AN ENERGY EFFICIENT APPROACH FOR DATA COLLECTION IN WIRELESS SENSOR NETWORK

A DISSERTATION

Submitted in partial fulfillment of the requirements for the award of the degree

of

MASTER OF TECHNOLOGY

in

INFORMATION TECHNOLOGY

By

MRUDANG MEHTA

DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY ROORKEE
ROORKEE - 247 667 (INDIA)

JUNE, 2008

CANDIDATE'S DECLARATION

I hereby declare that the work, which is being presented in the dissertation entitled "**AN ENERGY EFFICIENT APPROACH FOR DATA COLLECTION IN WIRELESS SENSOR NETWORK**" towards the partial fulfillment of the requirement for the award of the degree of **Master of Technology** in **Information Technology** submitted in the Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee, Roorkee (India) is an authentic record of my own work carried out during the period from July 2007 to June 2008, under the guidance of **Dr. Manoj Misra, Professor, Department of Electronics and Computer Engineering, IIT Roorkee.**

I have not submitted the matter embodied in this dissertation for the award of any other degree or diploma.

Date: 30/06/08

Place: Roorkee

[illegible]

(MRUDANG MEHTA)

CERTIFICATE

This is to certify that the above statement made by the candidate is correct to the best of my knowledge and belief.

Date: 30/06/08

Place: Roorkee

[illegible]

(Dr. MANOJ MISRA)

Professor

Department of Electronics and Computer Engineering

IIT Roorkee – 247 667

ACKNOWLEDGEMENTS

I would like to take this opportunity to extend my heartfelt gratitude to my guide, **Dr. Manoj Misra**, Professor, Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee, for providing me guidance throughout my graduate program at IIT Roorkee. His knowledge and enthusiasm in my work has been of great benefit to me. He has been extremely patient and critical all through my work.

I extend my sincere thanks to **Dr. D. K. Mehra**, Professor and Head of the Department of Electronics and Computer Engineering, Indian Institute of Technology, Roorkee for providing facilities for the work.

I am indebted to my parents, my sister and my friends for being a source of constant support and motivation. I am thankful to my friends and relatives who took care of my family in my absence.

I would like to thank AICTE for running Quality Improvement Program. I thank QIP center at IIT, Roorkee for their cooperation. I also thank Dharmsinh Desai University, Nadiad, Gujarat, for sponsoring me for pursuing master degree program.

[illegible]

MRUDANG MEHTA.

ABSTRACT

Wireless Sensor Network (WSN) has a large number of sensor nodes with the ability to communicate among themselves and also to an external sink or a base-station. The sensor nodes could be scattered randomly in harsh environments such as a battlefield or deterministically placed at specified locations. The sensors coordinate among themselves to form a communication network.

Flexible data collection in wireless sensor networks is a significant research challenge since it must support dynamic reconfiguration of collection tasks while accommodating constraints of the sensor network infrastructure. These constraints come from the sensor node (due to energy limitations), from the network (typically large, faulty and bandwidth-limited) and from the application (which poses timeliness needs). In this work, we address energy efficient approach for data collection in wireless sensor network.

In this report, we present an energy efficient data collection approach for wireless sensor network. There are many approaches available using client/server computing model. In client/server computing model, different source nodes send data periodically to the sink node. This increases congestion and energy consumption of node increases. Our approach uses mobile agent based computing model. By using mobile agent, we developed a technique through which we reduce the energy consumption of the sensor nodes. Instead of many source nodes transmit data individually to sink node, here, mobile agent visits each sensor node for data collection. This reduces energy consumption of sensor nodes and increases network lifetime.

Performance of our approach has been evaluated and compared by changing different parameters such as task duration, mobile agent access delay and sensed data size. The results have shown that our approach shows better performance than directed diffusion.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF PUBLICATION

1. M. Mehta, M. Misra, "Mobile Agent Based Data Collection Techniques For Wireless Sensor Network," in IEEE Madras sponsored National Conference in Mobile and Pervasive Computing (Accepted)

CHAPTER 1

INTRODUCTION

1.1 Overview

A wireless sensor network (WSN) is a wireless network consisting of spatially distributed autonomous devices using sensors to cooperatively monitor physical or environmental conditions, such as temperature, sound, vibration, pressure, or motion at different locations. [1] Development of Micro-Electro-Mechanical System (MEMS) and semiconductor technology have given sound base for the development of low cost, battery operated and small sized devices which does sensing, processing and communicating wirelessly. Such devices are called “motes”. A mote generally consists of radio front end, microcontroller, power supply and the actual sensor. The desirable size of a mote is few cubic millimeters and the desirable target price range less than one US dollar. A “mote” is single wireless sensor node in the wireless sensor network. Although, motes are autonomous they can form a network and co-operate with each other under various architectures. [2]

WSNs promise several advantages over traditional sensing methods in many ways: better coverage, fault tolerance and robustness. The ad hoc nature and deploy-and-leave vision make them even more attractive in real-world applications. Several real-world applications are being designed and developed that are taking advantage of this new technology. For example, approximately 300 sensors are distributed throughout the structure of the new Rion-Antirion Bridge in the Athens; strain gauges keep track of framework fatigue, displacement transducer monitor how the bridge blows in the wind, and three-dimensional accelerometers measure the impact of earthquake [3]. Sensors are also found in the nests at the Great Duke Island for monitoring one of the largest breeding colonies of Leach’s storm petrels [2], in building for fire crew assistance [4], in trees for fire detection [5], in wine grape vineyards for improving the quality of the crop and the performance of the land, and in many other applications such as military applications, monitoring toxic zones, agriculture, industrial automation, healthcare and volcano monitoring [6].

Although WSNs can be made pervasive due to their low cost and small nodes, they are also faulty and energy-constrained. Wireless links are prone to interference, collision, fading, and congestion. Nodes are powered by limited battery capability that cannot feasibly be recharged because WSN are large and sometimes scattered in unreachable places. Therefore, data is prone to loss, and energy is a valuable resource that must be used wisely.

1.2 Data Collection in Wireless Sensor Networks

Different approaches have been proposed for collecting data being sensed in the WSN in a flexible, reliable, and efficient manner.

Multipath approaches achieve a high degree of reliability by making use of multiple paths to send information from every sensor node to the collection point. Query propagation approaches tend to propagate an SQL-like query along a spanning tree that is rooted at the collection point and covers all sensor nodes. Query propagation approaches treat the WSN as a large distributed database -a feature that provides a high degree of flexibility.

This thesis focuses on an approach based on mobile agents. The mobile agents approach is a programming paradigm in which tasks are carried out by autonomous and self-aware programs known as mobile agents. A mobile agent is formed by its code and state (data, stack, and registers), and it can clone and migrate to other locations (copying and moving its own code and state). While pursuing the goal of collecting the sensed data in a WSN, instead of transmitting the raw data from the sources to the application in the collection point, the application (or a subset of it) is sent to where the data is. Thus, mobile agents carry and aggregate the data being sensed. Moreover, mobile agents can be aware of network failures.

This capability enables them to dynamically decide where to move or clone in the event of an unexpected failure or topology change. Therefore, mobile agents allow a great degree of flexibility regarding which data is collected and in what manner.

In terms of reliability, mobile agents provide a greater degree of fault-tolerance than query propagation and single-path approaches, comparable to multipath approaches. However, the time it takes the mobile agents to collect all the information (i.e. latency) tends to be larger than in other approaches such as multipath and single path approaches.

1.3 Statement of the Problem

The aim of this research is to develop energy efficient data collection mechanism for wireless sensor network using mobile agent based approach. The following are the objectives of our work:

1. To design and implement an efficient mechanism for mobile agent routing for data collection from sink to source (sink request for data and receives data from source).
2. To design and implement an efficient mechanism for mobile agent routing for data collection from source to sink (source receives request and sends data to sink).
3. To evaluate the performance of the approach.

1.4 Organization of the Report

This report comprises of six chapters including this chapter that introduces the topic and states the problem. The rest of the dissertation report is organized as follows.

Chapter 2 gives an overview of wireless sensor network. It introduces the topic, discusses issues in WSN. This chapter also provides overview of general mobile agent approach-its issues and applications. It discusses the related work and research gaps for the data collection in wireless sensor network.

Chapter 3 discusses the proposed energy efficient mobile agent based approach for data collection in wireless sensor network. It discusses basic idea and design of the approach.

Chapter 4 discusses implementation. It gives an overview of ns-2 Simulator and it discusses implementation detail of proposed approach.

Chapter 5 discusses the simulation results and displays the effectiveness of the proposed mechanism for data collection in wireless sensor network.

Chapter 6 concludes the work and gives the directions for future work.

CHAPTER 2

BACKGROUND AND RELATED WORK

2.1 Wireless Sensor Networks

Wireless sensor networks are recognized as a new frontier in communications. Today, due to availability of cutting edge MEMS (Micro Electro Mechanical Systems) Technology; low-powered, low-cost, tiny sensors can be made which can be used for monitoring physical world. These tiny sensors (also called as "motes") are self-organizing and can form multi-hop distributed wireless network called wireless sensor network.

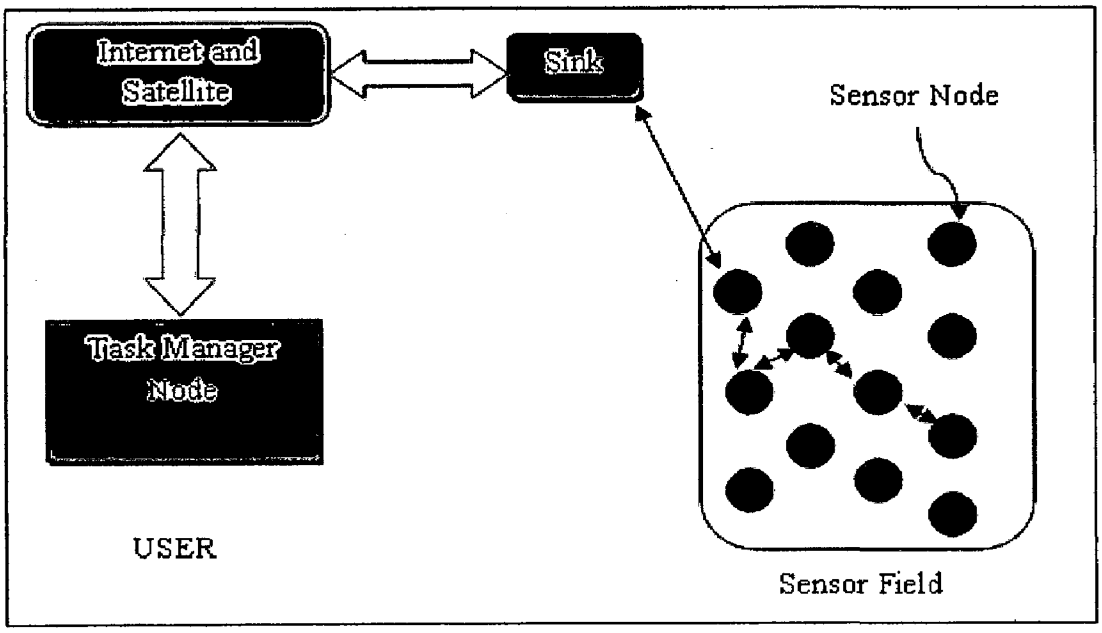


Figure.2.1 A Typical WSN

Figure 2.1 shows Typical WSN. It shows data pertaining to event monitoring can be gathered and transmitted to end user through Internet. Here, **Sensor Nodes** are wireless communication device which are small, inexpensive and low power devices capable of sensing and local processing. **Sensor Network** is a collection of sensor nodes which coordinate to perform some specific action for example detecting fire in the trees. Unlike traditional networks, these rely on dense deployment and coordination. **Sensor Field** is a target area where the sensor nodes have been deployed to acquire data. Depending on the nature of tasking and manner of being deployed, the sensor field may get partitioned at times. The information flow is through a sink node - which may dynamically change due to design or operational requirements. The sink connects to the user's network through internet or satellite.

2.1.1 Applications of Wireless Sensor Network

We can classify applications of WSN as per deployment of WSN. WSN can be deployed over ground, underground, underwater and outside earth.

Wireless Underground Sensor Network Applications (WUgSNA): In wireless underground sensor network (WUgSN), majority of the devices are placed completely below the ground. Each device contains sensors, processor, memory, antenna and power source. This makes it easy deployment than the existing underground solutions in which sensors are connected to surface through wired infrastructure. This area (WUgSN) possesses many interesting research challenges. Typical applications are to monitor a variety of conditions, such as soil properties for agricultural applications, toxic substances for environmental monitoring, infrastructure (mainly underground, e.g. pipes, electrical wiring, liquid storage tanks, etc.) monitoring, location determination of objects and military applications such as mine field monitoring and border patrol. [7]

Wireless Underwater Sensor Network Applications (WUwSNA): The enabling technology for applications of Wireless Underwater Sensor Networks (WUwSN) is wireless underwater acoustic networking. Under-Water Acoustic Sensor Networks (UW-ASNs) consist of a variable number of sensors and vehicles that are deployed to perform collaborative monitoring tasks over a given area. Sensors and vehicles are self-organizing and self-adaptive to the oceanic environment. Typical applications are oceanographic data collection, pollution monitoring, oil and gas exploration, disaster prevention, tactical surveillance and exploration of natural undersea resources. [8]

Outer Earth Wireless Sensor Network Application (OEWSNA): WSN has potential application in space exploration. WSN may be helpful in monitoring of planet other than earth, too. This is also very important as continuous manned monitoring may not be feasible for other planets. A research in this field is ongoing mainly by NASA. A proposed application of WSN is that a robot embedded with many sensors may be placed on MARS to collect data, which it could transmit to base station on MARS for real time data analysis. Base station can do analysis and direct future action to robot. [9][10]

Ground Based Wireless Sensor Network Applications (GBWSNA): This is the most general area where many applications of WSN are existing or being developed. The typical application areas are agriculture, industrial automation, earthquake monitoring, building structure monitoring, volcano monitoring, healthcare, security-civilian and military, robotics, and internet enabled applications and gaming applications. [1] [2]

2.1.2 Wireless Sensor Node Hardware

A typical sensor node has the sensing, computation, communication and power as its main components as shown in Figure 2.2. Mobiliser and GPS based location components may also be present. Sensing Unit comprises of a sensor and their associated ADC (Analog to Digital Converter). Processing Unit comprises of a Processor with memory storage. Transceiver may have an inbuilt antenna or a mount for an external whip. Batteries are generally used to satisfy energy requirements. [1]

There are some commercial vendors like Crossbow and Intel provides different types of sensor nodes also called motes. Figure 2.3 shows some sensor nodes from Crossbow and Intel. [11] [12]

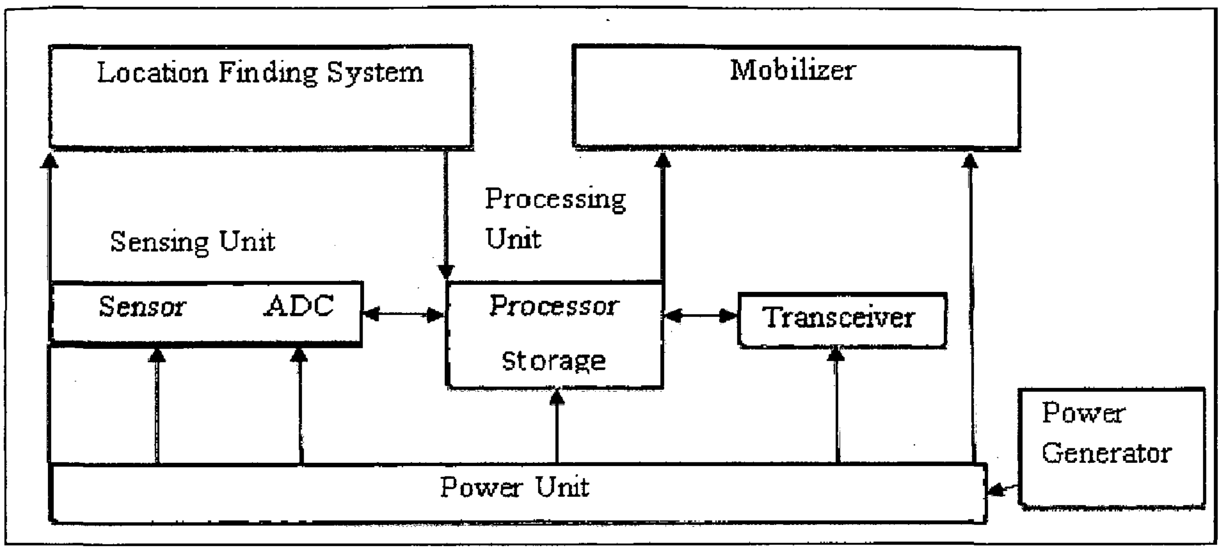
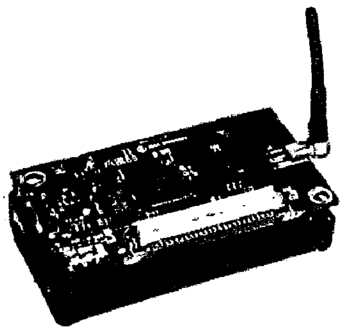


Figure 2.2 Sensor Node Components



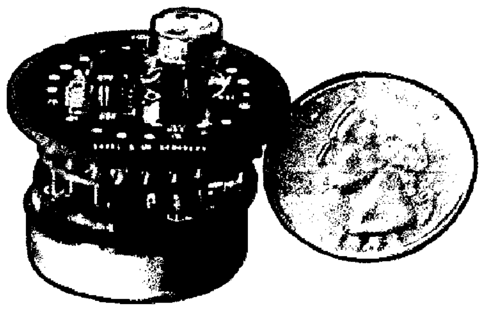


Figure 2.3 (a) MicaZ Mote (Crossbow) (b) Coin sized Intel Mote

2.1.3 Sensor Node Requirements for Applications

It has to be noted that different applications call for different characteristics having varied inter-se importance. Environmental requirements warrant nodes to be characteristically strong in the areas viz. energy efficiency (long battery life) to include intermittent connectivity and schedule sleep mode for redundant sensors, inexpensive nodes (large quantity needed), reduced size of nodes (small, microscopic), auto-configuration of sensors, scalable network, robust nodes to handle harsh environments of the operating conditions (heat, water, snow, humidity, wind).

In the *medical* scenario key characteristics required are energy efficiency (long battery life, heat/kinetic/bio battery), hidden device (not visually detectable), biologically safe, fault-tolerant, reliable, encrypted bio information (for privacy) and be interference-safe (RF noise, 900 MHz range, should not malfunction due to ambient RF conditions). In *military* applications, in addition to the above, key characteristics required are ubiquitous and undetectable, auto-deployment (motorised, robotised, missile warhead launched), fault-tolerant, reliable, strong encryption (low overhead), and they should also be made rugged to withstand shocks, vibrations several cyclic changes of extreme weather conditions. They should allow in-filtration of query at a point and ex-filtration of the output at a distant given point in given almost real time frame.

Similarly, urban requirements are diverse range of sensor types, interoperability, highly customisable, (for the diverse user base), scalable network (wide area of coverage and increasing the collaboration within the environment).

2.1.4 Research Issues in Wireless Sensor Network

Design issues: One of the major research challenges is design of wireless sensor network. Design of sensor network is majorly influenced by the following factors:

- Fault tolerance
- Scalability, Mobility and dynamic network topology
- Production cost

- Operating environment
- Hardware constraints
- Transmission media
- Power Consumption

The above factors deal with the developing efficient hardware, reliable transmission techniques and resolving communication issues of MAC and Network layer.

Routing issues: One major research issue is developing a routing protocol which consumes least energy as devices (motes) are generally battery operated. Protocols should be scalable and robust with topological changes of the network. Mobility is another issue which shall be taken care while developing routing algorithm as in many applications either sensors or source are mobile. Design and development of efficient protocol or techniques for data abstraction, data filtering and data aggregation is an important research issue to save energy consumption of overall network.

Middleware issues: In recent year, a novel approach of middleware is developed to bridge the gap between applications and low-level constructs. This approach resolves many WSN issues and enhances application development. We can view WSN middleware as a software infrastructure that glues together the network hardware, operating systems, network stacks and applications. A complete middleware solution should contain a runtime environment that supports multiple (homogeneous or heterogeneous) applications. It should standardize system services like data aggregation, control and management policies adapting to target applications, and mechanism to achieve adaptive and efficient system resources use to extend the network lifetime. Examples of middleware are Mate by University of California, Berkeley and Imapala by Princeton University.

2.2 Mobile Agent Technology

Mobile agents are a novel way of building distributed software systems. Traditional distributed systems are built out of stationary programs that pass data back and forth across a network. Mobile agents, by contrast, are programs that they themselves move from node to node: the computation moves, not just the attendant data. Mobile agents in [13] are defined by *following* important properties:

Agents encapsulate a thread of execution along with a bundle of code and data. Each agent runs independently of all others, is self-contained from a programmatic perspective, and preserves all of its state when it moves from one network node to another.

Any agent can move easily across the network. The underlying infrastructure provides a language-level primitive that an agent can call to move itself to a neighboring node.

Agents must be small in size. Because there is some cost associated with hosting and transporting an agent, agents are designed to be as minimal as possible. Simple agents serve as building blocks for complex aggregate behaviour.

An agent is able to cooperate with other agents in order to perform complex or dynamic tasks. Agents may read from and write to a shared block of memory on each node, and can use this facility both to coordinate with other agents executing on that node and to leave information behind for subsequent visitors.

An agent is able to identify and use resources specific to any node on which it finds itself. In the simulation presented in this chapter, the nodes are differentiated only by who their neighbours are (and agents do make use of this information). In a more heterogeneous network, certain nodes might have access to particular kinds of information – such as absolute location derived from a global positioning system receiver – which agents could leverage.

2.2.1 Mobile Agent vs. Stationary Agent

Mobility is an orthogonal property of agents. That is, all agents are not necessarily required to be mobile. An agent can remain stationary and communicate with the surroundings by conventional means like remote procedure calls (RPC) and remote object invocation (RMI) etc. The agents that do not or cannot move are called *stationary agents*. On the other side, a *mobile agent* is not bound to the system where it begins execution. The mobile agent is free to travel among the hosts in the network. Once created in one execution environment, it can transport its state and code with it to another execution environment in the network, where it resumes execution.

2.2.2 Mobile Agents and Mobile Agent Environment

A mobile agent must contain all of the following models: an agent model, a life-cycle model, a computational model, a security model, a configuration model and finally a navigation model. In [14], a working definition of a mobile agent is given as "A mobile agent consists of a self-contained piece of software that can migrate and execute on different machines in a dynamic networked environment, and that senses and (re) acts autonomously and proactively in this environment to realize a set of goals or tasks."

The software environment in which the mobile agents exist is called mobile agent environment. The definition of mobile agent environment as per [15] is given as "A mobile agent environment is a software system distributed over a network of heterogeneous computers. Its primary task is to provide an environment in which mobile agents can execute. It implements the majority of the models possessed by a mobile agent."

The above definitions state the essence of a mobile agent and the environment in which it exists. The mobile agent environment is built on top of a host system. Mobile agents travel between mobile agent environments. They can communicate with each other either locally or remotely. Finally, a communication can also take place between a mobile agent and a host service.

2.2.3 Comparison of Mobile Agent Paradigm with Traditional Client Server Paradigm

Today, client-server paradigm enjoys various techniques like remote procedure calling (RPC), remote object-method invocation (like Java RMI or CORBA) etc. The RPC paradigm, for example, is the prominent technique of the client-server paradigm. It views computer-to-computer communication as enabling one computer to call procedures in another. Each message that the network transports either requests or acknowledges a procedure's performance. Two computers whose communication follows the RPC paradigm have to agree upon the effects of each remotely accessible procedure and the types of its arguments and results. This agreement constitutes a protocol. For an example, as shown in Figure 2.4 a client computer initiates a series of remote procedure calls with a server in order to accomplish a task. Each call involves a request sent from client to server and a response sent from server to client. Thus the salient feature of client-server paradigm is that each interaction between the client and the server requires two acts of communication. That is, ongoing interaction requires ongoing communication.

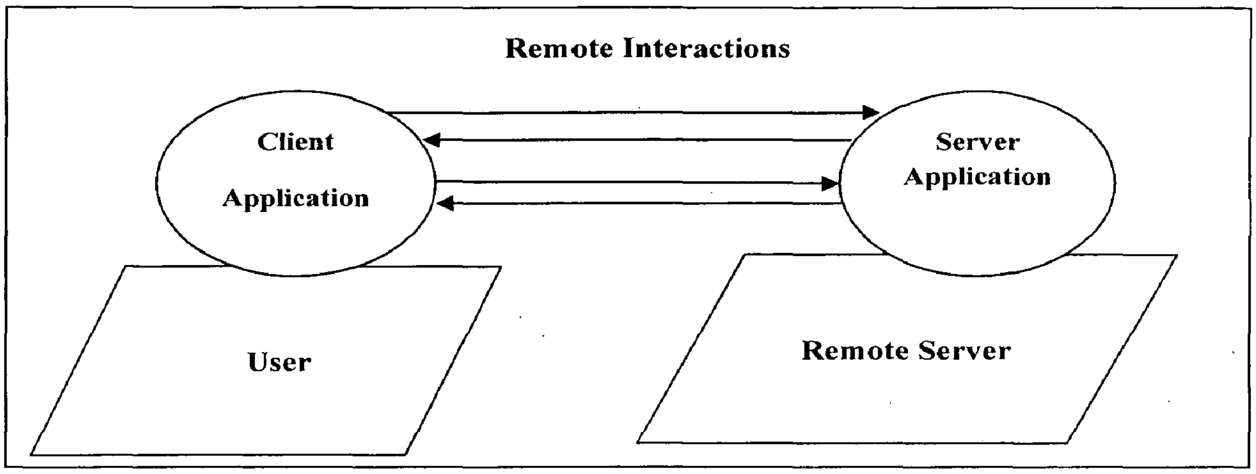


Figure 2.4 Communication Using Client-Server Paradigm

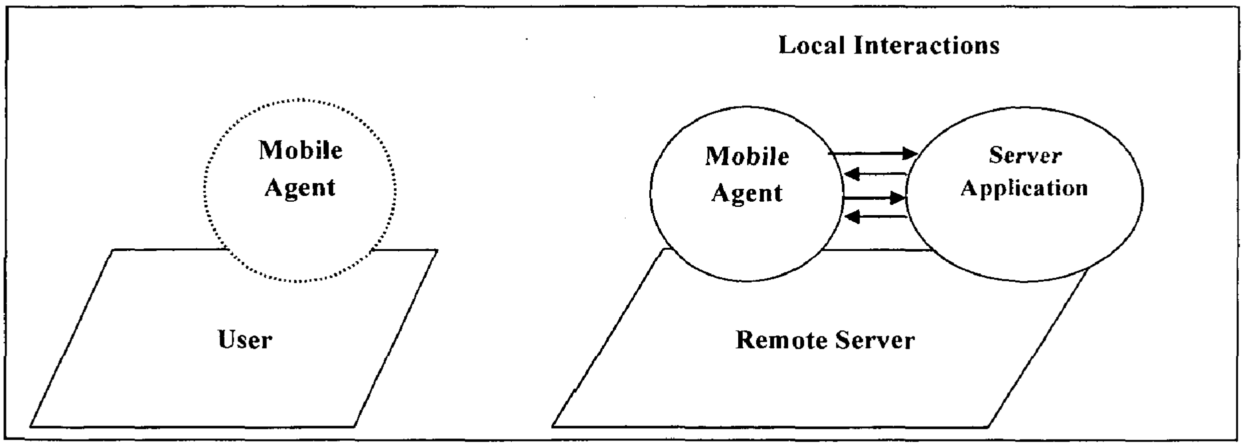


Figure 2.5 Communication Using Mobile Agent Paradigm

In contrast to client-server paradigm, the mobile agent paradigm views computer-to-computer communication as enabling one computer not only to call procedures in another, but also to supply the procedures to be performed. Each message that the network transports consists of a procedure whose performance the sending computer either began or continued and the receiving computer is to continue and the data which are the procedure's current state. Two computers whose communication follows the mobile agent paradigm have to agree upon the instructions that are allowed in a procedure and the types of data that are allowed in its state. This agreement constitutes a language. This language provides instructions that allow the procedure to examine and to modify its state, making certain decisions and call procedures provided by the receiving computer. But here the procedure calls will be local to the receiving computer, which is an important advantage of the mobile agent paradigm. Figure 2.5 represents the same example scenario as before but using mobile agent paradigm. Here the client computer sends an agent to the server whose procedure there makes the required requests to the server. The dotted line in Figure 2.5 shows the previous movement of the agent. All the request and responses in this case are local to the server and no network is required to complete a task. Thus the salient feature of mobile agent paradigm is that each a client computer and a server can interact without using the network once the network has transported an agent between them. That is, ongoing interaction does not require ongoing communication.

2.2.4 Advantages of Mobile Agent Paradigm

The mobile agents have several strengths. The following is the brief discussion of seven good reasons for using mobile agents [16]:

1. *They reduce network load*: The main motivation behind using mobile agents is to move the communication to the data rather than the data to the computations. Distributed systems often required multiple interactions to complete a task. But using mobile agent allows us to package a conversation and send it to a destination host. Thus all the interactions can now take place locally. The result is enormous reduction of network

traffic. Similarly instead of transferring large amount of data from the remote host and then processing it at the receiving host, an agent send to the remote host can processed the data in its locality.

2. *They overcome network latency*: Certain real-time systems require immediate action in response to the changes in their environment. But a central controller cannot respond immediately due to the network latency. Here mobile agents can be a good solution as they can be dispatched from a central controller to act locally in the system and thus can respond immediately.

3. *They encapsulate protocols*: Due to the continuous evolution of existing protocols in a distributed system, it is very cumbersome to upgrade protocol code property in each host. Result may be that protocols become a legacy problem. Mobile agents are able to move to remote hosts in order to establish "channels" based on proprietary protocols.

4. *They execute asynchronously and autonomously*: This is the reason why mobile agents are so promising in wireless networks. Due to the fragile and expensive wireless network connections, a continuous open connection between a mobile device and a fixed network will not be always feasible. In this case the task of the mobile user can be embedded into mobile agents, which can then be dispatched into the fixed network and can operate asynchronously and autonomously to accomplish the task. At a later stage the mobile user can reconnect and collect the agent with the results.

5. *They adapt dynamically*: Mobile agents are capable of sensing their execution environment and take decisions based on that dynamically.

6. *They are naturally heterogeneous*: Mobile agents are generally independent of the computer and the transport layer and depend only on their execution environment. Hence they can perform efficiently in any type of heterogeneous networks.

7. *They are robust and fault-tolerant*: The dynamic reactivity of mobile agents to unfavourable situations makes it easier to build robust and fault-tolerant distributed systems.

2.2.5 Migration Types of Mobile Agent

All the mobile agents systems have the same general architecture. A system server on each machine accepts incoming agents, and for each agent, starts up an appropriate environment, loads the agent's state information into the environment, and resumes agent execution. However, some differences are quite notable. Some systems, like Java-based systems (e.g. Aglets, Concordia and MOA) have multi-threaded servers and run each agent in a thread of the server process itself while some other systems have multi-process servers and run each agent in a separate interpreter process and the rest uses some combination of these two extremes.

Mobile agent systems generally provide one of the two types of migration:

1. Strong migration that captures an agent's object state, code and control state, allowing it to continue execution from the exact point at which it left off.. The strong migration is more convenient for the end programmer, but ore work for the system developer since routines to capture control state must be added to the existing interpreters.
2. Weak migration that captures only the agent's objects state and code, and then calls a known entry point inside its code to restart the agent on the new machine. All the java-based systems do not capture an agent's thread (or control) state during migration and thus use weak migration. This is because thread capture requires modifications to the standard Java virtual machine. In other words, thread capture means that the systems could be used with one specific virtual machine, significantly reducing market acceptance.

2.3 Related Work

There have been several efforts for achieving data collection in wireless sensor network (WSN). To our knowledge, none of the approaches addresses completely all the goals of reliability, flexibility, and efficiency. These approaches can be called as multipath approaches and query propagation approaches. Multipath approaches focus on reliability. Query propagation approaches provide flexibility but lack reliability.

Multipath approaches provide higher reliability than single-path data routing approaches with similar latency (i.e. time to get all the information at the collection point) [17, 18, 29]. They achieve a higher degree of reliability by having multiple paths for the same source-destination pair. The main difference between multipath approaches is whether packets are redundantly forwarded through all multiple paths simultaneously or only forwarded along non-primary paths if the primary path fails. Another main difference is whether the multiple paths are completely disjoint.

By forwarding the information through multiple paths, it is possible to achieve a high degree of reliability as well as the same low latency as single path approaches. However, less energy is spent when using a single path, since multiple paths do not need to be maintained. More importantly, single path approaches do not transmit multiple copies of the same packet. In [19], the authors describe a protocol to deliver packets at a desired reliability by sending copies of each packet along multiple paths from source to sink. Upon reception of a packet, every node decides how many replicas it needs to make and where to forward them. This decision is based on some minimal state information carried by the packet and the node's local information.

Every node decides how many paths it needs to span based on the desired degree of reliability. Load-balancing is achieved by selecting these paths randomly. In Figure 2.6, we can see how the data is being forwarded through multiple paths. The authors also study and compare end-to-end unreliable (with no ACKs) multipaths with end-to-end reliable multipaths (with end-to-end ACKs). End-to-end unreliable multipaths are preferred to end-to-end reliable multipaths. The latter adds more complexity; yet it has a similar number of retransmissions. In particular, when the number of hops or the error rate is high, the likelihood of the acknowledgements being lost is also high, which triggers retransmissions. When the number of hops or the error rate is low, the extra overhead due to the use of end-to-end acknowledgements is not justified. In [18], the authors present and compare two approaches for multipath routing in WSNs: node-disjoint multipath and braided paths. The first of the two approaches is based on a classical node-disjoint multipath.

Several studies propose to propagate SQL-like queries on WSNs [19, 20]. Here, the sensor network becomes a large distributed database able to answer queries. Query can be generated as described in Figure 2.6. This brings more flexibility in terms of re-using the already deployed WSN for different purposes. SQL queries are generated at a collection point which acts as a root in a routing tree.

In these query-based approaches, queries are always propagated following a tree rooted at the collection point. This becomes a fundamental flaw in terms of failure resilience. If an internal node fails, its whole subtree becomes isolated. To overcome this inherent drawback, several enhancements have been proposed to the basic query-propagation approach.

```
SELECT AVG(volume),room FROM sensors
WHERE floor = 6
GROUP BY room
HAVING AVG(volume) > threshold
EPOCH DURATION 30s
```

Figure 2.6 Query to Monitor the Occupancy of the Conference Rooms on a Particular Floor of a Building

All of the approaches described above are basically based on traditional client/server computing model, where each sensor node sends its sensory data to a backend processing center. Because the communication bandwidth of a wireless sensor network is typically much lower than that of a wired network, a sensor network's data traffic may exceed the network capacity. We can use advantages of mobile agent technology for data collection process in WSN [21], [22].

Regarding mobile agent route planning, related work appears in the robotic community regarding robot navigation. For instance, [23] proposes a series of algorithms to decide the safest path a mobile robot must take to reach its destination. Obstacles are sensed by a sensor network and the safest path is computed at the sensor network as well. The mobile robot is then safely guided by the sensor network to its destination. In particular, they propose to use a combination of artificial potential fields and dynamic programming. Artificial potential fields compute the risk level at every node.

To efficiently deploy mobile agents in a WSN, the appropriate runtime support has to be present in the WSN. One of the first attempts to provide mobile code and reprogramming capabilities on a WSN is presented in [24]. In this approach (Mate), a byte-code interpreter virtual machine is installed on every node. This virtual machine hides the intrinsic of the underlying architecture and provides a finite set of common WSN-related instructions. In addition, through viral infection, the Mate applications being executed at the WSN are remotely updated.

In this thesis we have focused on the problem of energy efficient data collection in wireless sensor network. In [24], authors have proposed mobile agent based approach to solve the problem of overwhelming data traffic. In that they have assumed clustering in WSN. The assumption of clustering poses limitation on several applications of WSN. The limitation of clustering can be addressed by a flat sensor network. We have used mobile agent based approach for data collection in WSN for flat sensor network. We have designed and implemented a novel approach for energy efficient data collection in WSN.

CHAPTER 3

PROPOSED APPROACH FOR DATA COLLECTION

3.1Motivation

For proposed approach for data collection in wireless sensor network we used mobile agent based computing model instead of client/server based computing model. MA based computing model is introduced in [25] and its advantages over the traditional client/server computing model are discussed.

The main contribution in our work is as following.

1. Here, we develop energy efficient data collection approach for wireless sensor network.
2. We use mobile agent based computing model rather than traditional client/server based computing model to reduce energy consumption.
3. The entire approach is broadly divided in two parts. First part is path establishment. We flood interest message to get information about available source nodes in sensor network. At the same time source nodes sends exploratory data to provide information about their presence to sink node. Second part is data collection procedure.
4. Data collection procedure involves mobile agent route planning. We have divided this process into mobile agent migration from sink node to first source node, first source node to last source node and last source node to sink node.

3.2 Mobile Agent Based Computing Model

As described in Chapter 2, WSNs have presented unique challenges to many aspects of network design and information processing. In order to respond to these challenges the underlying data collection techniques need to be scalable, adaptive, energy-aware and capable of delivering reliable information in real time. Furthermore, data collection techniques should provide progressive accuracy as the collaboration process could be terminated upon achieving desired accuracy to conserve energy.

In data collection process the most commonly used computing model is client-server model. Here, individual sensors act as clients. Processing center acts as the server. Individual sensors (clients) send raw data or preprocessed data to processing center (server). So, data collection is carried out at center (server). There are few drawbacks of this client/server based computing model. It requires many round trips over the network in order to complete one transaction. Each transaction consumes network bandwidth and communication energy. Also, there are some head nodes in WSN with higher computing capabilities, bigger storage and more energy. These head nodes acts as processing center. But in some automatic and homogenous sensor networks, this is not always be the case. Given the unreliability and low bandwidth of the wireless link used in sensor networks, the client/server based computing is not appropriate to carry out the data collection between multiple sensor nodes. So, we introduce Mobile Agent (MA) based computing model.

In MA based computing model, instead of each sensor node sending raw data or pre-processed data to processing center, the processing code is moved to the data locations through mobile agents. Figure 3.1 presents MA based computing model.

This model has following features.

- The performance of the network is not affected when the number of sensor nodes is increased. This addresses scalability issue.
- Mobile agents can be sent when the network connection is alive and return results when the connection is re-established. Therefore, the performance of the MA-based computing is not affected much by the reliability of the network.
- Mobile agents can be programmed to carry different task-specific integration processes which extend the functionality of the network.

- The itinerary of the mobile agent is dynamically determined based on both the information gain and energy constraints. It is tightly integrated into the application and is energy-efficient.

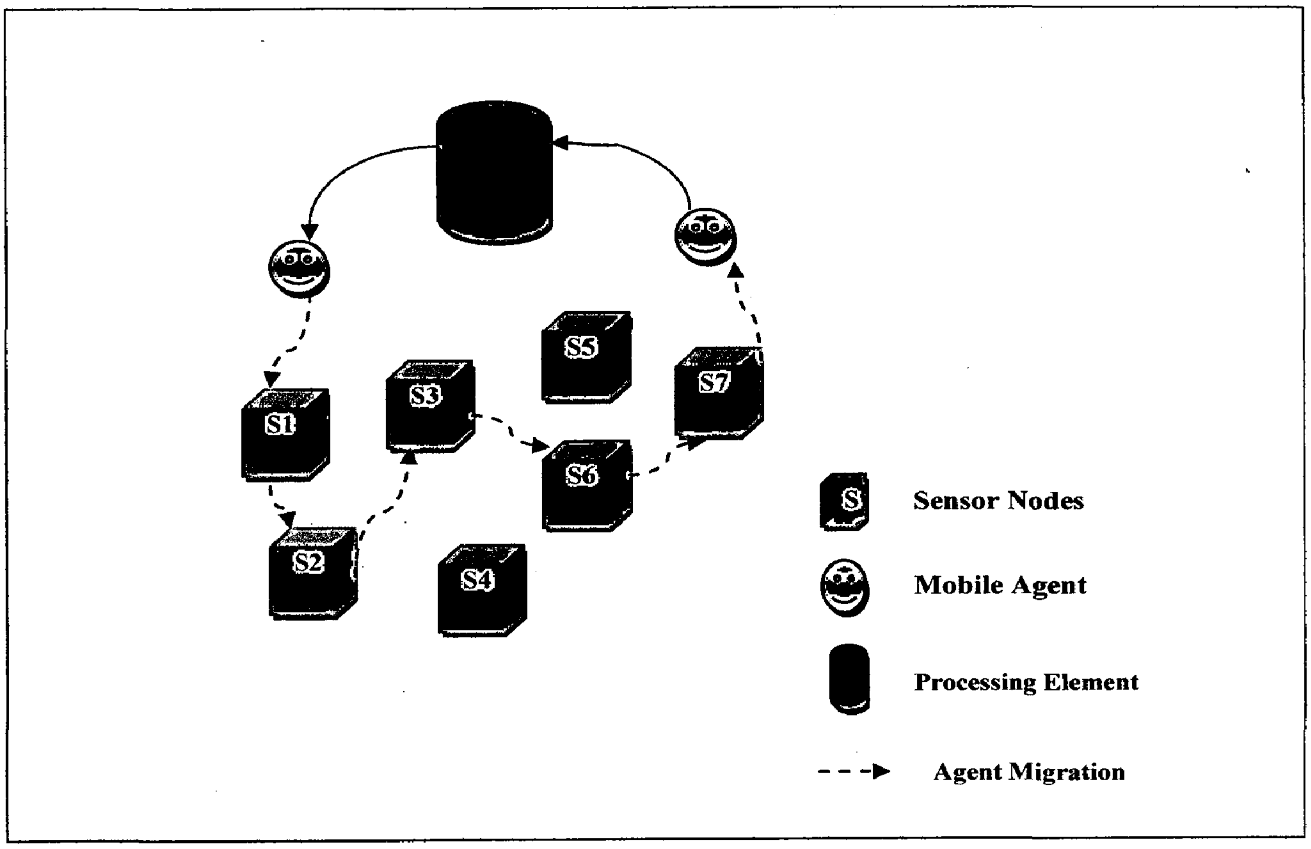


Figure 3.1 Mobile agent based computing model

- A mobile agent always carries a partially integrated result generated by nodes it already visited. As the mobile agent migrates from node to node, the accuracy of the integrated result is constantly improved assuming the agent follows the path determined based on the information gain. Therefore, the agent can return results and terminate its itinerary any time the integration accuracy satisfies the requirement. This feature, on the other hand, also saves both network bandwidth and computation time since unnecessary node visits and agent migrations are avoided.

3.3 Structure of Mobile Agent

Mobile Agent is a special kind of software. Once dispatched, it can migrate from node to node, performing data processing autonomously. Figure 3.2 shows the structure of mobile agent. It shows MA has attributes namely identification, itinerary, processing code, host, status and other detail.

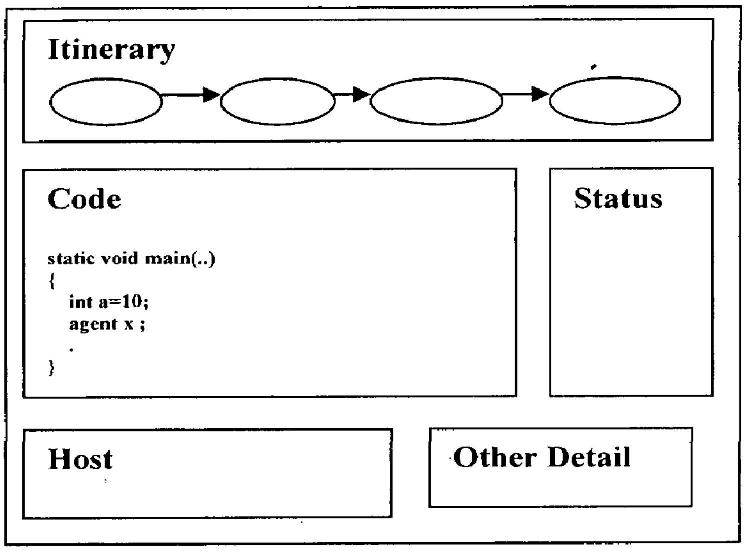


Figure 3.2 Structure of Mobile Agent

A mobile agent generally consists of the following components, as illustrated in Figure 3.2: Itinerary records the route and current position of the agent; Code stores fragments of the program; State records agent status and Host stores the server position; Other necessary details store other information related to the agent, so that operators will know what the agent does and who the owner is.

3.4 Mobile Agent Route Planning

3.4.1 Assumptions

- Here we assume sensor network architecture to be flat. Flat sensor network architectures are suitable for wide range of applications.
- We consider mobile agent (MA) in multihop environments with the absence of cluster head.

- The target sensor nodes are geographically close to each other when compared with the distance to the sink node.
- Only those source nodes whose interest packets match will store the processing code carried by an MA. The sink does not flood processing code to the whole network, since the associated communication overhead may be too high.
- Processing code is stored in the source node when the MA visits it at the first time. The processing code will be operating until the task is scheduled to finish. It may be discarded when the task is finished.
- The locally processed data in each source node will be aggregated into the accumulated data result of the MA by a certain aggregation ratio.

3.4.2 Basic Design

We propose gradient based solution to decide in which order source nodes to be visited. Figure 3.3 describes how data collection is carried out at source nodes and then collected data sent finally to sink node. As shown in figure, sink node diffuses an interest for exploratory events which are intended for path setup.

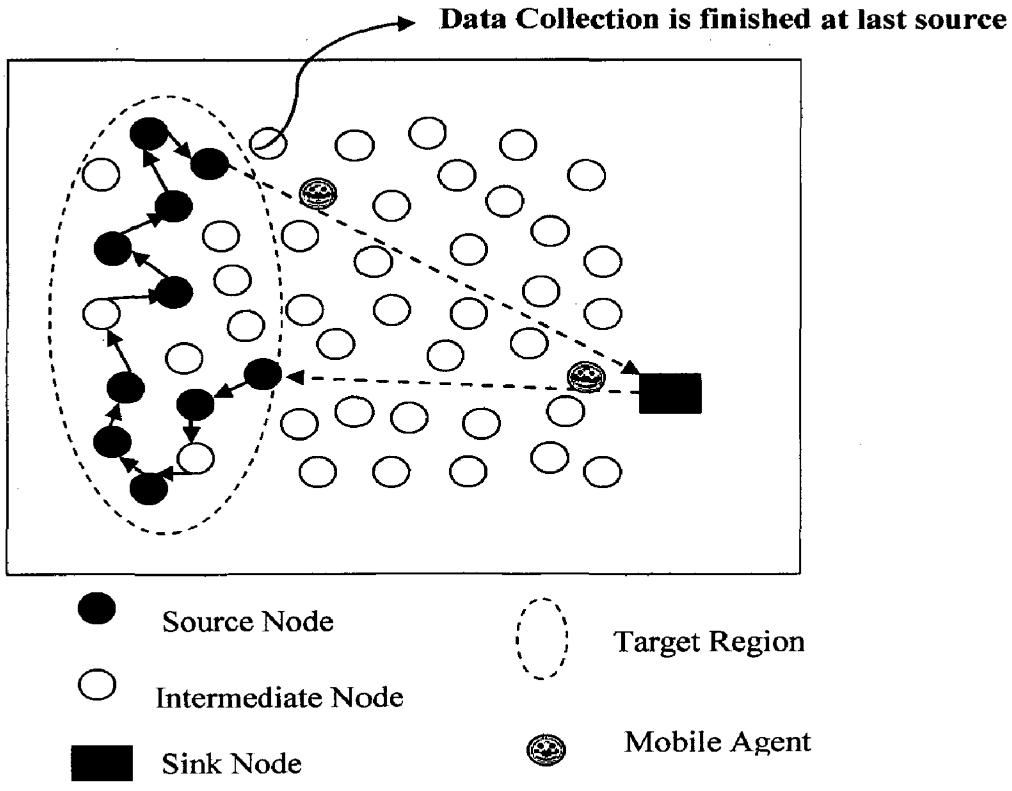


Figure 3.3 Basic design

Once target sources receive the corresponding interest, they send exploratory data, possibly along multiple paths, toward the sink. The gradients set up for exploratory events are called exploratory gradients. Once gradients are set up, sink sends MA for data collection to target source nodes matching sink's interest. Sink selects first and last source node. The sink also reinforces the path to the last source node. When MA reaches first source node, it is stored in the source node. MA travels from first source to last source. Multiple rounds are carried out for this process for collecting desired data. Finally, Reinforced path from last source node to sink node will be used by MA to traverse sink node.

3.4.3 Data Aggregation

Mobile Agent aggregates individual sensed data when it visits each target source.
A sequence of data result can be fused with an aggregation ratio μ ($0 <= \mu <= 1$).
Let D^i_{ma} be the amount of accumulated data result after the MA leaves source i.
Let d_i is the amount of data that will be aggregated by μ.

$D^i_{ma} = D^{i-1}_{ma} + (1 - \mu) \cdot d_i$.   or

$$D^i_{ma} = d_1 + \sum_{k=2}^{k=i} (1 - \mu) \cdot d_k.$$

In equation, at first source node no data aggregation takes place. The value of μ is dependent on type of application. For example in image processing application, when we fuse two Region of Interest (ROI) images, effective data fusion can be attained only if statistical characteristics of the image are known (for example, Slepian-Wolf coding schemes).

3.4.4 Data Redundancy Reduction

MA based approach provides efficient way of dynamically deploying new application. It also allows a source node to perform local processing on the raw data as requested by the application. This capability enables a reduction in the amount of data to be transmitted.

Let α $(0 < \alpha < 1)$ be the reduction ratio. Let D^{i}_{data} be the size of raw data at source i. Let d_i be the size of reduced data. Then, $d_i = D^{i}_{data}(1 - \alpha)$

3.5 Detailed Design

3.5.1 Design of Mobile Agent Packet Format

SID	**SeqNo**	**FS**	**LS**	**ICount**	**IFlag**
NS	**NH**	**SFlag**	**List**		

Figure 3.4 Design of mobile agent packet format

MA packet's identification is represented by pair (SID, SeqNo). SID is Sink Identifier. When sink sends new MA packet, SeqNo is incremented. FS represents first source node and LS represents last source node. MA first visits node represented by FS and visits eventually to the last node represented by LS. So, data collection by MA begins with FS and ends at LS. ICount value is associated with the current value of the data collection round. Initially, sink sets this value to 1. FS increments value of ICount for each new round. IFlag value is set by FS. IFlag value indicates that current round is last round of data collection. When MA with IFlag set arrives at a source node, it can make the system unmount the corresponding processing code after its execution.

NS indicates next source node to be visited by MA. NH indicates immediate next hop node (it can be intermediate sensor node or destination source node). If NH equals NS then the next hop node is destination node. List contains the identifiers (IDs) of target sensor nodes that remain to be visited in current round. When MA is created List contains all source node IDs. After MA visits particular source node that ID is deleted from the list. When all target source nodes have been visited, SFlag is set which indicates that destination of MA is sink node.

3.5.2 Working of Proposed Approach

(a) Sink node floods interests and Source node floods exploratory data

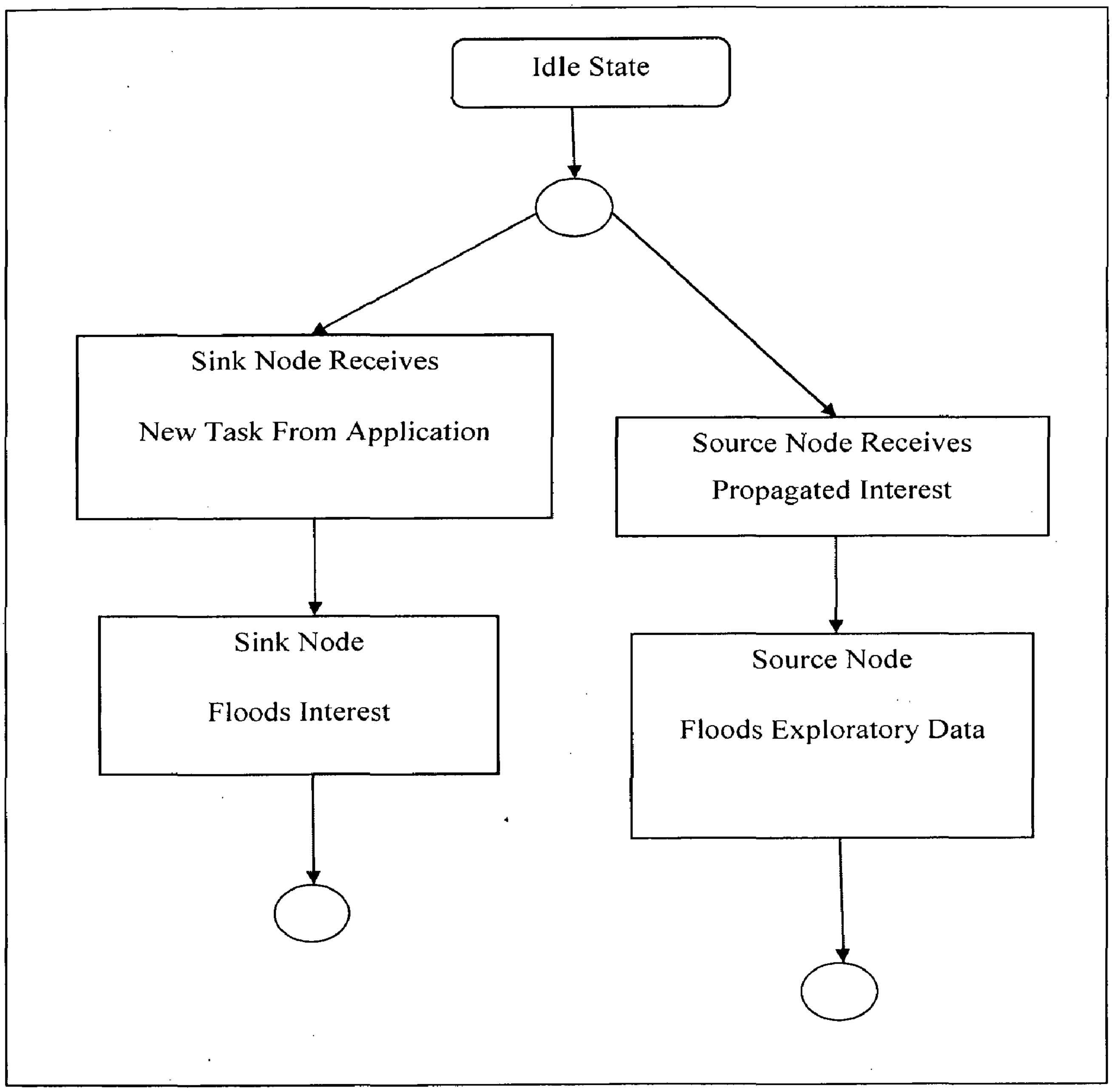


Figure 3.5 Initial Operations of Sink node and Source nodes

As per shown in Figure 3.5, Application requests a new task to sink node. Sink node then floods interest packet to find out source nodes which will perform the task. On the other side, if source nodes receive interest, they send exploratory data to the sink node individually. Now, sink node receives these exploratory data and prepares a list of source nodes that will be visited by MA.

MA related operations begin when sink create MA and dispatches it. MA operation ends when MA returns to the sink after data collection. These operations are shown in Figure 3.6, 3.7 and 3.8. Figure 3.6 shows mobile agent creation and dispatch procedure. Figure 3.7 shows steps carried out during data collection process. Figure 3.8 shows procedure about mobile agent finishing task and returns to sink node.

The MA route can be divided into three parts; from sink to FS, FS to LS, and LS to the sink. Each source node is expected to generate the sensory data periodically with some interval, which means same code (MA) needs to be stored for multiple running. Therefore, when MA arrives at FS, it will be stored.

Then FS sets Timer. Timer is used to trigger the next round to dispatch the MA to collect data from the relevant source again. The time period between two successive rounds of data collection will be equal to sensory data generating rate which is set to the value of the Timer. This round will be repeated until the task is finished. At the end of last round the task is finished. When Timer expires, FS starts a new round by dispatching the MA along all the sensors. After MA visits last source, it discards the processing code and carries aggregated result to the sink.

(b) MA creation and dispatch by Sink node

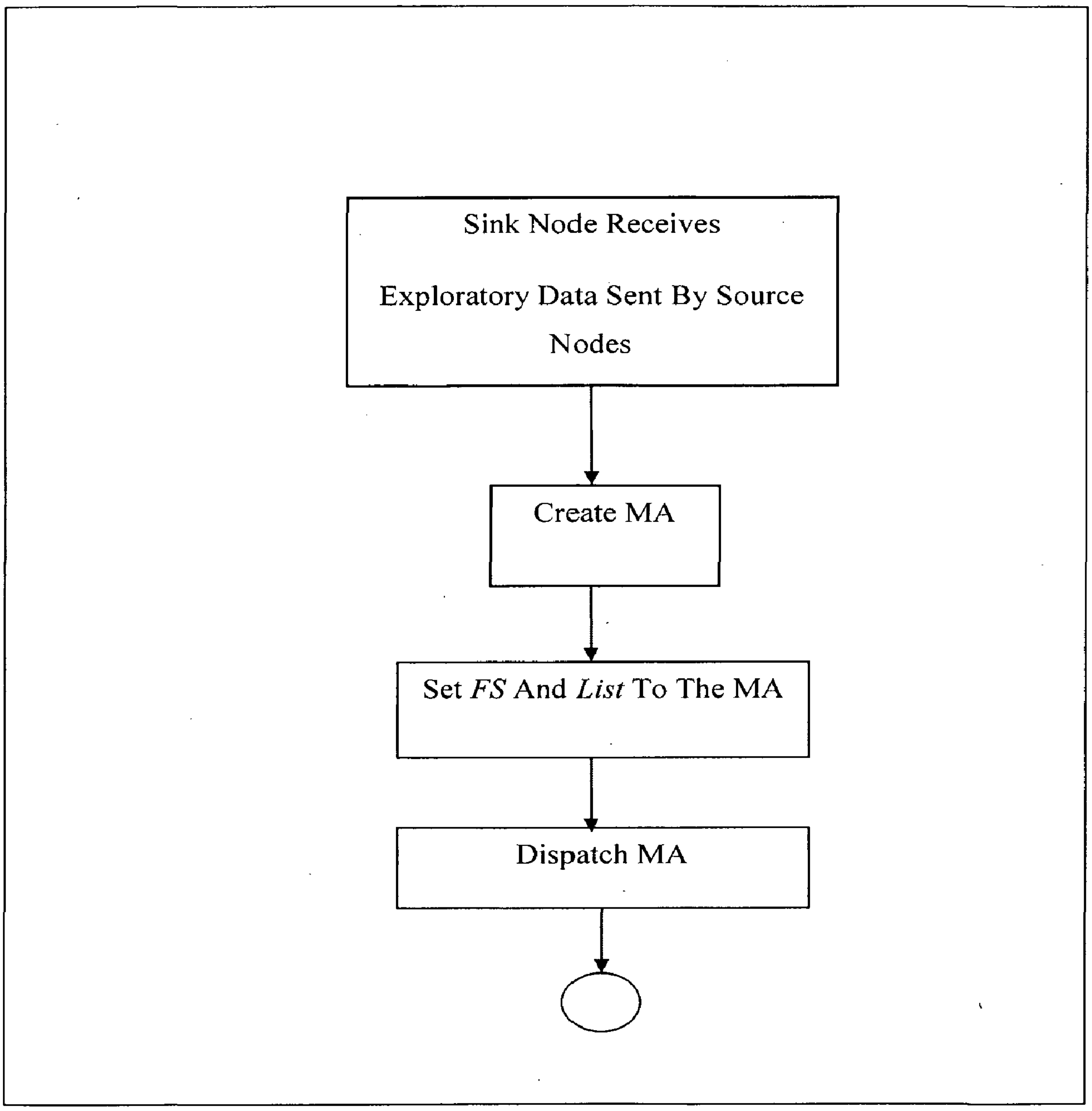

Sink Node Receives
Exploratory Data Sent By Source Nodes
Create MA
Set *FS* And *List* To The MA
Dispatch MA

Figure 3.6 MA Creation and Dispatch

(c) Data collection process

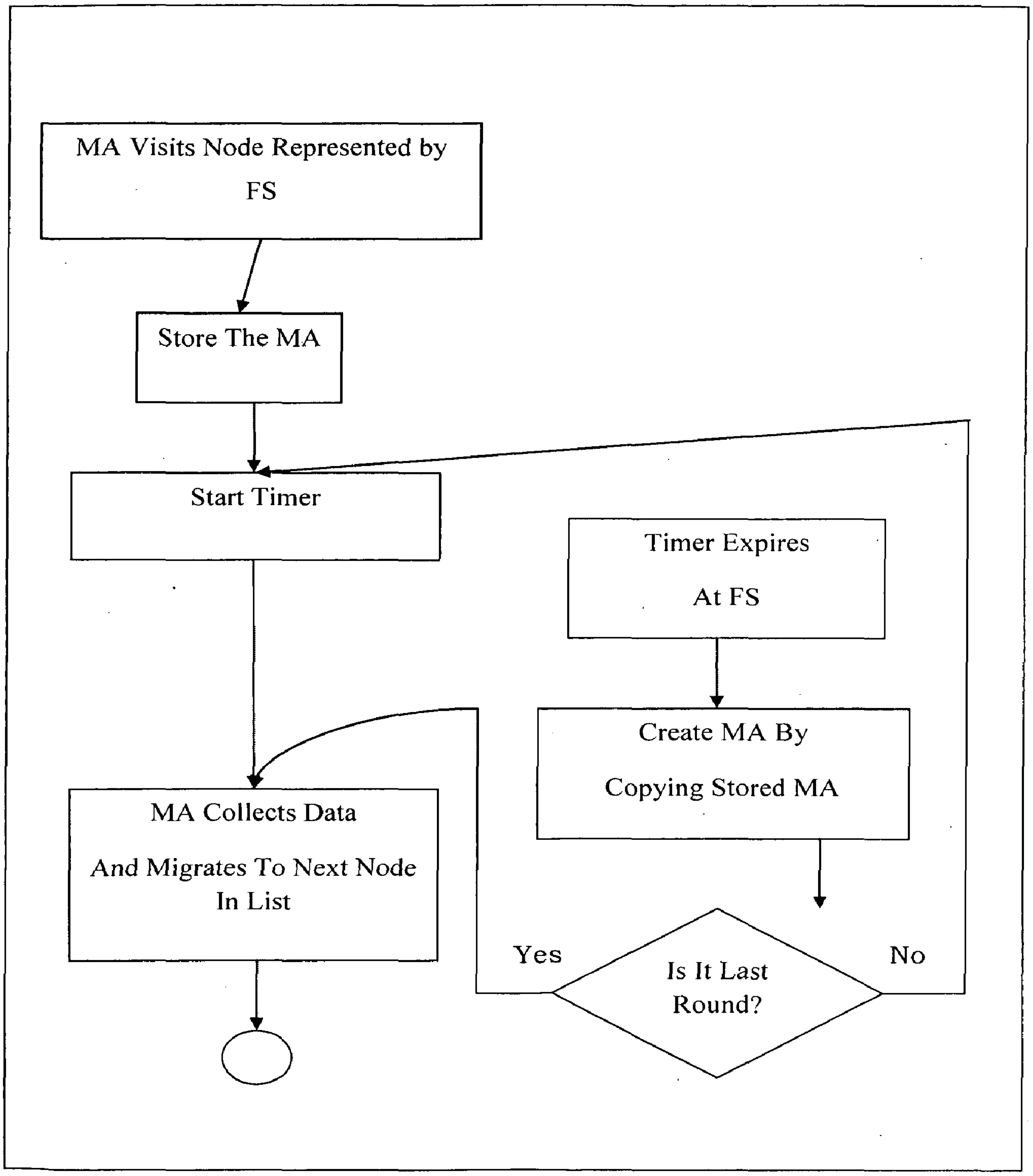

MA Visits Node Represented by FS
Store The MA
Start Timer
Timer Expires At FS
Create MA By Copying Stored MA
MA Collects Data And Migrates To Next Node In List
Is It Last Round?
Yes
No

Figure 3.7 Data Collection Process

(d) Migration of mobile agent to sink node

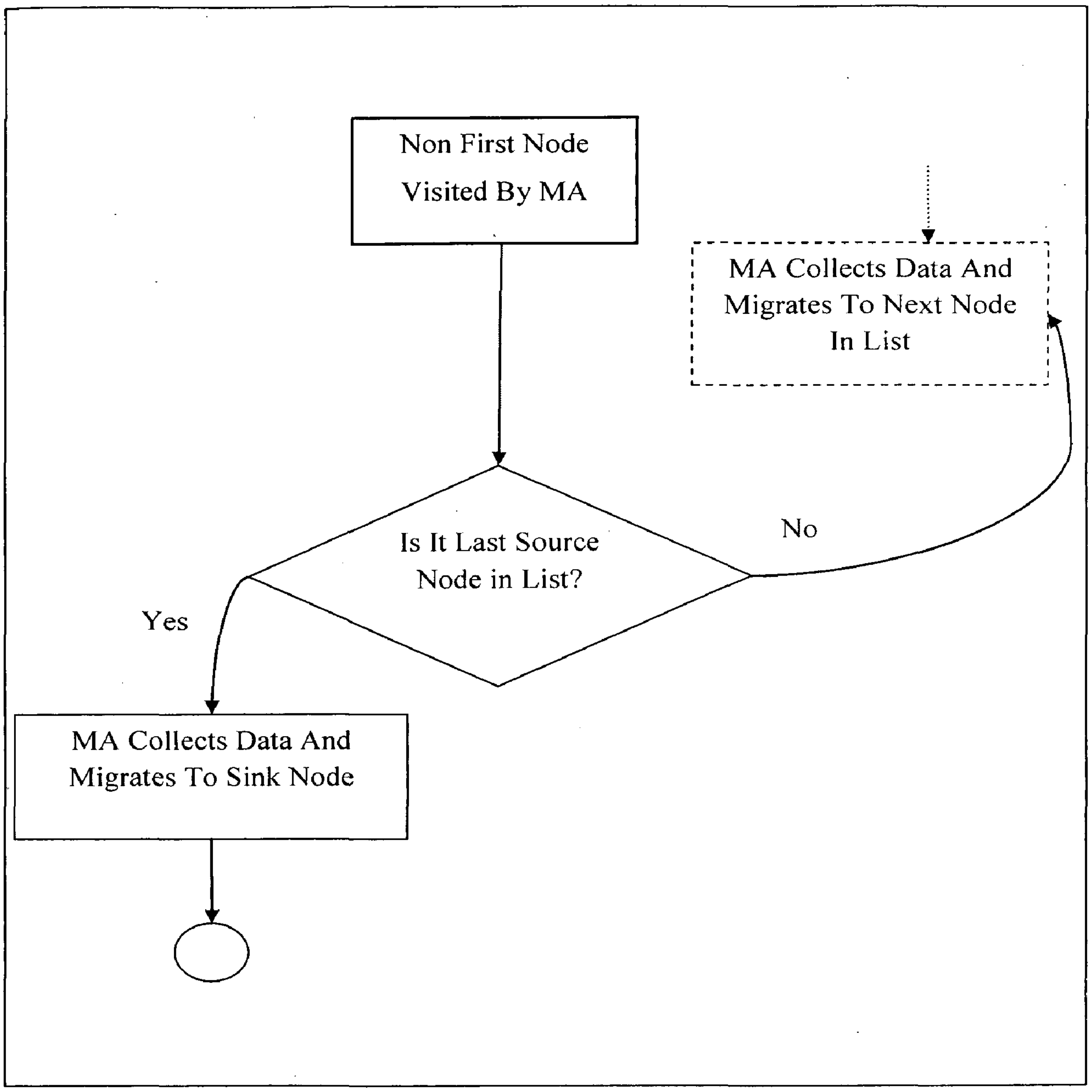

Non First Node Visited By MA
MA Collects Data And Migrates To Next Node In List
Is It Last Source Node in List?
No
Yes
MA Collects Data And Migrates To Sink Node

Figure 3.8 MA Migrates to Sink Node

Among these target source nodes, sink node choose first source node and last source node. It then dispatches MA with the packet structure described in 3.5.1 and sends to MA to first source node. At the same time sink node reinforces path to last source node. When MA arrives at the first source node, it is stored in the node. The whole task is divided into rounds, where each round requires the MA to visit all the chosen target sensors and to return the data result to the sink node. Figure shows, S1, S2, S3 and S4 are source nodes to be visited by MA. S1 is first source node and S4 is last source node. Reinforced path from S4 to S is shown. Nodes with number represent intermediate nodes. Node S is sink node.

(d) Migration of mobile agent to sink node

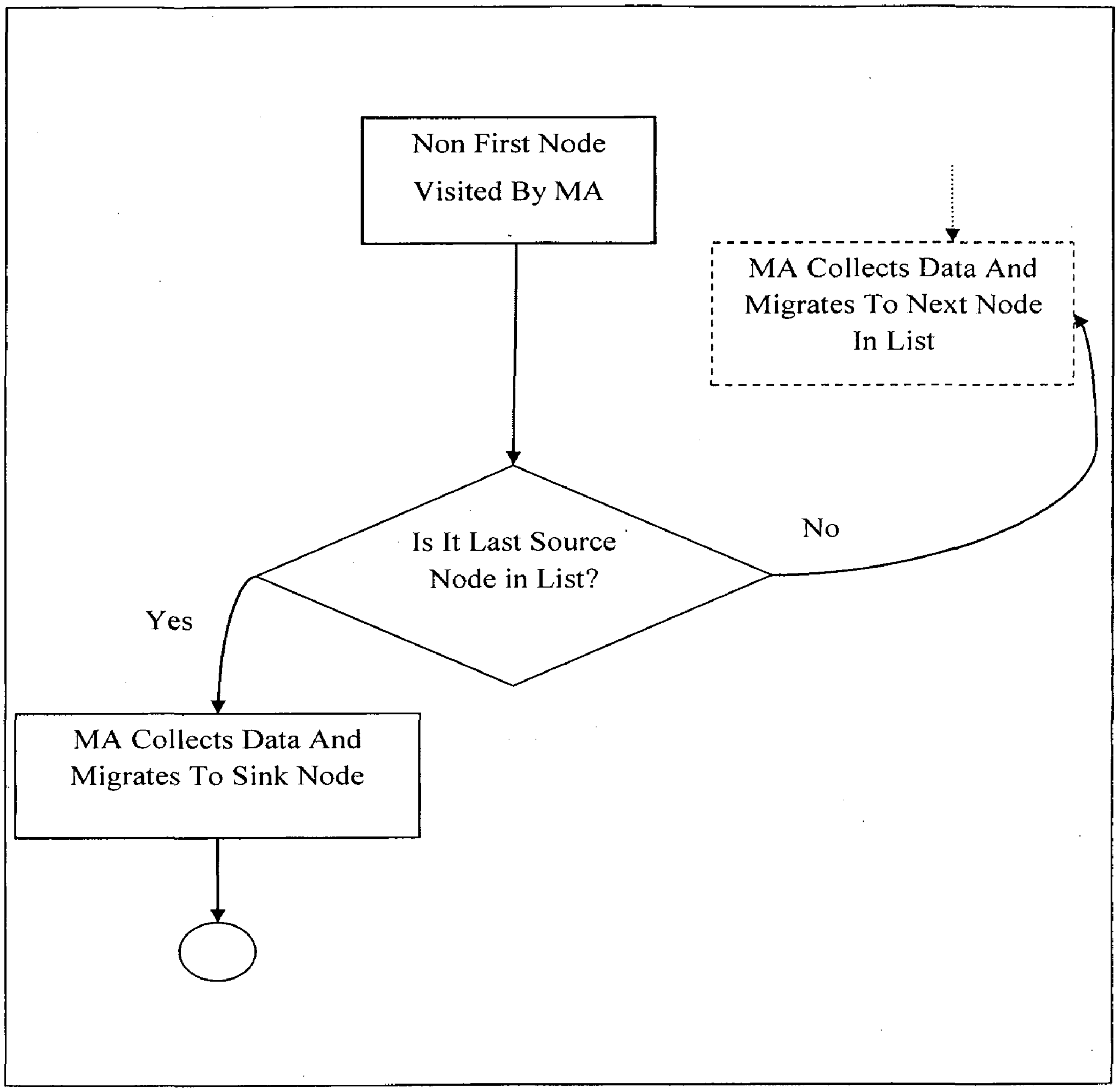

Non First Node Visited By MA
MA Collects Data And Migrates To Next Node In List
Is It Last Source Node in List?
No
Yes
MA Collects Data And Migrates To Sink Node

Figure 3.8 MA Migrates to Sink Node

So, we can summarize that MA based data collection process is divided into two part. First part deals with the path (gradient) setup between sink node and source nodes. Second part deals with the data collection process by creating and dispatching MA.

Figure 3.9 explains second part of the process by means of example. At the end of the first part, the target sensor nodes generate multiple exploratory message flows to the sink node. Since ultimate goal in sensor network is detection of event, sink node may stop handling any exploratory message flows if it considers that the number of source nodes is large enough to meet the requirement of reliable event detection. Thus, sink node may choose all source nodes or subset of source nodes for MA visit.

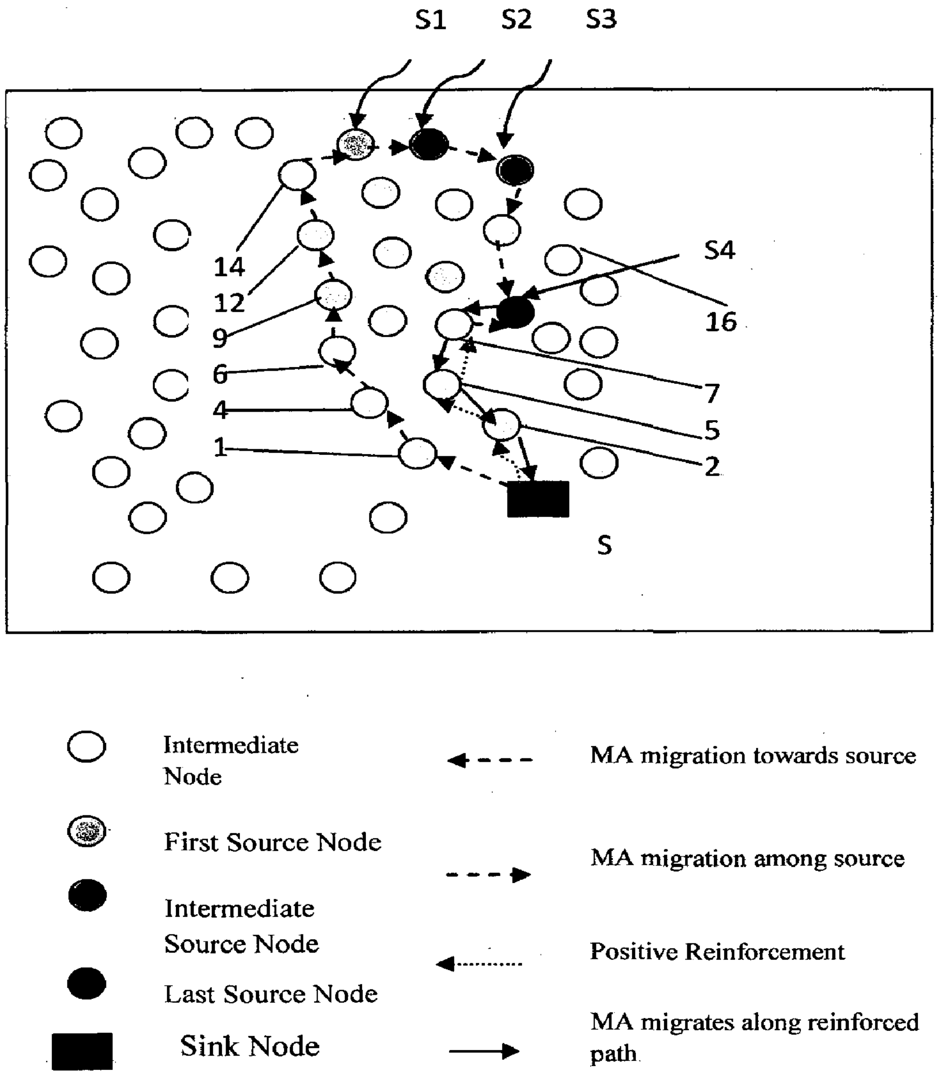


Figure 3.9 Second part of proposed approach

Among these target source nodes, sink node choose first source node and last source node. It then dispatches MA with the packet structure described in 3.5.1 and sends to MA to first source node. At the same time sink node reinforces path to last source node. When MA arrives at the first source node, it is stored in the node. The whole task is divided into rounds, where each round requires the MA to visit all the chosen target sensors and to return the data result to the sink node. Figure shows, S1, S2, S3 and S4 are source nodes to be visited by MA. S1 is first source node and S4 is last source node. Reinforced path from S4 to S is shown. Nodes with number represent intermediate nodes. Node S is sink node.

CHAPTER 4

IMPLEMENTATION DETAIL

4.1 NS-2 Simulator

NS2 is an object-oriented, discrete event-driven network simulator developed at UC Berkeley written in C++ and OTcl [26]. NS2 is very useful for developing and investigating variety of protocols. These mainly include protocols regarding TCP behavior, router queuing policies, multicasting, multimedia, wireless networking and application-level protocols.

4.1.1 Software Architecture

NS2 software promotes extensions by users. It provides a rich infrastructure for developing new protocols. Also, instead of using a single programming language that defines a monolithic simulation, NS uses the split-programming model in which the implementation of the model is distributed between two languages. The goal is to provide adequate flexibility without losing performance. In particular, tasks such as low-level event processing or packet forwarding through simulated router require high performance and are not modified frequently once put into place. Hence, they can be best implemented in compiled language like C++. On the other hand, tasks such as the dynamic configuration of protocol objects and exploring a number of different scenarios undergo frequent changes as the simulation proceeds. Hence, they can be best implemented in a flexible and interactive scripting language like OTcl. Thus, C++ implements the core set of high performance primitives and the OTcl scripting language express the definition, configuration and control of the simulation.

4.1.2 C++ - OTcl Linkage

NS supports a compiled class hierarchy in C++ and also similar interpreted class hierarchy in OTcl. From the user's perspective, there is a one-to-one correspondence (Figure 4.1) between a class in the interpreted hierarchy and a class in the compiled hierarchy. The root of this class hierarchy is the class TclObject. Users create new simulator objects through the interpreter. These objects are instantiated within the interpreter and are closely mirrored by a corresponding object in the compiled hierarchy. The interpreted class hierarchy is automatically established through methods defined in class TclClass while user instantiated objects are mirrored through methods defined in class TclObject.

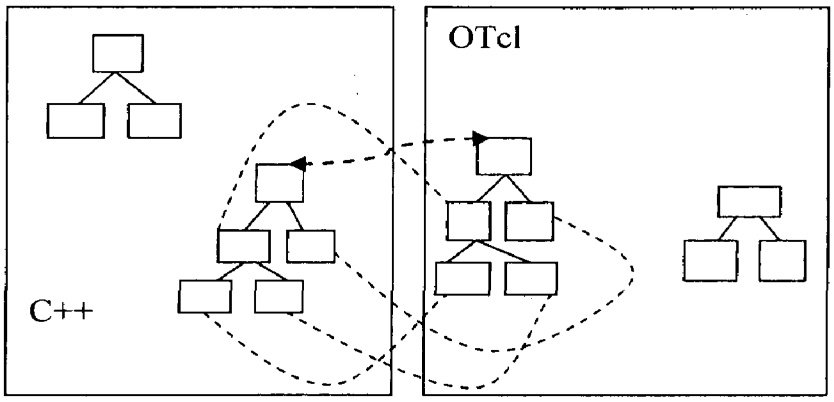


Figure 4.1.C++ --OTcl linkage.

4.1.3 Why NS2

NS2 is a publicly available common simulator with support for simulations of large number of protocols. It provides a very rich infrastructure for developing new protocols. It also provides the opportunity to study large-scale protocol interaction in a controlled environment. Moreover, NS software really promotes extension by users. The fundamental abstraction the software architecture provides is "programmable composition". This model expresses simulation configuration as a program rather than as a static configuration.

In addition, in our work, we needed to implement our data collection approach for wireless sensor network. To simulator wireless sensor network we have used WSN framework for NS2 [27]. We also needed to extend some functionality for mobile agent support, which we find is suitable to be done in NS2.

4.2 Functionalities Implemented

We have implemented the following functionalities for mobile agent.
Context class provides the execution environment required for agent's execution including creating any number of agents, retracting agents from a remote site, disposing agent, registering agent and actual transferring of agent to another context. By registering agents, it maintains the list of currently executing agents in a hash table mapped with the agent's ID. In order to transfer an agent's reference, as discussed before, it needs to define an ADU (Application Data Unit) that derives from the class AppData. AppData is the base class for all ADUs. The class thus defined to carry agent's reference is named as MobileAgentData.

Methods provided by Context class are as follows:
void process_data(int, AppData)*: It over-rides the behaviour of TcpApp and is the principle method for processing the incoming agent. It retrieves the agent's reference stored in the MobileAgentData and loads and starts the agent by calling agent's startArrivedAgent() method.
void retractAgent(const char)*: It is used to draw back the agent from a remote site.
void registerAgent(MAgent, int)*: It is used to initialize the agent and register in the agent's table.
void disposeAgent (int): It just erases the agent from the table.

bool is_active(): Checks if the agent is currently in ACTIVE state or not.

As MAgent class contains some abstract methods, it cannot be instantiated. To instantiate an agent object, one must extend from the MAgent class. To facilitate this extension in both C++ and OTcl, a new class named MAgentInst is created by inheriting from MAgent class. It is actually this class's object that shadows with the OTcl class MAgent. It is used to create a default implementation of all the abstract methods in MAgent class to call the corresponding method in OTcl and thus allowing extending the MAgent class directly in OTcl.

4.3 Simulation Settings

4.3.1 Sensor network model

Some of the components of sensor network model are following:
Sensor Node: Sensor node is any sensor node in the wireless sensor network. Sensor nodes perform sensing, communication and processing. There are three types of sensor nodes we have in simulation of sensor network model: sink node, source node and intermediate node.

Sink Node: Sink node is sensor node; responsible for getting request for data collection from application. It then creates mobile agent and sends it to source nodes for data collection.

Source Node: Source node is sensor node which is detecting event and sending data to sink node periodically. There are many source nodes which are deployed near the event.

Intermediate Node: Intermediate node is sensor node which comes in the path between sink node and source node. Mobile agent travels through intermediate nodes to source nodes and return back through intermediate nodes to sink node.

Sensor Field: Sensor nodes are deployed in an area which is close to the event. This area is sensor field. Nodes deployed in sensor field can detect the event and transmit data to the processing center or sink node.

Here, periodic transmission of data packets takes places for each task with a constant bit rate 1 packet/second. Here, we assume that sink node has sufficient energy supply which is equal to infinite energy when compared with source nodes. Source nodes are battery operated with limited energy. We also assume that source and sink nodes are stationary. Source nodes are located at one corner of the area, while the sink node is located on the other corner of the area. Every node starts with same initial energy.

4.3.2 Simulation Parameters

Energy Calculation:

Energy consumption is calculated as per following equation:

$\text{Energy} = m \times \text{Size} + \text{b} + P_{\text{idle}} \times t \times 1000\ (\mu\text{W}\cdot \text{s})$

m is incremental cost compared to power consumption in idle state, b represents the fixed cost independent of packet size, t represents the duration of state. Size represents packet size.

The energy model we used here is as per [28].

Parameter Values:

Parameter Values are tabled in Table 4.1 and Table 4.2

Table 4.1 Energy Consumption parameters configuration of lucent IEEE 802.11 WaveLAN card

Normalized Initial Energy of Sensor Node (W · s)	4500	
Incremental Cost (μW· s / bytes)	m_{tx}	1.9
	m_{recv}	0.5
Fixed Cost (μW· s)	b_{tx}	454
	b_{recv}	356
P_{idle} (mW)	843	

Table 4.2 Simulation Parameters

Network Size	300m × 300 m
Topology Configuration Mode	Randomized
Total Sensor Nodes	600
Data Rate	1 Mbps
Transmission Range of Sensor Node	30m
Source Nodes	5
Sensed Data Size	1KB
Sensed Data Packet Interval	1 s
Duration	300s

4.4 Performance Metrics

We have considered following performance metrics to evaluate performance of our approach.

Energy Consumption

Energy consumption which is denoted by ***e***, is the ratio of network energy consumption to the number of data packets delivered successfully to the sink. The network energy consumption includes all the energy consumption by transmitting and receiving during simulation.

We do not account energy consumption for idle state, since this part is approximately the same for the both schemes simulated. Let E_{total} be the energy consumption by transmitting, receiving and overhearing during simulation and n_{data} denotes number of data packets delivered to the sink.

Therefore, Energy consumption,

$$e = E_{total} / n_{data}$$

Packet Delivery Ratio (Reliability)

Packet delivery ratio which is denoted by P, is the ratio of the number of data packets delivered
to the sink to the number of packets generated by the source nodes.

Average end-to-end packet delay

Average end-to-end packet delay which is denoted by T_{ete}, includes all possible delays during data dissemination, caused by queuing, retransmission due to collision, and transmission time.

Energy * delay / Reliability

In wireless sensor network, it is important to consider both energy and delay.
Here, energy * delay metric can reflect both the energy usage and the end-to-end delay.
Furthermore, in unreliable environment, the reliability is also important metric.

$$\eta = e . T_{ete} / P$$

CHAPTER 5

SIMULATION RESULTS AND DISCUSSION

Results and Discussion

We compare results carried out by our approach with the directed diffusion approach presented in [29]. Some of important parameters chosen for comparison are duration of task, reduction ratio (α), aggregation ratio (μ), size of sensed data of each sensor (D_{data}). When our approach is applied to wide range of applications, the consideration of varying both reduction ratio and aggregation ratio is necessary. In the image processing application, if target camera sensors are sparsely distributed, the redundancy between two ROI images is low which means that the value of μ would be small. In the following subsections we present comparisons under different conditions for both approaches: proposed approach and directed diffusion.

5.1Effect on energy consumption by changing task duration

In this experiment, we change T_{task}, from 10 seconds to 600 seconds. As shown in Figure 5.1 as T_{task} increases energy consumption *e* decreases in both approaches. When the T_{task} is small (< 60 seconds) our approach consumes more energy compared with directed diffusion.

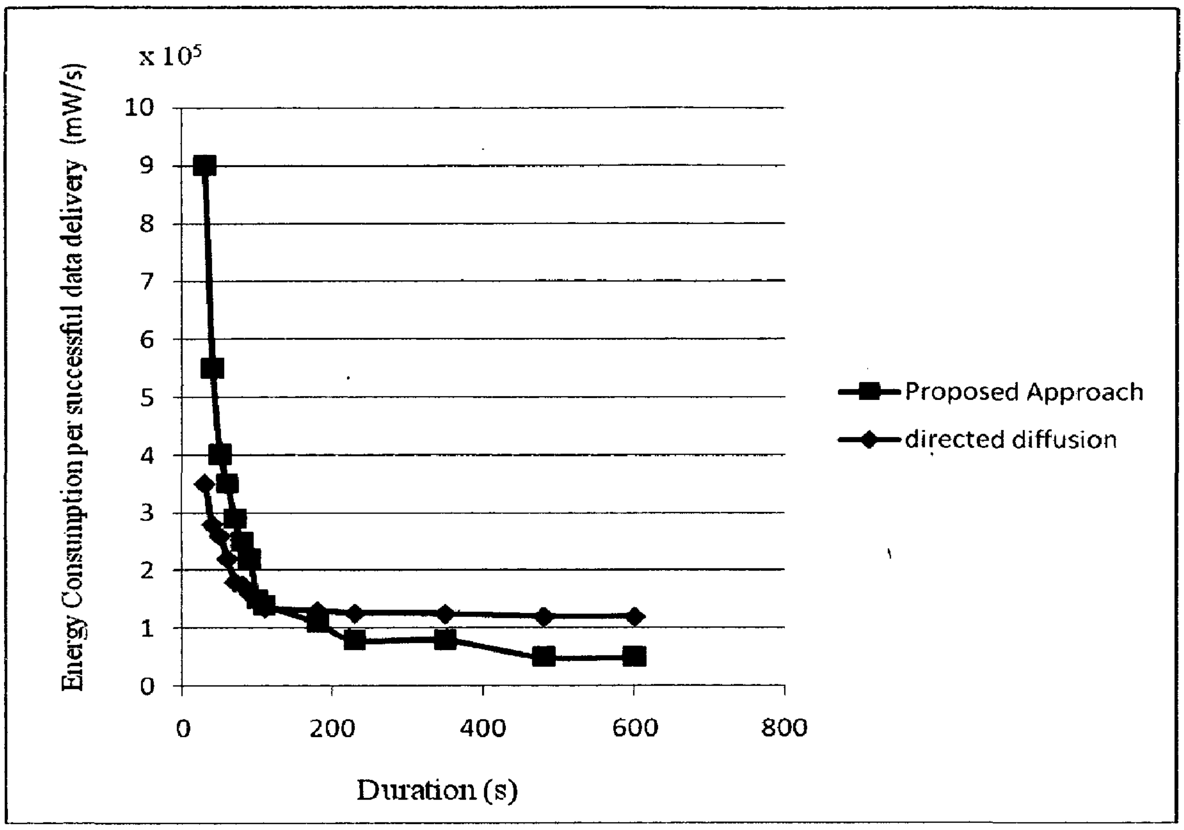


Figure 5.1 Effect of Task duration on energy consumption

This is due to some fixed additional energy consumption for processing code. If T_{task} is small, n_{data} is small and energy consumption, e is large. However, when T_{task} is more than 90 seconds with α=0.8 and μ = 0.2, our approach has lower energy consumption than directed diffusion. Thus our approach performs better in applications where source nodes process enough long streams of data.

5.2 Effect on average end to end delay by changing mobile agent accessing delay

Mobile agent accessing delay represented by τ, is the time for an MA to amount processing code in target source node. Here, we change delay τ from 0 to 0.05 seconds. As shown in Figure 5.2, Average end-to-end delay for directed diffusion approach (T_{dd}) is constant since changing τ has no effect on directed diffusion. As τ is introduced in proposed approach, it causes average end-to-end delay (T_{ma}) increase fast. In Figure 5.2 when it is beyond 0.042 seconds with α=0.8 and μ = 0.2, our approach has larger end-to-end delay than directed diffusion.

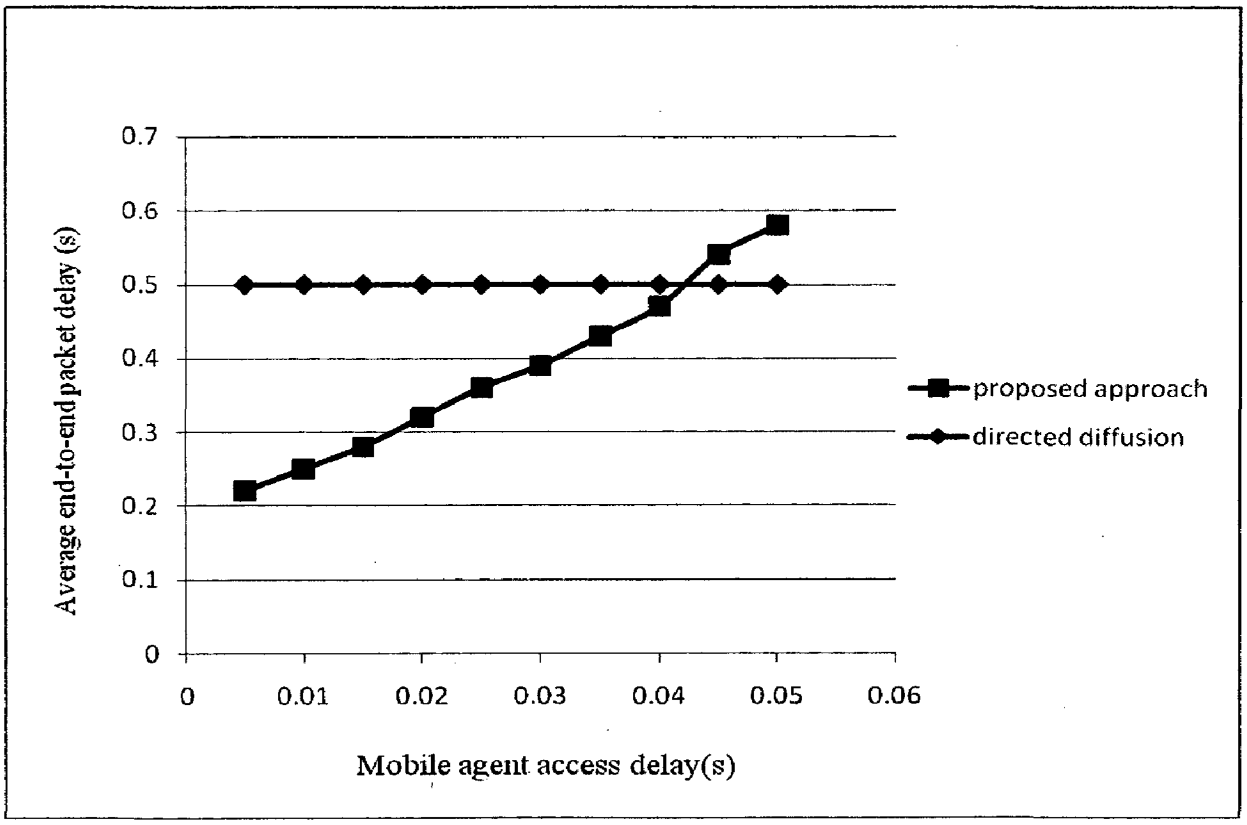


Figure 5.2 Effect of MA delay on end-to-end delay

5.3 Effect on packet delivery ratio by changing size of sensed data

Here, we change the size of sensed data of each sensor (D_{data}) from 0.5 KB to 2KB by increasing 0.25 KB each time and keep the other parameters unchanged. As shown in Figure 5.3, proposed approach always outperforms directed diffusion in terms of P. In our approach, only single data flow is sent for each round. In contrast, multiple data flows from individual source nodes are sent in directed diffusion. Thus, Congestion is more likely in directed diffusion. When D_{data} increases, the congestion is more serious and P of directed diffusion will decrease more.

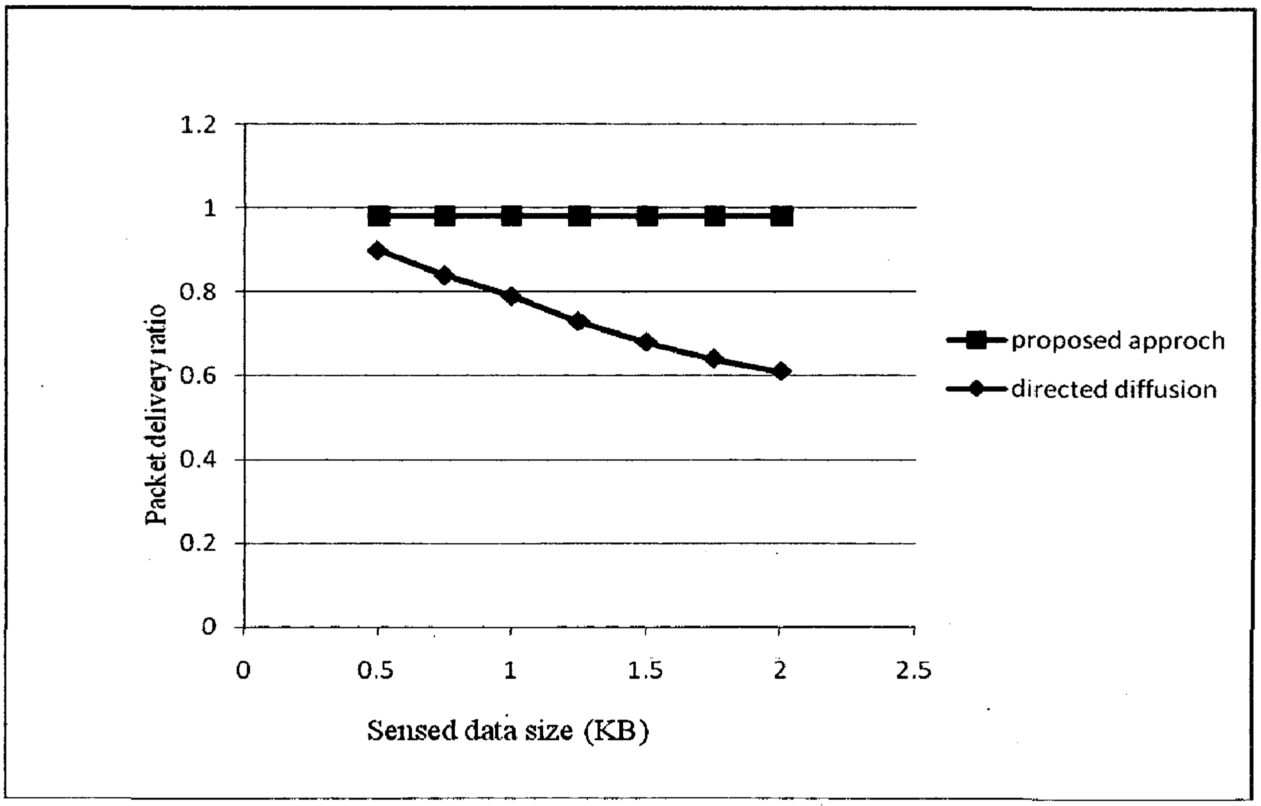


Figure 5.3 Effect of sensed data size on packet delivery ratio

5.4 Effect on energy consumption by changing size of sensed data

As shown in Figure 5.4, the energy consumption of directed diffusion is larger than that of our Mobile agent based approach in most cases. The larger the value of α or μ, the smaller is e in our approach. When α=0.9 (RR) and μ= 1 (AR), e is lowest among all cases. If μ=1, all the sensory data will be fused into a data with fixed size. To take best advantage of our approach the value of μ should be higher. To take conservative approach in evaluation, we take small μ. By taking μ = 0.2 fixed, when α is more than 0.6, e of our approach is always less than that of directed diffusion. Performance gain increases as α increase. When α is less than 0.4, our approach tends to have larger e. This is because the smaller is α, the larger is the size of accumulated data result.

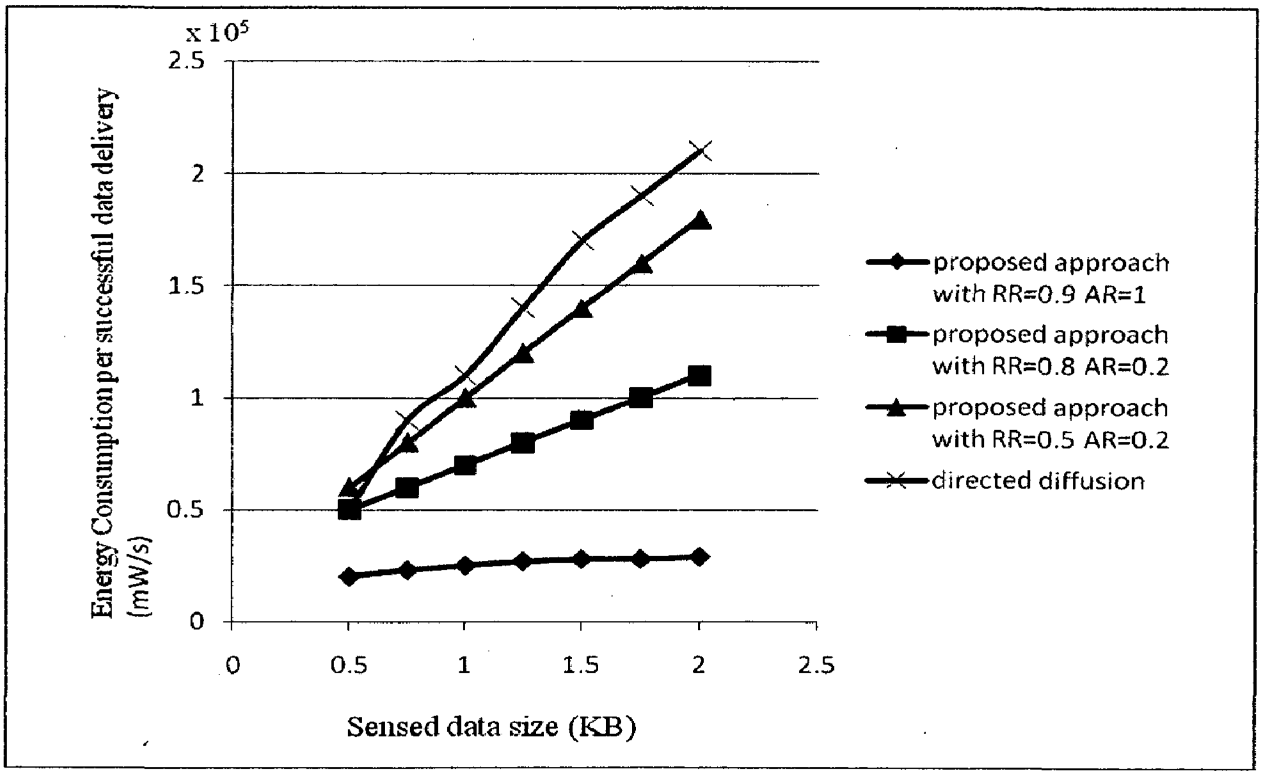


Figure 5.4 Effect of sensed data size on energy consumption

5.5 Effect on average end-to-end delay by changing size of sensed data

As shown in Figure 5.5, average end-to-end packet delay (T_{ma}) exhibits similar trend as energy consumption of our approach. Average end-to-end packet delay (T_{ma}) is more sensitive to the size of sensed data of each sensor (D_{data}) than energy consumption. When α (RR) is less than 0.6 and μ (RR) is equal to 0.2, our approach tends to have larger end-to-end packet delay than that of directed diffusion.

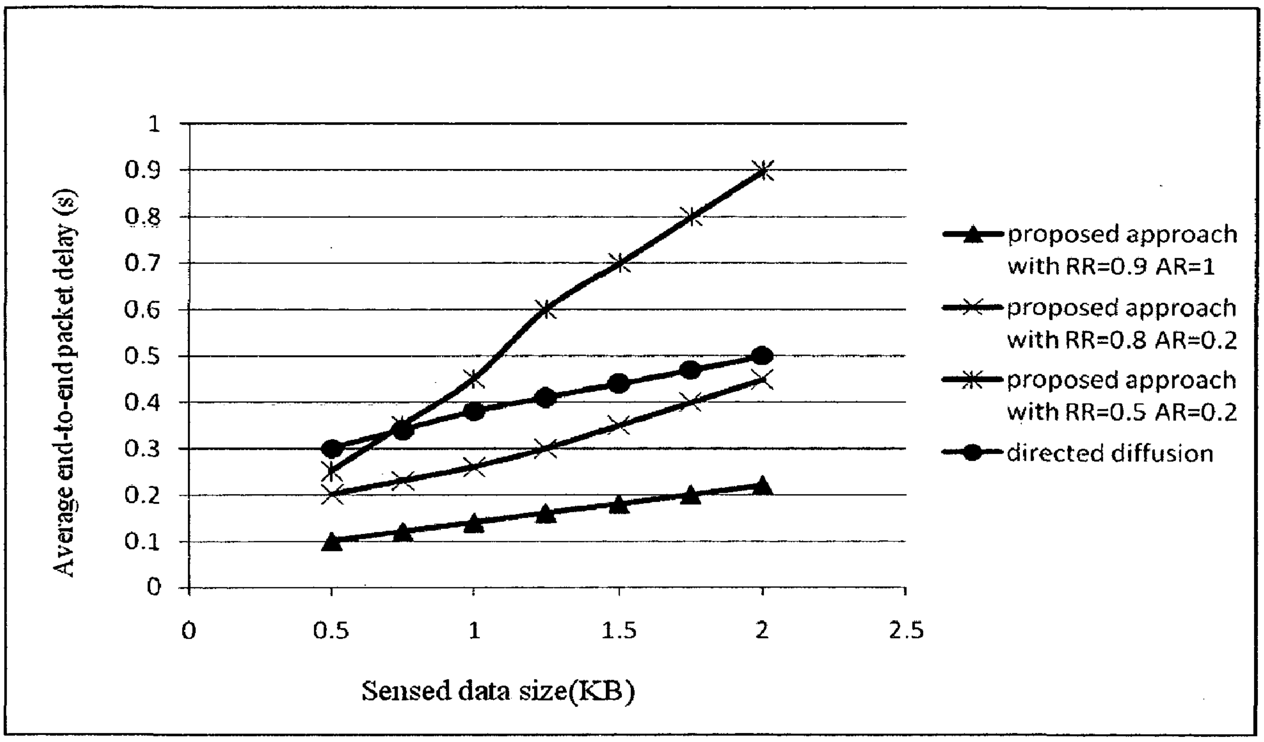


Figure 5.5 Effect of sensed data size on average end-to-end delay

5.6 Effect on energy, delay and reliability by changing size of sensed data

Figure 5.6 is result of α in terms of (energy * delay / reliability) η obtained after fixing μ. Here, we observe that our approach does not show advantage if the value of α is less than 0.4 and μ is equal to 0.2. The value of reduction ratio and aggregation ratio is dependent on type of application. Therefore, before adopting proposed approach for data collection, the features of the application should be investigated.

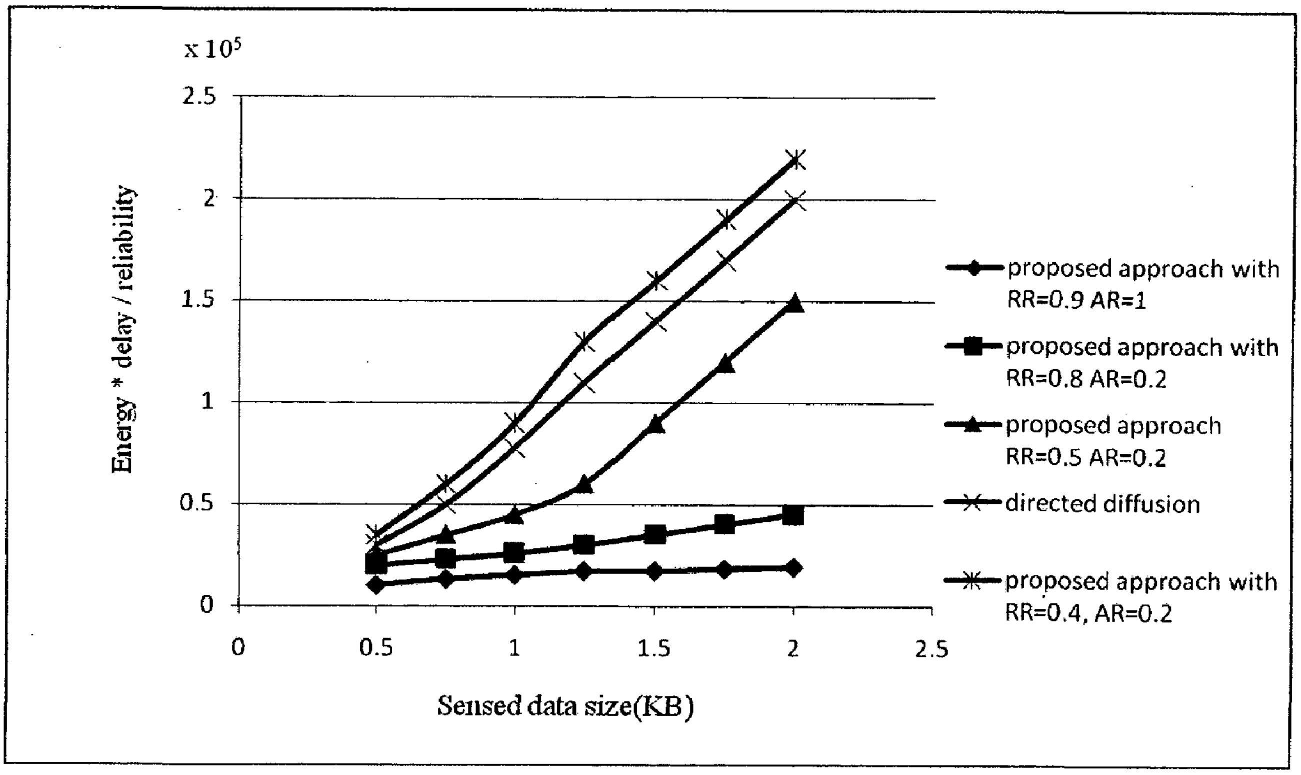


Figure 5.6 Effect of sensed data size on energy*delay/ reliability

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

6.1Conclusions

In this thesis we proposed an energy efficient approach for data collection in wireless sensor network. We have applied mobile agent based approach for efficient and reliable data collection. Our objectives were to design and implement mobile agent migration plan among sink nodes and source nodes to perform data collection such that energy consumption of WSN nodes decreases.

We designed and implemented gradient based route planning for mobile agent for data collection in WSN. We investigated efficiency of our approach by comparing it with another data collection method for WSN, directed diffusion. We have evaluated our approach based on performance metrics defined in chapter 5. Results of comparison are shown and discussed in chapter 5.

We can conclude from results shown in Figure 5.1 that our approach performs better for applications of WSN where source nodes process enough long streams of data. From Figure 5.2 we can say that if mobile agent access delay increase beyond some value along with large reduction ratio and small aggregation ration than average end-to-end packet delay is more compared with directed diffusion. Based on Figure 5.3 we can conclude that our approach achieves higher reliability than directed diffusion. Figure 5.4 and Figure 5.5 shows that our approach has lower energy consumption than that of directed diffusion.

Thus, in applications where energy consumption is of primary concern, mobile agent based approach for data collection exhibits substantially longer network lifetime than directed diffusion.

6.2 Suggestions for Future Work

In our approach, we have assumed that source nodes are deployed near event and are close to each other. Here, we send one mobile agent for data collection towards source nodes from sink node. Considering an application with more number of source nodes and distance between source nodes is also large, we can send more than one mobile agents from sink node for data collection. This will further reduce energy consumption of network nodes. This approach may be considered for future work.

REFERENCES

[1] Akyildiz, I., Su, W., Sankarasubramaniam, Y., and Cayirci, E,"A Survey on Sensor Networks", IEEE Communications Magazine, August 2002.

[2] Arampatzis,Th., Lygeros, J., Manesis,S., "A Survey of Applications of Wireless Sensors And Wireless Sensor Networks", Proceedings of the 13th Mediterranean Conference on Control and Automation Limassol, Cyprus, June 27-29, 2005.

[3] Lcpc geotechnical centrifuge the rion-antirion bridge. [Online]. Available:http://www.lcpc.fr/en/presentation/moyens/centrifugeuse/index1.dml (2008, June)

[4] Fire information and rescue equipment. [Online]. Available: http://fire.me.berkeley.edu (June 2008)

[5] G. Boone, "Reality mining: Browsing reality with sensor networks," Sensors Online, vol. 21, no. 9, Sept. 2004. Available: http://archives.sensorsmag.com/articles/0904/14/ .(June 2008)

[6] Deb, B., Bhatnagar, S., and Nath, B., "Reinform: Reliable information forwarding using multiple paths in sensor networks," in IEEE International Conference on Local Computer Networks (LCN'03), 2003.

[7] Akyildiz, I.F., and Stuntebeck, E, "Underground wireless sensor networks: research Challenges", Ad Hoc Networks (Elsevier), in press, June 2006

[8] Akyildiz, I.F., Pompili, D., Melodia, T., "'Underwater Acoustic Sensor Networks: Research Challenges," Ad Hoc Networks Journal, (Elsevier), March 2005

[9] Available Online: http://aisrp.nasa.gov/projects_nonav/be9f74be.html .(June 2008)

[10] NASA-AISRP Year 2 Annual Report Document [Online] Available: http://aisrp.nasa.gov/projects_nonav/reports/10dc6797c15.pdf (June 2008)

[11] Crossbow mote datasheet [Online] Available: http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/WSN_PRO_Series_Datasheet.pdf (June, 2008)

[12] Intel Mote [Online] Available: www.intel.com/research/exploratory/motes.htm (June, 2008)

[13] Minar, N., Hultman Kramer, K., and Maes, P., "Cooperating Mobile Agents for

Dynamic Network Routing", Proceedings of the 1st Hungarian National Conference on Agent Based Computation, 1999.

[14] Jain, R., Anjum, F., and Umar, A., "A comparison of mobile agent and Client-Server paradigms for information retrieval tasks in virtual enterprises", AiWoRC Workshop, Buffalo, New York (April), 2000.

[15] Mahmoud, Q., "MobiAgent: A mobile agent-based approach to wireless information systems." In Proceedings of the 3rd International Bi-Conference Workshop on Agent Oriented Information Systems (AOIS-2001), Montreal, 2001.

[16] Lange, D., and Oshima, M., "Mobile agents with Java: The Aglet API.", World Wide Web, vol. 1, no. 3:n.pag., 1998.

[17] Deb, B., Bhatnagar, S., and Nath, B., "Reinform: Reliable information forwarding using multiple paths in sensor networks", in IEEE International Conference on Local Computer Networks (LCN'03), 2003.

[18] Ganesan, D., Govindan, R., Shenker, S., and Estrin, D., "Highly-resilient, energy-efficient multipath routing in wireless sensor networks", ACM Mobile Computing and Communications Review, 2003.

[19] Madden, S., Franklin, M. J., Hellerstein, J. M. and W. Hong, "Tag: a tiny aggregation service for ad-hoc sensor networks," in USENIX OSDI, 2002.

[20] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "The design of an acquisitional query processor for sensor networks," in SIGMOD, 2003.

[21] H. Qi, S. Iyengar, and K. Chakrabarty, "Multi-resolution data integration using mobile agents in distributed sensor networks", IEEE Trans. Syst., Man, Cybern. C, vol. 31, no. 3, pp. 383-391, Aug. 2001.

[22] Q. Wu, N. S. V. Rao, J. Barhen, S. S. Iyengar, V. K. Vaishnavi, H. Qi, and K. Chakrabarty, "On computing mobile agent routes for data fusion in distributed sensor networks," IEEE Trans. Knowledge Data Eng., vol. 16, no. 6, June, 2004.

[23] Q. Li, M. D. Rosa, and D. Rus, "Distributed algorithms for guiding navigation across a Sensor network," in ACM MobiCom, 2003.

[24] P. Levis and D. Culler, "Mate: A tiny virtual machine for sensor networks," in ASPLOSX, 2002.

[25] H. Qi, Y. Xu, and X. Wang, "Mobile agent based collaborative signal and information processing in sensor networks," Proceedings of the IEEE, vol. 91, no. 8, pp.1172-1183, 2003.

[26] Network Simulator 2 [Online]
Available: http://www.isi.edu/nsnam/ns (June 2008)

[27] NRL Extension to NS2 [Online]
Available: http://cs.itd.nrl.navy.mil/products/ (June 2008)

[28] L. Feeney and M. Nilsson; "Investigating the energy consumption of a wireless network interface in an ad hoc networking environment," INFOCOM 2001.

[29] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, F. Silva; "Directed diffusion for wireless sensor networking," IEEE/ACM Transactions on Networking, vol.11, no.1, pp. 2-16, Feb 2003

Appendix: Source Code Listing

MAgent.h

```
#ifndef MAgent_h
#define MAgent_h

#include <stdio.h>
#include <g++-3/list.h>
#include <g++-3/map.h>
#include <g++-3/multimap.h>

#include "config.h"
#include "scheduler.h"
#include "timer-handler.h"
#include "Context.h"

#define DEFAULT_COLOR  "blue"

enum agent_state {MOBILE, ACTIVE,  DEACTIVATING, IDLE, TERMINATING,
DISPATCHING,ARRIVAL, REVERTING, CLONE, CLONED, CLONING};


class MobilityListener;
class PersistencyListener;

class MAgent : public TclObject, public TimerHandler {

public:
  MAgent();
  MAgent(const MAgent* agent);
  MAgent(Context* context);
  ~MAgent();
  void create_bindings();
  void initialize();

  int command(int argc, const char*const* argv);
  virtual void run(Context*){};
  virtual void run(const char*){};
  virtual void onCreation(){};
  virtual void onDisposing(){};
  int  getId() { return id_; };
  double getAge();
  void dispatch();
  void dispatch(const char* context);

  void deactivate(long duration);
  void dispose();
  bool is_active();
    void addMobilityListener(MobilityListener* listener);
  void addPersistencyListener(PersistencyListener* listener);

  void removeMobilityListener();
  void removePersistencyListener();

  Context*  getContext(){ return context_;};
  int get_size() {
    size_ =code_size_+ data_size_+ status_size_+ (int)((1-
selectivity_)*rep_size_);
    return size_;
  }
  void startCreatedAgent(Context*);
  void startArrivedAgent(Context*);
  void startRetractedAgent(Context*);
  virtual void expire(Event*);
  void sendMessage(const char*, int size);



private:
  MobilityListener* m_listener_;
  PersistencyListener* p_listener_;
  void configure(Context*);
protected:
  inline double now() {
    return Scheduler::instance().clock();
  }

  int id_;
  int size_;
  int code_size_;
  int data_size_;
  int status_size_;
  int rep_size_;
  double m_factor_;
  double selectivity_;
  double process_delay_;
  double launch_delay_;
  agent_state state_;
  char* lastnode_;
  char* homenode_;
  list<char*>* tablelist_;
  list<char*>* visitlist_;
  Context* context_;
  char* dst_context_;
  char* node_;
  static int idCount;
};

#endif
```

```
MAgent.cc

#include <stdio.h>
#include <unistd.h>
#include <iostream.h>

#include "tcl.h"
#include "MAgent.h"
#include "config.h"
#include "MobilityListener.h"
#include "PersistencyListener.h"



int MAgent::idCount=0;

MAgent::MAgent()  : TimerHandler()
{
  create_bindings();
  context_ = NULL;
  initialize();
}

MAgent::MAgent(Context* context)  : TimerHandler()
{
  create_bindings();
  context_ = context;
  initialize();
}


void MAgent::create_bindings()
{
  bind("id_", &id_);
  bind("size_", &size_);
  bind("code_size_",&code_size_);
  bind("data_size_",&data_size_);
  bind("status_size_",&status_size_);
  bind("rep_size_",&rep_size_);
  bind("selectivity_",&selectivity_);
  bind("m_factor_", &m_factor_);
  bind("process_delay_", &process_delay_);
  bind("launch_delay_", &launch_delay_);
}

void MAgent::initialize()
{
  id_ = idCount++;
  node_ = NULL;
  homenode_ = NULL;
  dst_context_ = NULL;
  lastnode_ = NULL;
  state_ = IDLE;
  tablelist_ = NULL;
```

```
    visitlist_ = NULL;
    m_listener_ = NULL;
    p_listener_ = NULL;
}

// COPY Constructor for cloning an agent
MAgent::MAgent(const MAgent* agent) : TimerHandler()
{
    id_ = idCount++;
    state_ = agent->state_;
    size_ = agent->size_;
    tablelist_ = agent->tablelist_;
    visitlist_ = agent->visitlist_;
    context_ = agent->context_;
    node_= agent->node_;
    m_listener_ = agent->m_listener_;
    p_listener_ = agent->p_listener_;
    homenode_, agent->homenode_;
    lastnode_, agent->lastnode_;
    status_ = agent->status_;
}

MAgent::~MAgent()
{

    if(tablelist_) delete tablelist_;
    if(visitlist_) delete visitlist_;
}

int MAgent::command(int argc, char const *const *argv)
{
    Tcl& tcl = Tcl::instance();
    if(argc==2) {
       if(strcmp(argv[1], "context")==0) {
          tcl.resultf("%s", getContext()->name());
          return TCL_OK;
       }
       if(strcmp(argv[1], "node")==0) {
          tcl.result(node_);
          return TCL_OK;
       }
       if(strcmp(argv[1], "is_active")==0) {
          tcl.resultf("%d", is_active());
          return TCL_OK;
       }
       if(strcmp(argv[1], "dispose")==0) {
          dispose();
          return TCL_OK;
       }
       if(strcmp(argv[1], "dispatch")==0) {
          dispatch();
          return TCL_OK;
       }
      if (strcmp(argv[1], "removeMobilityListener") == 0) {
        removeMobilityListener();
        return (TCL_OK);
      }
      if (strcmp(argv[1], "removePersistencyListener") == 0) {
        removePersistencyListener();
        return (TCL_OK);
      }
    }

    if (argc >= 3) {
      if (strcmp(argv[1], "dispatch") == 0) {
        dst_context_ = (char*) argv[2];
        dispatch(argv[2] );
        return (TCL_OK);
      }
       if(strcmp(argv[1], "resched")==0){
         resched(atof(argv[2]));
         return TCL_OK;
       }
      if (strcmp(argv[1], "deactivate") == 0) {
        deactivate((long) atoi(argv[2]) );
        return (TCL_OK);
      }
      if (strcmp(argv[1], "addMListener") == 0) {
        addMobilityListener((MobilityAdapter*)TclObject::lookup(argv[2]));
        return (TCL_OK);
      }
      if (strcmp(argv[1], "addPListener") == 0) {

addPersistencyListener((PersistencyListener*)TclObject::lookup(argv[2]));
        return (TCL_OK);
      }

      if (strcmp(argv[1], "sendMessage") == 0) {
        sendMessage(argv[2], atoi(argv[3]));
        return (TCL_OK);
      }
    }

    return (TclObject::command(argc, argv));
}


bool MAgent::is_active()
{
    return(state_== ACTIVE);
}

void MAgent::dispose()
{
        state_= TERMINATING;
        onDisposing();
        context_->disposeAgent(id_);
        delete this;
```

```
}


void MAgent::startCreatedAgent(Context* context)
{


  context_ = context;
  node_= context->node();

  homenode_=context->node();

  tablelist_ = new list<char*>();
  tablelist_->push_back(node_);
  size_ = get_size();
  onCreation();

}


void MAgent::startArrivedAgent(Context* context)
{
 // May do initializations here...
  if(m_listener_) {
     m_listener_->state_= ARRIVAL;
     Event e;
     m_listener_->expire(&e);

  }

  configure(context);
  if(process_delay_==0)
      process_delay_= 0.000005 * code_size_;
  resched(process_delay_);
  cout << "Process Delay "<<process_delay_<<endl;
}


void MAgent::startRetractedAgent(Context* context)
{
 // May do initializations here...
  if((state_==MOBILE )&&(m_listener_)) {
    m_listener_->state_= REVERTING;
    m_listener_->resched(0);
  }
  configure(context);
  resched(process_delay_);
}


void MAgent::configure(Context* context)
{
  state_ = ACTIVE;
```

```
   size_ = get_size();

   context_ = context;

   node_= context->node();
   tablelist_->push_back(node_);
 }


 void MAgent::addMobilityListener(MobilityListener* listener)
 {
  if(listener)
     m_listener_ = listener;
  else
     printf("MAgent::addMobilityListener: Cant' add listener nil");
 }


 void MAgent::addPersistencyListener(PersistencyListener* listener)
 {
   if(listener)
     p_listener_ = listener;
  else
     printf("MAgent::addPersistencyListener: Cant' add listener nil");
 }


 void MAgent::removeMobilityListener()
 {
   delete m_listener_;
   m_listener_ = NULL;
 }

 void MAgent::removePersistencyListener()
 {
   delete p_listener_;
   p_listener_ = NULL;
 }


 void MAgent::dispatch()
 {
   if(!visitlist_ || visitlist_->empty()) {
      dispatch(NULL);
       return;
   } else {
      char* context_nm =  visitlist_->front();
      visitlist_->pop_front();
      dispatch(context_nm);
     }
 }


 void MAgent::dispatch(const char* context_nm)
```

```
{
  lastnode_= node_;
  state_ = MOBILE;
  if(m_listener_) {
    m_listener_->state_ = DISPATCHING;
    m_listener_->resched(0);
  }
  if(launch_delay_ == 0)
    launch_delay_ = 2*m_factor_*get_size();
  cout << "Launch Delay "<<launch_delay_<<endl;
     cout << "DST context "<<dst_context_ <<endl;
  resched(launch_delay_);
}

void MAgent::deactivate(long duration)
{
  state_= DEACTIVATING;
  if(p_listener_) {
    p_listener_->state_ = DEACTIVATING;
    p_listener_->resched(0);
  }

   resched(duration);

}

void  MAgent::sendMessage(const char* agent_nm, int size)
{
  MAgent* agent = (MAgent*) TclObject::lookup(agent_nm);
  Tcl& tcl = Tcl::instance();
 tcl.evalf( "%s connect-to %s ", context_->name(), (agent->getContext())-
>name());
  tcl.evalf( "%s send %d \"%s handleMessage %s %d\" ", context_->name(),
size, agent_nm, name(), size );
}

void MAgent::expire(Event* e)
  // Timeout handler
{
   switch(state_)
  {
     case MOBILE:
    cout << "DST context "<<dst_context_ <<endl;
                if(dst_context_)
                    context_->move(id_,dst_context_,get_size());
                else
                    context_->move(id_, get_size());
                break;
     case ACTIVE:
                run(context_);
                break;
     case IDLE:
                state_ = ACTIVE;
                run(context_);
                break;
     case DEACTIVATING:
                  // as the agent wake's up now..
                  state_= ACTIVE;
                  if(p_listener_){
                    p_listener_->state_ = ACTIVE;
                    p_listener_->resched(0);
                  }
     default:
                printf("\nMAgent::Illegal mobile agent's state\n");

  }
}
```

Context.h

```
#ifndef Context_h
#define Context_h

#include <map.h>
#include <g++-3/list.h>
#include <g++-3/vector.h>
#include <g++-3/map.h>
#include "tclcl.h"
#include "ns-process.h"
#include "webcache/tcpapp.h"

class MAgent;
class MobileAgentData;

class Context: public TcpApp {
public:
  Context(Agent* recv_tcp, Agent* send_tcp, const char* );
  ~Context();
  int command(int argc, const char*const* argv);
  void recv(Packet* p, Handler* h);
  void process_data(int size, AppData* data);

  void retractAgent(const char*);

  void registerAgent(MAgent*, int, int);


  void disposeAgent(int);
  void startAgent(const char*);
  const char* getAgent(int );
  void move(int id, int size);
  void move(int id,const char*, int size);
  void registerAgent(MAgent*, int);
  char* node() { return node_;}
private:
  int init();
  int disable();
  inline double now() {
    return Scheduler::instance().clock();
  }
  char* node_;
  map<int, MobileAgentData*>  agentList_;
  bool enable_;
};


class MobileAgentData : public AppData {

private:
      char* reference_;
public:
      MobileAgentData() : AppData(MOBILE_AGENT)
```

```
        {
              reference_ =NULL;
        }
        MobileAgentData(const char* ref )  : AppData(MOBILE_AGENT)
        {
                    reference_ = new char[strlen(ref)+1];
              strcpy(reference_, ref);
        }
        MobileAgentData(MobileAgentData& d)  : AppData(d)
        {
                    reference_ = new char[strlen(d.reference_)+1];
                    strcpy(reference_, d.reference_);
        }
        ~MobileAgentData() {
              if(reference_)
                    delete []reference_;
        }
        char* str() {return reference_;}
        void set_agent(const char* s)
        {
              if(s) {
                    reference_ = new char[strlen(s)+1];
                    strcpy(reference_, s);
                    }
        }

        // abstract methods of AppData required to be over-ridden
        virtual int size() const {return sizeof(MobileAgentData); }
        AppData* copy() { return (new MobileAgentData(*this)); }

};


#endif
```

Context.cc

```
#include <stdio.h>
#include <iostream.h>
#include <math.h>
#include <values.h>

#include "ip.h"
#include "tclcl.h"
#include "Context.h"
#include "MAgent.h"


#define DEBUG(a) ;
#define FLOAT_PRECISION  10



static class ContextClass : public TclClass {
public:
  ContextClass() : TclClass("Application/TcpApp/Context") {
  }
  TclObject* create(int argc, const char*const* argv) {
        if(argc!=7)
             return NULL;
        Agent *recv_tcp = (Agent*) TclObject::lookup(argv[4]);
        if(recv_tcp ==NULL)
             return NULL;
        Agent *send_tcp = (Agent*) TclObject::lookup(argv[5]);
        if(send_tcp ==NULL)
             return NULL;
        return (new Context(recv_tcp,send_tcp, argv[6]));
  }
}class_context;


Context::Context(Agent* recv_tcp, Agent* send_tcp, const char* node)
  : TcpApp(recv_tcp, send_tcp)
{
  enable_ = false;
  node_ = new char[strlen(node)+1];
  strcpy(node_,node);
}

Context::~Context()
{
}

int Context::command(int argc, const char*const* argv)
{
  Tcl& tcl = Tcl::instance();
  if(strcmp(argv[1], "start") == 0) {
    init();
    return TCL_OK;
    }
    if(strcmp(argv[1], "shutdown") == 0) {
        disable();
        return TCL_OK;
    }
    if(strcmp(argv[1], "node") == 0) {
        tcl.result(node_);
        return TCL_OK;
    }
    if(strcmp(argv[1], "retractAgent") == 0) {
        retractAgent(argv[2]);
        return TCL_OK;
    }
    if(strcmp(argv[1], "getAgent") == 0) {
        const char* agent= getAgent(atoi(argv[2]));
        if(agent) {
        tcl.result(agent);
          return TCL_OK;
        }
        else
         return TCL_ERROR;
    }
    if(strcmp(argv[1], "startAgent") == 0) {      // argv[2] -> agent name
        startAgent(argv[2]);
        return TCL_OK;
      }
    return TcpApp::command(argc,argv);
}

void Context::process_data(int size, AppData* data)
{
    if(enable_)
    {
        if(data->type()== MOBILE_AGENT)
        {
            MobileAgentData* agent_data = (MobileAgentData*) data;
            char* agent_ref= NULL;
            MAgent* agent= NULL;
            if(agent_data) {
                agent_ref = agent_data->str();
                if (agent_ref != NULL) {
                  agent = (MAgent*)TclObject::lookup(agent_ref);
              }
                  else {
                printf("Context.cc:recv():AGLET with no agent reference
received.");
              return;
           }
        }
        else {
           printf("Context:AppData recevied from agent is NULL\n");
           return;
        }
        if(agent != NULL)
      {
          int agentId = agent->getId();
          agent_data = new MobileAgentData((char*)agent_ref);
          agentList_[agentId] = agent_data;

          agent->startArrivedAgent(this);

      }
      else {
          Tcl& tcl = Tcl::instance();
          tcl.evalf("%s get_id ", agent_ref);
          int agentId = atoi(tcl.result());
          agent_data = new MobileAgentData(agent_ref);
          agentList_[agentId] = agent_data;
          tcl.evalf("%s run %s", agent_ref, name());
      }

    }
    else if(data->type()== TCPAPP_STRING) {
      TcpAppString *tmp = (TcpAppString*)data;
      Tcl::instance().eval(tmp->str());
    }

  } else
  printf("\nContext you are trying to reach is not enabled.");
}

int Context::init()
{
  enable_ = true;
  return TCL_OK;
}

int Context::disable()
{
   enable_ = false;
   return TCL_OK;
}

void Context::move(int agentId, int size ) {

  MobileAgentData* data = (MobileAgentData*)agentList_[agentId];
  agentList_.erase(agentId);
  send(size, data);
}


void Context::move(int agentId,const char* context_nm, int size ) {

  MobileAgentData* data = agentList_[agentId];
  agentList_.erase(agentId);
  Tcl& tcl = Tcl::instance();
  tcl.evalf("%s connect-to %s", name(), context_nm);
  send(size, data);
```

```
}

void Context::startAgent(const char*  agent_nm)
{
  MAgent* agent= (MAgent*) TclObject::lookup(agent_nm);
  if(agent)
  {

    int agentId = agent->getId();

    MobileAgentData* data = new MobileAgentData(agent_nm);

    agentList_[agentId] = data;

    agent->startCreatedAgent(this);
    agent->run(this);
  }
  else {
    printf("\n Context: startAgent(): Agent with name %s not found in the
TclObject Table", agent_nm);
  }
}

void Context::retractAgent(const char*  rem_agent)
{
  MAgent* agent= (MAgent*) TclObject::lookup(rem_agent);
  if(agent!=NULL) {
    agent->dispatch(name());
  }
  else
     printf("\nAgentContext::retractAgent(): mentioned agent not found");
}
void Context::registerAgent(MAgent* agent, int agentId, int size)
{
      MobileAgentData* data = new MobileAgentData(agent->name());
      agentList_[agentId] = data;
}

void Context::disposeAgent(int agentId) {

  unsigned int size= agentList_.erase(agentId);
}

const char* Context::getAgent(int agentId) {
  MobileAgentData* data  = agentList_[agentId];
  const char* agent_ref = data->str();
  if (agent_ref != NULL) {
    return agent_ref;
  }
  else {
    printf("AgentContext.cc: getAgent(): No agent found in the acket\n");
    return NULL;
  }
}
```

Context.cc

```
#include <stdio.h>
#include <iostream.h>
#include <math.h>
#include <values.h>

#include "ip.h"
#include "tclcl.h"
#include "Context.h"
#include "MAgent.h"


#define DEBUG(a) ;
#define FLOAT_PRECISION  10



static class ContextClass : public TclClass {
public:
  ContextClass() : TclClass("Application/TcpApp/Context") {
  }
  TclObject* create(int argc, const char*const* argv) {
        if(argc!=7)
              return NULL;
        Agent *recv_tcp = (Agent*) TclObject::lookup(argv[4]);
        if(recv_tcp ==NULL)
              return NULL;
        Agent *send_tcp = (Agent*) TclObject::lookup(argv[5]);
        if(send_tcp ==NULL)
              return NULL;
        return (new Context(recv_tcp,send_tcp, argv[6]));
  }
}class_context;


Context::Context(Agent* recv_tcp, Agent* send_tcp, const char* node)
  : TcpApp(recv_tcp, send_tcp)
{
  enable_ = false;
  node_ = new char[strlen(node)+1];
  strcpy(node_,node);
}

Context::~Context()
{
}

int Context::command(int argc, const char*const* argv)
{
  Tcl& tcl = Tcl::instance();
  if(strcmp(argv[1], "start") == 0) {
    init();
    return TCL_OK;
```

```
   }
   if(strcmp(argv[1], "shutdown") == 0) {
       disable();
       return TCL_OK;
   }
   if(strcmp(argv[1], "node") == 0) {
       tcl.result(node_);
       return TCL_OK;
   }
   if(strcmp(argv[1], "retractAgent") == 0) {
       retractAgent(argv[2]);
       return TCL_OK;
   }
   if(strcmp(argv[1], "getAgent") == 0) {
       const char* agent= getAgent(atoi(argv[2]));
       if(agent) {
       tcl.result(agent);
         return TCL_OK;
       }
       else
        return TCL_ERROR;
   }
   if(strcmp(argv[1], "startAgent") == 0) {      // argv[2] -> agent name
       startAgent(argv[2]);
       return TCL_OK;
     }
   return TcpApp::command(argc,argv);
}

void Context::process_data(int size, AppData* data)
{
  if(enable_)
  {
      if(data->type()== MOBILE_AGENT)
      {
          MobileAgentData* agent_data = (MobileAgentData*) data;
          char* agent_ref= NULL;
          MAgent* agent= NULL;
          if(agent_data) {
              agent_ref = agent_data->str();
              if (agent_ref != NULL) {
                agent = (MAgent*)TclObject::lookup(agent_ref);
            }
                else {
              printf("Context.cc:recv():AGLET with no agent reference
received.");
            return;
         }
      }
      else {
         printf("Context:AppData recevied from agent is NULL\n");
         return;
      }
      if(agent != NULL)
      {
          int agentId = agent->getId();
          agent_data = new MobileAgentData((char*)agent_ref);
          agentList_[agentId] = agent_data;

          agent->startArrivedAgent(this);

      }
      else {
          Tcl& tcl = Tcl::instance();
          tcl.evalf("%s get_id ", agent_ref);
          int agentId = atoi(tcl.result());
          agent_data = new MobileAgentData(agent_ref);
          agentList_[agentId] = agent_data;
          tcl.evalf("%s run %s", agent_ref, name());
      }

    }
    else if(data->type()== TCPAPP_STRING) {
      TcpAppString *tmp = (TcpAppString*)data;
      Tcl::instance().eval(tmp->str());
    }

  } else
  printf("\nContext you are trying to reach is not enabled.");
}

int Context::init()
{
  enable_ = true;
  return TCL_OK;
}

int Context::disable()
{
   enable_ = false;
   return TCL_OK;
}

void Context::move(int agentId, int size ) {

  MobileAgentData* data = (MobileAgentData*)agentList_[agentId];
  agentList_.erase(agentId);
  send(size, data);
}


void Context::move(int agentId,const char* context_nm, int size ) {

  MobileAgentData* data = agentList_[agentId];
  agentList_.erase(agentId);
  Tcl& tcl = Tcl::instance();
  tcl.evalf("%s connect-to %s", name(), context_nm);
  send(size, data);
```

```
}

void Context::startAgent(const char*  agent_nm)
{
  MAgent* agent= (MAgent*) TclObject::lookup(agent_nm);
  if(agent)
  {

    int agentId = agent->getId();

    MobileAgentData* data = new MobileAgentData(agent_nm);

    agentList_[agentId] = data;

    agent->startCreatedAgent(this);
    agent->run(this);
  }
  else {
    printf("\n Context: startAgent(): Agent with name %s not found in the
TclObject Table", agent_nm);
  }
}

void Context::retractAgent(const char*  rem_agent)
{
  MAgent* agent= (MAgent*) TclObject::lookup(rem_agent);
  if(agent!=NULL) {
    agent->dispatch(name());
  }
  else
     printf("\nAgentContext::retractAgent(): mentioned agent not found");
}
void Context::registerAgent(MAgent* agent, int agentId, int size)
{
      MobileAgentData* data = new MobileAgentData(agent->name());
      agentList_[agentId] = data;
}

void Context::disposeAgent(int agentId) {

  unsigned int size= agentList_.erase(agentId);
}

const char* Context::getAgent(int agentId) {
  MobileAgentData* data  = agentList_[agentId];
  const char* agent_ref = data->str();
  if (agent_ref != NULL) {
    return agent_ref;
  }
  else {
    printf("AgentContext.cc: getAgent(): No agent found in the acket\n");
    return NULL;
  }
}
```