and wmove. In order to benchmark strong and weak migrations, we used test agents. We measured consumed time during the smove (strong move), wmove (weak move), sclone (strong clone) and wclone (weak clone) operations.

The first letter indicates whether the operation is weak or strong. A weak migration transfers only the code, all execution state is reset and the agent resumes running from the beginning when it arrives at the destination. A strong migration transfers everything, meaning an agent resumes execution where it left off. Our experience with the fire tracking application shows that the choice between strong vs. weak migration significantly affects the application overhead, performance, and reliability.

6. SIMULATION

We used two different scenarios to benchmark move (smove and wmove) and clone (sclone and wclone) instructions. The difference between the scenarios is the heap operations; 10 variables are recorded to the heap in the first scenario while heap operations were not used in the second scenario. The heap is a random-access storage area that can store up to 12 variables. First, we test smove and wmove instructions for each scenario (with and without heap operations). Then, sclone and wclone instructions are simulated for same scenarios.

6.1. SMOVE VS. WMOVE (WITH HEAP OPERATIONS)

The agent used to simulate smove instruction with heap operations is shown in Fig 2. This agent saves 10 values to its heap and moves to a random neighbor carrying its code, program counter, stack and heap. Simulation of these agents are done on a virtual wireless sensor network with 25 nodes. The simulations is repeated 50 times for each agent (smove and wmove) and the average elapsed times for smove and wmove operations were computed.

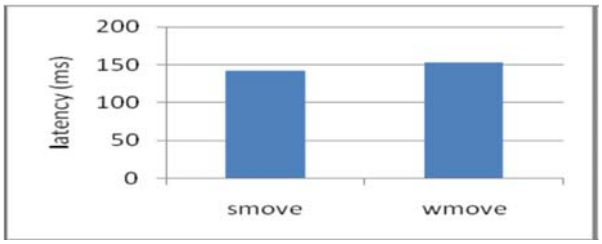

Fig 2 : Latency of smove and wmove instructions with heap operations.

According to Fig 2, average migration time of an agent from one node to another is 298 milliseconds with the smove instruction; while wmove takes 153 milliseconds. This result is expected, because smove instruction transfers agent's heap together with the agent's code, while wmove instruction transfers only the agent's code. Each variable in the heap is 40 bits; therefore, 10 variables use 400 bits. This extra-load increases the migration time of smove agent.

6.2. SMOVE VS. WMOVE (WITHOUT HEAP OPERATIONS)

In these agents, heap operations are removed to test the amount of the time taken for smove and wmove. The simulations is repeated for 50 times for each agent (smove or wmove) and the average latency is computed for smove and wmove operations.

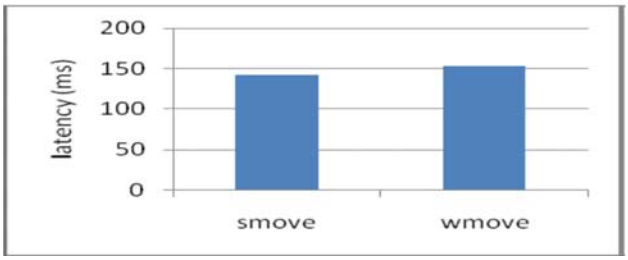

Fig 3: Latency of smove and wmove instructions without heap operations

Average migration time of an agent from one node to another is 139 milliseconds with the smove instruction; while wmove takes 153 milliseconds. This result is expected, because in a weak migration, agent resumes execution from the beginning.

6.3. SCLONE VS. WCLONE (WITH HEAP OPERATIONS)

The simulations was reapeated for 50 times on a simulated wireless sensor network with 25 nodes as in the previous simulations. Fig 4 displays the results of the simulations for sclone and wclone agents with heap operations.

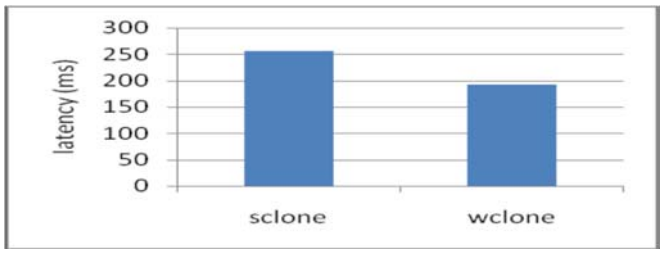

Fig 4. Latency of sclone and wclone instructions with heap operations

According to Fig 3, average migration time of an agent from one node to another is 256 milliseconds with sclone instruction; while wclone takes 193 milliseconds. This result is expected, because transferring agent's code together with the agent's heap increases latency.

6.4. SCLONE VS. WCLONE (WITHOUT HEAP OPERATIONS)

According to Fig 5, average migration time of an agent from one node to another is 170 milliseconds with the smove instruction; while wmove takes 193 milliseconds. This result is

expected, because in a weak migration, agent resumes execution from the beginning.

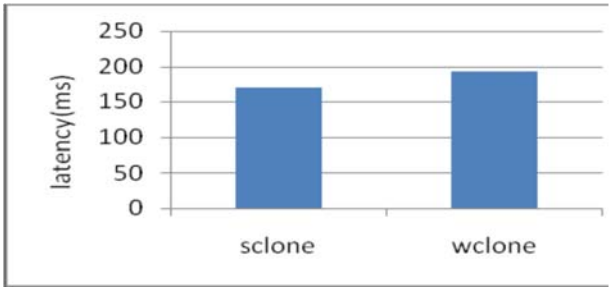

Fig 5: Latency of sclone and wclone instructions without heap operations

7. SPREADING OF AGENT BENCHMARK

To measure the spreading performance of the agents, a scenario is created according to which an agent is injected to a node of the WSN to sense the temperature of the node and send the recorded value to the base station, and the agent clones itself to its neighbors for spreading into the network. Neighbor nodes are chosen randomly by randnbr instruction. This instruction looks at the neighbor list of the agent and chooses a random location from the list. The goal of the agent is to visit all the nodes of the WSN and send temperature readings to the base station. Clone operations were preferred to provide faster spreading. Since wclone instruction is more efficient than sclone as mentioned before, it was chosen for migration. This agent has been tested on simulated wireless sensor networks which included various number of nodes. The agent was tested 10 times on each WSN. Average values of the tests are taken as the final result. Simulation results are shown in Fig 6. The graph shows that the agent can spread into a 5-node Wsn in 0.4 seconds while it can spread into a 15-node WSN in 1.3 seconds. As expected, larger WSN increases spreading time. On the other hand, latency per node is decreased as the network size increases. For example, 115 ms is consumed per node for a 5-node WSN, while 80 ms is consumed per node for a 150-node WSN.

As mentioned before, the agent clones itself to its neighbors randomly. Accordingly, the agent is cloned to some nodes more than once while other nodes aren't visited by the agent. Reliability of this agent is computed according to the equation:

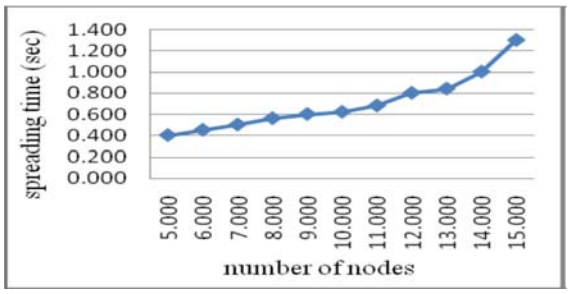

Fig 6: The Agent's Spreading Time into WSN

Reliability = total number of nodes – number of unvisited node

Total number of nodes

Result of the reliability analysis is shown in Fig 7. Reliability of the agent changes between 76% and 85% inconsistently because of the random nature of the migration operation.

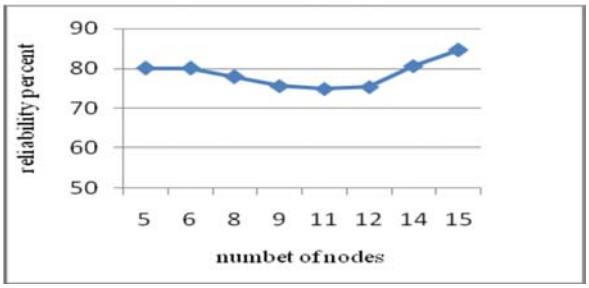

Fig 7: Reliability of the agent in Fig 6.

8. CONCLUSIONS

Simulation was performed on mobile agent middlewares in TinyOS and presented a simulation environment for mobile agents running on Agilla using TOSSIM. Different Agilla commands were tested and the execution time of each agent were measured using simulation. By utilizing TinyViz to visually observe the simulation process and AgentInjector to transfer Agilla to the simulation environment, different Agilla commands were tested and execution time of each agent was measured using simulation and the results were analyzed for simulation validation process. These experiments will be carried out using

different network topologies and different agents in the future. Actual network will be setup and experiments will be compared with simulations as future work.

REFERENCES

[1] Wang, M., Cao, J., Li,J., and Sajal K. Dasi. (2006). "Middleware for Wireless Sensor Networks: A Survey". Department of Computer Science, Internets and Mobile Computing Lab, Department of Computer Science and Engineering,, Supported by Hong Kong Polytechnic University under the ICRG grant NO.G-YE57, Hong Kong RGC under the Grant of A Research Center Ubiquitous Computing and the National Hi-Tech Research and Development 863 Program of China under Grant No.2006AA01Z231.

[2] Md.Atiqur.R.(2009). "Middleware for wireless sensor networks Challenges and Approaches". TKK T-110.5190 Seminar on Internetworking.

[3] Tong, S. (2009). "An Evaluation Framework for middleware approaches on Wireless Sensor Networks". Seminar on Internetworking. Helsinki University of Technology, TKK T-110.5190.

[4] Molla M.M., Ahamed S.I. 2006. A survey of middleware for sensor network and challenges. In Proceedings of IEEE International Conference on Parallel Processing Workshops (Milwaukee, Wisconsin, August 14 – 15, 2006)

[5] Yao, Y. and Gehrke, J. 2002. The cougar approach to innetwork query processing in sensor networks. SIGMOD Rec. 31, 3 (Sep. 2002), 918.

[6] Yu, X., Niyogi K., Mehrotra, S. and Venkatasubramanian N. 2003. Adaptive Middleware for Distributed Sensor Environments. IEEE DS Online 4, 5 (May 2003).

[7] Madden, S. R., Franklin, M. J., Hellerstein, J. M., and Hong, W. 2005. TinyDB: an acquisitional query processing system for sensor networks. ACM Trans. Database Syst. 30, 1 (Mar. 2005), 122173.

[8] MC Lin, YC Chen, SL Tsao. (2012). "Design and implementation of a home and building gateway with integration of nonintrusive load monitoring meters" Industrial Technology (ICIT), 2012 IEEE International Conference. 148-153.

[9] Prasannasrinivasa.S and Suman.N. (2012). "A Study of Middleware for Wireless Sensor Networks". International Journal of Research and Reviews in Ad Hoc Networks (IJRRAN) Vol. 2, No. 2, June 2012, ISSN: 2046-5106.

[10] Kumar.A, Xie.B. (2012)." Handbook of Mobile Systems Applications and Services. "University of Louisville, Louisville, KY, USA, University of Cincinnati, Cincinnati, OH, USA .

[11] Md.Atiqur.R. (2009). "Middleware for wireless sensor networks Challenges and Approaches". Helsinki University of Technology, Seminar on Internetworking. TKK T-110.5190.

[12] Fok, C.-L. , Roman, G.-C. , and Lu, C. (2009). "Agilla: A mobile agent middleware for self-adaptive wireless sensor networks". ACM Trans. Autonom. Adapt. Syst. 4, 3, Article 16 (July 2009), 26 pages DOI = 10.1145/1552297.1552299