



8-2003

Development of mobile agent framework in wireless sensor networks for multi-sensor collaborative processing

Phani Teja Kuruganti

Follow this and additional works at: https://trace.tennessee.edu/utk_gradthes

Recommended Citation

Kuruganti, Phani Teja, "Development of mobile agent framework in wireless sensor networks for multi-sensor collaborative processing. " Master's Thesis, University of Tennessee, 2003.
https://trace.tennessee.edu/utk_gradthes/5247

This Thesis is brought to you for free and open access by the Graduate School at TRACE: Tennessee Research and Creative Exchange. It has been accepted for inclusion in Masters Theses by an authorized administrator of TRACE: Tennessee Research and Creative Exchange. For more information, please contact trace@utk.edu.

To the Graduate Council:

I am submitting herewith a thesis written by Phani Teja Kuruganti entitled "Development of mobile agent framework in wireless sensor networks for multi-sensor collaborative processing." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

Hairong Qi, Major Professor

We have read this thesis and recommend its acceptance:

Accepted for the Council:

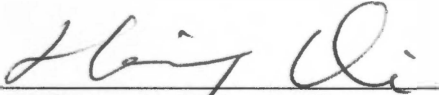
Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

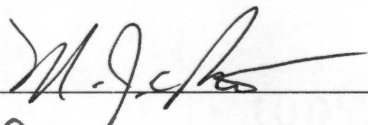
To the Graduate Council:

I am submitting herewith a thesis written by Phani Teja Kuruganti entitled "Development of Mobile Agent Framework in Wireless Sensor Networks for Multi-Sensor Collaborative Processing". I have examined the final paper copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.



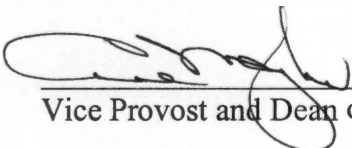
Hairong Qi, Major Professor

We have read this thesis
and recommend its acceptance:



Daniel B. Kodh

Accepted for the Council:



Vice Provost and Dean of Graduate Studies

Thesis
2003
.K97

Development of Mobile Agent Framework in Wireless Sensor Networks for Multi-Sensor Collaborative Processing

A Thesis
Presented for the
Master of Science Degree
The University of Tennessee, Knoxville

Phani Teja Kuruganti

August, 2003

Acknowledgment

Foremost, I would like to express my deepest gratitude to my advisor, Professor Hairong Qi, for her excellent guidance, infinite patience and endless support during my graduate study and research at The University of Tennessee, Knoxville.

I also wish to give my thanks to Professor Michael J. Roberts and Professor Daniel B. Koch for serving on my thesis committee.

I'm deeply indebted to my parents and relatives for the support and the encouragement they provided to me to explore higher levels of education.

I would like to give my special thanks to my sister, Srilalitha, who gave me the strongest support during my study life.

I want to thank people from BBN, Sensoria, BAE, Auburn and ISI who helped during field demos and provided developmental support to make this thesis possible.

Finally, I want to express my deepest gratitude to my friends for the support they gave me during my graduate study at UT.

Abstract

Recent advances in processor, memory and radio technology have enabled production of tiny, low-power, low-cost sensor nodes capable of sensing, communication and computation. Although a single node is resource constrained with limited power, limited computation and limited communication bandwidth, these nodes deployed in large number form a new type of network called the *wireless sensor network* (WSN). One of the challenges brought by WSNs is an efficient computing paradigm to support the distributed nature of the applications built on these networks considering the resource limitations of the sensor nodes.

Collaborative processing between multiple sensor nodes is essential to generate fault-tolerant, reliable information from the densely-spatial sensing phenomenon. The typical model used in distributed computing is the client/server model. However, this computing model is not appropriate in the context of sensor networks. This thesis develops an energy-efficient, scalable and real-time computing model for collaborative processing in sensor networks called the mobile agent computing paradigm. In this paradigm, instead of each sensor node sending data or result to a central server which is typical in the client/server model, the information processing code is moved to the nodes using mobile agents. These agents carry the execution code and migrate from one node to another integrating result at each node. This thesis develops the mobile agent framework on top of an energy-efficient routing protocol called directed diffusion.

The mobile agent framework described has been mapped to collaborative target classification application. This application has been tested in three field demos conducted at Twentynine palms, CA; BAE Austin, TX; and BBN Waltham, MA.

Contents

1	Introduction	1
1.1	Wireless Sensor Networks	4
1.2	Sensor Nodes	6
1.3	Applications of Wireless Sensor Networks	10
1.3.1	Military Applications	11
1.3.2	Civilian Applications	11
1.3.3	Medical Application	11
1.4	Challenges in Wireless Sensor Networks	12
1.5	Computing Models in WSNs	14
1.5.1	The Client/Server Approach	15
1.5.2	The Mobile-Agent Based Approach	16
1.6	Routing Protocols in Wireless Sensor Networks	18
1.6.1	Flooding	19
1.6.2	SPIN	20
1.6.3	Directed Diffusion	21
1.7	Contribution of Research	22
1.8	Structure of the Thesis	22

2	Development of a Mobile Agent Framework on Directed Diffusion	24
2.1	Architectural Overview	25
2.2	Directed Diffusion API	26
2.3	Mobile Agent Framework	30
2.3.1	MAF - The Finite State Machine	30
2.3.2	Mobile Agent Creation	31
2.3.3	Mobile Agent Dispatch	32
2.3.4	Mobile Agent Migration	33
2.3.5	Local Processing	34
2.3.6	MAF - Implementation	34
2.4	Analysis of the Architecture	36
3	Experimental Demonstrations	42
3.1	System Overview	42
3.1.1	Sensor Node Platform	43
3.1.2	Sensoria RF Modems	47
3.1.3	BAE Low Level Signal Processing	49
3.1.4	UTK Target Classification	49
3.1.5	MAF based Collaborative Processing	50
3.2	SITEX02 Demo	51
3.3	BAE Austin Demo	56
3.4	BBN Waltham Demo	61
4	Conclusion and Future Work	65
	Bibliography	68

Appendix	74
A	75
A.1 Dependencies to Build MAF	75
A.2 Compilation of MAF	81
A.3 Running the MAF	82
Vita	85

List of Figures

1.1	Traditional layered architecture of a network [22].	2
1.2	Sensor node architecture.	6
1.3	UC Berkeley Mote.	8
1.4	PC-104 and Sensoria nodes.	9
1.5	The hardware and software architectures of WINS NG 2.0 node.	10
1.6	Different computing paradigms [34].	15
1.7	Conceptual model for mobile agent computing in WSN [21].	17
1.8	Mobile agent components [45].	18
2.1	Architectural overview of the system.	26
2.2	A simplified schematic of directed diffusion [27].	29
2.3	Illustration of different aspects of diffusion [27].	29
2.4	MAF - finite state machine.	30
2.5	Implementation of MAF	35
2.6	Flow chart of MAF server.	37
2.7	Flow chart of mobile-agent.	38

2.8	Screen shot of processor usage, the last three lines (PIDs 269, 270, 271) show the processor usage by the mobile agent daemon in idle state.	39
2.9	Screen shot of MAF server.	39
3.1	Overview of nodal architecture for collaborative classification.	43
3.2	Processor specification.	44
3.3	High speed analog sampling system.	45
3.4	Power specification.	45
3.5	High speed analog front-end specification.	45
3.6	GPS specification.	46
3.7	RF modem specification.	46
3.8	Mobile-agent-based multi-sensor fusion [42].	51
3.9	Sensor node in the field with all sensors.	53
3.10	Plan of the DARPA SensIT experimental demo [38].	53
3.11	Node distribution and radio configuration on north-South leg [38].	54
3.12	Nodes on east-west and center [38].	54
3.13	Vehicles used for classification at 29 Palms.	55
3.14	Node lay down at BAE Austin - T-Junction.	57
3.15	Node lay down at BAE Austin - Parking lot.	58
3.16	Node in weather proof box, BAE, Austin, TX.	58
3.17	Vehicles used for classification.	59
3.18	Node lay down at BBN, Waltham.	62
3.19	Radio configuration at BBN, Waltham.	63

Chapter 1

Introduction

Recent advances in the design of micro-electro-mechanical systems, wireless communications, and digital electronics have made it possible to produce tiny sensor nodes which integrate sensing, processing, and communication capabilities. These sensor nodes paved the way for a next generation of distributed networks called wireless sensor networks (WSN). A large number of these compact sensor nodes can be quickly deployed in the field, where each sensor independently senses the environment but collaboratively achieves complex information gathering and dissemination tasks like intrusion detection, target tracking [41, 42, 49], localization, environmental monitoring [14], health systems [26] remote sensing, and the like [33, 36]. Unique to these sensor networks is the ability to cover wide areas that no single sensor could possibly observe and to provide a dense spatial sampling with multiple aspect and sensing modalities [20]. Unlike traditional networks the sensor networks require energy efficient protocols and innovative communication techniques for efficient use of bandwidth because the sensor nodes are supplied with only a limited amount of

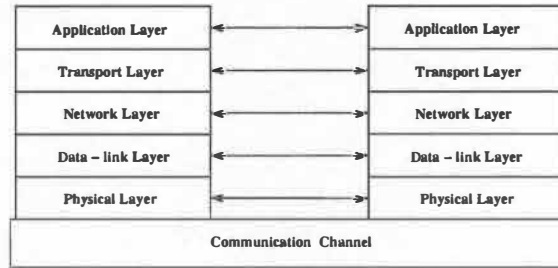


Figure 1.1: Traditional layered architecture of a network [22].

energy and computation power. Conventional networks are designed based on ISO OSI (Open Systems Interconnect) layered approach. Each layer of the system is designed separately and independent of the application. The protocols designed upon these layers can be used by different applications but are not optimal for a given application [22]. Instead of using general-purpose architectures, it is advantageous if the systems are built to exploit the nature of supporting applications. This can enhance the performance of system. This kind of cross-layer design is primarily used for building sensor networks.

The traditional networking topology as shown in Fig. 1.1 is a well defined architecture with protocols designed in each network layer capable of talking to the upward or downward layer in the network stack. This kind of layered networking approach has been popularized since the late 70s and 80s and was proved structurally advantageous especially due to the growing popularity of the Internet during the time. Over the years, more complex networking applications have been developed. In Boeing's Seattle engineering laboratories, a sheet of networked sensors covering a wing provides a profile of stress patterns as the structural integrity of the wing is being tested [44]. A network of sensors deployed in an unmanned terrain helps in remote surveillance of the terrain. All these kind of applications have edged

out from a traditional networking technology and move close toward the application requirements. This forces the designers to build networks governed more by the application layer. Sensor networks are the new generation networks which are tightly coupled with the physical phenomenon, with the functionality of the network depending heavily on the change in this phenomenon. The technology and architectures supporting this kind of network is currently a research topic and has a lot of potential to explore. The way the sensors collaborate and co-ordinate themselves to provide a unanimous decision on the sensed phenomenon is a challenge widely addressed in the current sensor network research community.

The development of the sensor networks is similar to that of packet-switched networks during their new development in the 60s and 70s. Few at the time could have predicted that that kind of basic technology would revolutionize the world-encompassing Internet [17]. As the OSI based layered network and the Internet grew stronger, more research went into development of efficient routing, processing and physical connection. Drawing parallel with the traditional network development, the sensor network is new and the research needs to turn, as it did at the corresponding time of the packet-switched networks, to developing the appropriate models, abstractions and methodologies that will make these systems built on a large scale, for a wide variety of uses, by necessarily a large collection of people [17].

In this chapter the sensor networks and how they are unique from the traditional networks are first described in Sec. 1.1, followed by a discussion on the architecture and existing sensor nodes in Sec 1.2. In Sec 1.3 the challenges provided by the sensor networks are discussed. Sec 1.4 focuses on different computing models in WSNs. Different routing protocols used in the context of the WSNs are discussed in Sec.

1.5. The rest of the chapter discusses applications, the contribution of this thesis and the structure of the thesis.

1.1 Wireless Sensor Networks

The emergence of sensor networks is actively supported by the sophistication in sensor development, enhancement in communication system design and the reduced feature size of the silicon fabrication. The growth of these networks has prompted their usage in plethora of environments. In today's scenario there are many places where the sensor networks are applied like automated factories, building surveillance, environmental monitoring, etc.

Compared to the traditionally structured 7-layer networks, sensor networks are unique in nature. Some of the unique features of sensor networks are listed below:

- Sensor nodes are deployed in hundreds and thousands and are randomly placed compared to an orderly placement and configuration of the traditional networks. This kind of random placement of the sensor nodes does not follow any fixed pattern and the density of nodes is not dependent on any factor.
- The nodes in the sensor network are not named by their IP or any other kind of addresses as in the traditional networks. Since there is no meaning for reading sensor data from a single sensor. It is unlikely for a sensor network application to ask a question like: What is the speed of a vehicle at sensor #23 or temperature at sensor #27, etc. Rather the applications focus on the *data* generated by the sensors. The queries on the network usually reflect like: In which direction and at what speed a certain vehicle is moving in the

sensor field? or Where are the nodes whose temperatures recently exceeded 40 degrees? This approach decouples data from the sensor. Data are named by attributes and applications request data matching certain attribute values [17]. This calls for a collaborative answer from the network, hence no conventional naming scheme is applied to sensor networks.

- The sensor nodes have limited battery and computational power as well as limited communication bandwidth. This requires optimization of the sensor networks at all the different levels of design including algorithms, operating system, hardware design, sensor design, MAC layer design, keeping all the limiting factors in view.
- Substantial amounts of sensor nodes could fail due to battery exhaustion, damaged node, accidental injury to sensors or nodes as well as environmental changes. Some nodes may be added to or removed from the network at any time. This indicates a need for an efficient co-ordination and collaboration among sensor nodes. On the other hand, fault-tolerance can be obtained by increasing the redundancy of sensing, However the increase in redundancy is in contrast to the limited available power and computation of nodes. Thus these networks call for a design that balances efficient usage of available power and fault-tolerance.

All of these unique features in sensor network design and the marvel of the applications they can support pave way for a futuristic research in *wireless sensor network*.

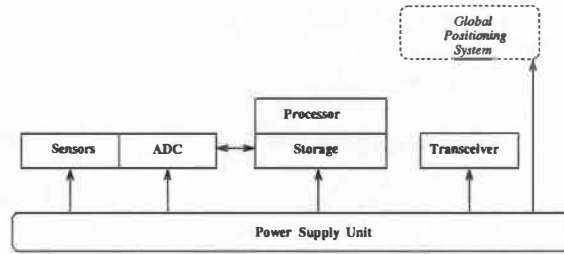


Figure 1.2: Sensor node architecture.

1.2 Sensor Nodes

A sensor node is a battery-operated device consisting of a sensing unit, processing unit, transceiver and storage memory. Typical architecture of the sensor node is shown in Fig 1.2.

The sensing unit consists of a sensor and an analog to digital converter (ADC) which converts all the real-time continuous signals to digital data and makes them available for further processing. The sensors are essentially transducers mostly converting the physical phenomenon into electrical signals. The number of sensors that a sensor node can take in depends on the capacity of the ADC and how many channels it can handle. Usually the sensor nodes provide the user with multiple sensing modalities.

The processing unit consists of a processor associated with some form of storage. The processor or sometimes just a micro controller unit (MCU), is responsible for executing signal processing and networking algorithms on the gathered sensor data. The processor capacity varies depending on the application for which the sensor nodes are deployed since the processor consumes considerable amount of the power of the battery. Nodes manufactured by Sensoria use Hitachi SH-4, a 167 MHz [1]

processor, which can handle complex algorithms but with high energy dissipation. The Smart dust mote from Berkeley uses an Atmel AVR 8535 at 4 MHz. The μ -AMPS sensor node from MIT contains SA-1110 processor running at 59 - 206 MHz. Although there are plenty of choices in choosing the processing units the application desired and the energy consumption issues determine what processor to use.

The transceiver unit helps the node to talk on the network and establish connection between the nodes. Typically sensor nodes have a radio frequency (RF) based transceivers. They can also be active or passive optical devices like what's used in the smart dust mote. The RF communication is preferred in WSN because the packets transmitted are small, at low data rate and the bandwidth usage is efficient due to short ranges of communication distances [25]. The RF communication involves modulation, band pass, filtering, demodulation and multiplexing circuitry.

Finally, the most important unit is the power unit which can be in the form of a conventional battery with finite charge or an alternate energy source like a solar cell. The power of the battery unit determines the life time of the sensor network. Usually a sensor network is deployed in inhospitable conditions where changing the battery of the node or recharging it is not an option.

The sensor nodes apart from the above described essential units, also occasionally utilize the global positioning system (GPS) to locate the position of the sensor node since most applications like target tracking and localization require it. The Sensoria nodes carry a GPS device on board with a resolution of 5m. Apart from the existing sensor node implementations, there is an industry-standard bus PC-104 to custom manufacture a sensor node using custom off the shelf (COTS) equipment.

Currently, the most popular choices of sensor nodes in building a sensor network

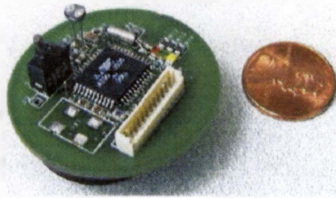
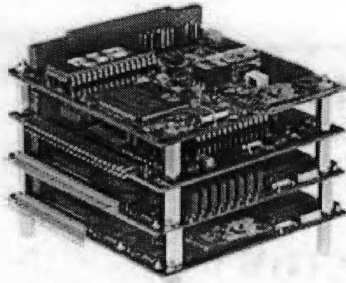


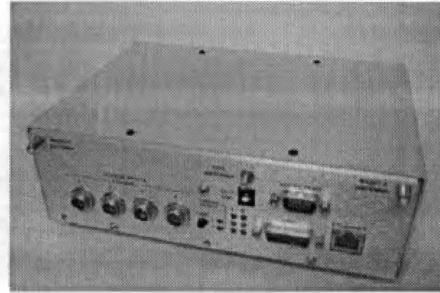
Figure 1.3: UC Berkeley Mote.

are:

- **UC Berkeley Smart Dust Motes:** These nodes are designed by University of California, Berkeley and are popularly called smart dust [24]. The mote is shown in Fig. 1.3. This is a tiny node containing an MCU (ATMEL 90LS8535) [3] with 8-bit Harvard architecture and 16-bit addresses. This controller provides 32 8-bit general purpose registers and runs at 4 MHz and 3.0 V. The system memory comprises 8KB flash memory and 512-byte SRAM as data memory. The radio is an asynchronous input/output device with hard real-time constraints. It consists of an RF Monolithics 916.50 MHz transceiver, antenna and collection of physical-layer components to configure the physical layer characteristics such as signal strength and sensitivity. It comes with a temperature sensor with an option to mount custom-selected sensors on the sensor board. The nodes run TinyOS [2] operating system which fits in 178 bytes of memory supporting two-level scheduling and allows for high concurrency to be handled in a very small amount of space [24].
- **PC-104 based Nodes:** PC-104 is an industry standard of PC-compatible modules that can be stacked together to form a custom-designed embedded system [5]. The term PC-104 is derived from the connector used to stack



(a) PC-104 based sensor node



(b) WINS NG 2.0 sensor node

Figure 1.4: PC-104 and Sensoria nodes.

different boards having 104 pins. The standard was initially released in 1992 [4]. Since these systems are made with hardware compatible with PC systems it is easy to configure them along with the PCs. The PC-104 sensor nodes are custom built with chosen processor, memory configuration and hard disk. The SCADDS testbed of USC/ISI consists of 30 nodes built using PC-104 based products [6]. Fig. 1.4 (a) shows a PC-104 based node.

- **Sensoria WINS NG nodes:** WINS NG node is a Linux based embedded computing platform with several interfaces to externally connect sensors, wireless extension cards and any serial port devices. Fig. 1.4 (b) shows a Sensoria sensor node. This is the node used to build the sensor network for this thesis work. This node uses the Hitachi SH-4 processor running at 167 MHz. The SH-4 is a 32-bit RISC with a 128-bit vector floating point unit (FPU) and super-scalar implementation providing higher speeds at low clock rates [9]. The sensor node supports four sensors and also hosts a GPS module for geo-location information of the nodes. The node communicates with the dual

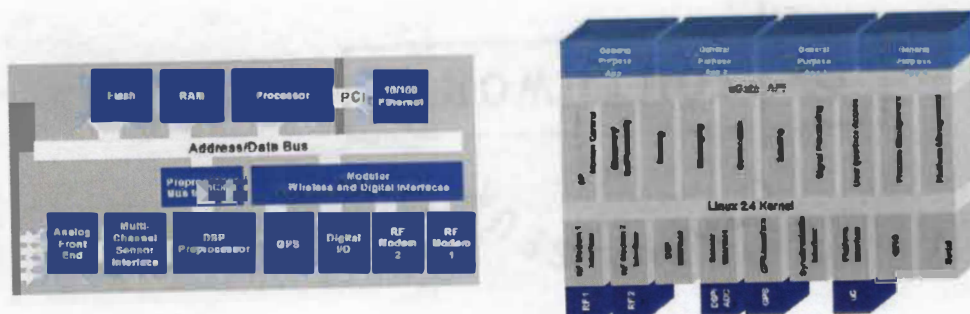


Figure 1.5: The hardware and software architectures of WINS NG 2.0 node.

RF modems built-in, both of them in the 2.401 - 2.495 GHz ISM band using frequency-hopping spread spectrum (FHSS). The hardware architecture of the node is seen in the left side of Fig. 1.5 [1] and the software architecture of the node in right side of Fig. 1.5. These nodes run Linux kernel 2.4.16 and all the hardware used in the node is supported by the kernel . The SH-4 cross-compilers are used to compile the code written onto these nodes. Sensoria provides the API [11] required for RF modem control and data acquisition.

1.3 Applications of Wireless Sensor Networks

The sensor nodes in a network can be used for continuous sensing, event detection, localization and tracking. This use of the network as a sensor opens up a wide variety of applications in civil, military and medical applications.

1.3.1 Military Applications

The WSNs can be an integral part of the military environment. The ease of deployment, self-organization and extensive sensing ability will help in surveillance, targeting, intelligence and control applications. All the environmentally-harsh terrain can be covered by a sensor network for surveillance and intrusion detection. Due to its continuous-sensing nature and varied sensing modalities, when deployed in the battlefield, it allows acquisition of the targets in the network and can also classify the target as a friend or foe. The WSN can also be effectively used to sense the chemical level in the environment in the event of biological or chemical attacks.

1.3.2 Civilian Applications

The sensor networks can be used in the civilian context for building surveillance. They can also be effectively used to monitor a forest fire before it spreads uncontrollably [25]. Other prospective applications are flood detection, Pesticide control in agriculture, unauthorized intrusion and creation of smart spaces for application-specific environments.

1.3.3 Medical Application

Wireless sensor networks have been successfully applied in various medical applications, including remote patient monitoring, in-hospital environment control, drug administration [25] and in-hospital patient and doctor surveillance. Sensors can be implanted into the human body and are capable of communicating with external computer systems via a wireless interface. Wireless networking of human-embedded

smart sensor arrays and a preliminary approach for wireless networking of a retina prosthesis is discussed in [26].

1.4 Challenges in Wireless Sensor Networks

As discussed earlier the sensor nodes are deployed in an indeterministic fashion like dropping from aircraft with no networking infrastructure. The sensor nodes should be able to communicate with each other and form an untethered network. This kind of self-configuring behavior is the first challenge in the creation of the sensor network. This kind of self-organization should help in building a fault-tolerant network allowing re-organization when a set of nodes dies out. Since there cannot be a centralized authority in this kind of environment, distributed fault-tolerant self-organization of the network is the first challenge in WSN research.

Sensor nodes rely on the power supplied by a battery which has finite energy. Minimization of energy consumption, the time integral of power, is important for extended battery life and subsequently the sensor network life [40]. Since replacing the battery of the sensor node is not an available option in sensor networks, each sensor network has a deterministic upper limit for its lifetime. To increase the lifetime of the network the energy usage should be optimized in all aspects like hardware design, radio transmission, signal processing and protocol behavior. Adaptive node scheduling and energy saving algorithms will be the next challenge for creating an effective sensor network. Apart from being energy efficient, the protocols should also be fault tolerant, to work around problems like nodes dying out, so as to maximize system life time [22].

The sensor network is deployed to observe the real time phenomena and subsequently the data dissemination protocols on the network should minimize the end-to-end latency in reporting the phenomenon to the monitoring authority.

Apart from these, the signal processing algorithms should be distributed and the decision making should be collaborative and scalable. To summarize the challenges in WSNs:

- Self-configuring and self-organizing behavior for infrastructure-less deployment.
- Energy efficient design of algorithms for maximum lifetime.
- Minimal end-to-end latency for data dissemination.
- Energy-efficient collaborative signal and information processing (CSIP).
- Scalable algorithms to thousands of sensor nodes.

This thesis discusses implementing an efficient method to address the above challenges in WSNs exploiting the application-level information. While in traditional networks the computers mainly interact with the users, the nodes of the WSN interact more directly with the physical world. The WSNs can be tasked to answer any number of queries about the environment under sensing. Although these networks provide us with increased amount of information, the limitations trigger us to explore new ways of communication, computing and integration paradigms to make efficient use of the information and to ensure that such systems operate reliably, safely and predictably.

The sensor node couples a tremendously diverse functionality with sensors, DSP circuitry, radio communication and computing ability. Throughout its lifetime a node may be called upon to be a data gatherer, a signal processor, and a relay station [31]. The efficacy of the WSN depends on a power-aware and application-aware system design. The optimization can be done at all levels of system hierarchy, including signal processing algorithms, operating system, network protocols, computing paradigm and even the integrated circuit level. Computation and communication are partitioned and balanced for minimal energy consumption [31]. In this thesis we explore the design and implementation of an efficient computing paradigm for collaborative processing in WSN called the *mobile-agent based computing paradigm*. The implementation is done on top of an energy-efficient and data-centric routing protocol for sensor network called the *Directed Diffusion* developed by Information Sciences Institute, University of Southern California [27].

1.5 Computing Models in WSNs

The WSN is a truly distributed network environment with resource limitation as explained earlier. Computing in this kind of environment is different from a fixed conventional TCP/IP based network. The two computing paradigms compared here are the *client/server* based paradigm and the *mobile-agent* based paradigm. Fig. 1.6 illustrates both paradigms.

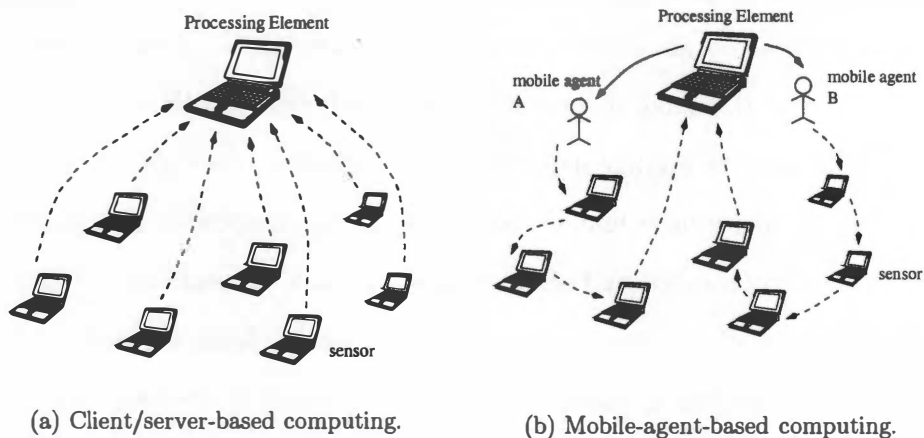


Figure 1.6: Different computing paradigms [34].

1.5.1 The Client/Server Approach

The client/server paradigm has been one of the most popular models adopted in distributed computing [12], where the server is a central processing node capable of receiving multiple requests/data from the clients, processing the data, generating the output and sending it back to the clients, while the clients are nodes which send request/data to the server and receive the response from the server. The client/server paradigm is shown in Fig. 1.6 (a). Most of the traditional network applications are designed using the client/server based approach. In the context of sensor networks, the client node collects data from the sensing modality and sends the data to the server to be processed. The server in turn handles such requests from clients and responds to them. This model, although widely used, has several disadvantages in the context of WSN.

The servers or central processing nodes usually demand more energy and computing power than peer nodes', is a luxury that cannot be achieved in this context.

First, the hostile environment in which the WSN exists and the thousands of nodes deployed in the field do not leave changing the battery and supportive resources as an option. Thus this kind of computing model is bound to reduce the lifetime of the network since the lifetime depends on the lifetime of the server node. Secondly, the sensor nodes have limited bandwidth of communication and power of transmission (both controlled by Federal Communications Commission (FCC)). In some applications, such as data processing or data fusion, large amounts of data must be mobilized between client and server, which potentially causes poor system performance. Finally, the performance of a client/server based system is defined by the number of clients and the estimated network traffic. However in WSN the random deployment of sensor nodes creates an unknown traffic and node layout pattern. It is not possible to adaptively configure the network with varying load in real time. When the number of sensors deployed increases they cannot perform the load balancing without changing the structure of the network.

1.5.2 The Mobile-Agent Based Approach

We implement a mobile agent (MA)-based paradigm partly as a solution to the above-discussed sensor network challenges. Mobile agents are basically programs, typically written in a script language, which may be dispatched from one computer and transported to a remote computer for execution [21]. The sensor network is a complex distributed system without any global authority. The client/server paradigm in such an environment is not effective or energy efficient. Using the MA, instead of N nodes reporting to one sever node as in the client/server paradigm, one mobile agent travels to the N nodes carrying out integration at each node. This

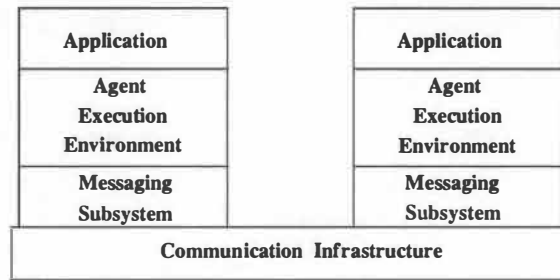


Figure 1.7: Conceptual model for mobile agent computing in WSN [21].

reduces the traffic in the network and is energy efficient. The autonomous migration of the MA provides the progressive accuracy of the integrated result. Fig. 1.7 shows the conceptual model for mobile-agent-based computing.

The agent program can be written to be executed in machine language or an interpreted language. To support the heterogeneity of the computing platforms in sensor networks an interpreted language is used. Using the mobile-agent-based paradigm the following goals are attained:

- Network bandwidth requirement is reduced. Instead of passing large amounts of raw data over the network, only the small agent is sent. This is especially important for real-time applications and where the communication is through low-bandwidth wireless connections.
- Better network scalability can be achieved. The performance of the network is not affected when the number of sensors is increased. Agent architecture can support adaptive network load balancing automatically.
- Extensibility is supported. Mobile agents can be programmed to carry task-adaptive processes and that extends the capability of the system.

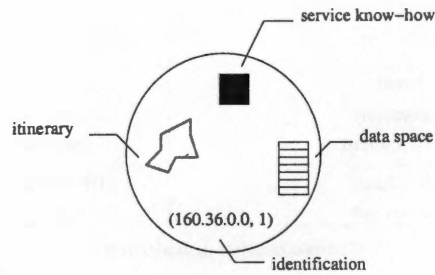


Figure 1.8: Mobile agent components [45].

- **Stability.** 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 system is not much affected by the reliability of the network. The agent can also take care of dead nodes, by-passing them during its itinerary.

The mobile agent is defined as an entity of four attributes: *identification*, *itinerary*, *data space*, and *method*, where identification is used to uniquely identify the mobile agent, data space is the agent's data buffer which carries the partially integrated results, itinerary is the route of migration, and method is the processing task (or execution code) carried with the agent [46]. The components are shown in Fig. 1.8.

1.6 Routing Protocols in Wireless Sensor Networks

The sensor nodes are mostly randomly distributed in a sensor network. The number of nodes in a WSN is usually in the magnitude of hundreds or thousands. Routing protocols designed for the ad-hoc [15] network do are not valid in sensor networks due to the sheer size of the network. Sensor nodes mainly use a broadcast communication

paradigm whereas most ad-hoc networks are based on point-to-point communication [25]. Akylidz et. al describe the principles for designing a network layer in the WSN [25]:

- Power efficiency is always an important consideration.
- Sensor networks are mostly *data-centric*.
- Data aggregation is useful only when it does not hinder the collaborative effort of the sensor nodes.
- An ideal sensor network has attribute-based addressing and location awareness.
- Sensor networks should be tailored to be application-specific to the sensing task at hand. Similarly, the protocols designed should support this nature of the WSN.

The three popular data-dissemination protocols under consideration for developing mobile-agent paradigm are *Flooding*, *Sensor Protocol for Information via Negotiation (SPIN)* and *Directed Diffusion*. All of these protocols are described below.

1.6.1 Flooding

Flooding is an old data-dissemination technique borrowed from traditional network and can be applied to wireless sensor network. In this protocol each node receiving the packet will broadcast the same further onto the network. This avoids the requirement of any complex route discovery algorithm for the packet. The packets are

broadcasted until the destination is reached. There are, however, several deficiencies exist in this protocol. If two different nodes share the same neighborhood then these nodes will likely receive duplicate packets and similarly if two nodes are in the neighborhood and detect any phenomenon they would send the same message to all the neighborhood. In either case the resources are not optimally utilized. This is a severe fall back considering the severe resource limitations in WSNs.

1.6.2 SPIN

SPIN is developed at MIT [23, 28]. It is a protocol designed for wireless sensor networks and is designed to address several deficiencies in *classic flooding* by negotiation and resource adaptation [25]. SPIN names its data using high-level data descriptors, the *meta-data*. The meta-data and the raw data have a one-to-one mapping relation. The format of meta-data is application-specific.

SPIN has three types of messages, ADV, REQ and DATA. The initiating node which has new data *advertises* (ADV) the data to its neighboring nodes using the meta-data. If the neighboring node needs this kind of data, it sends a *request* (REQ) to the initiator for the data. The initiator node responds and sends *data* (DATA) to the sinks. This mechanism of data dissemination is based on data-centric [23] routing where the sensor nodes broadcast an advertisement for the available data and wait for the request from interested sinks. Each node has its own resource manager to keep track of the usage of energy resource. Before data transmission, each node polls its resources to make a decision whether it should participate in the activity or cut it back. SPIN is essentially a flooding protocol, however, the use of meta-data for negotiation and the adaptation to resources available on the

sensor nodes help it eliminate most of the redundant data transfer, making it more selective in forwarding third-party data.

1.6.3 Directed Diffusion

Directed diffusion [17, 27] is developed at the Information Sciences Institute, USC. It is a robust, scalable, energy efficient and data-centric paradigm for data-dissemination in WSN. It is application-specific and provides good support for event-driven applications typical in WSN. Data gathered by sensor networks are named by attribute-value pairs. Sinks or nodes that request data send out *interests* into the network. If the attributes of the data generated by the source node match these interests, a *gradient* is setup within the network and data will be *pulled* toward the sinks. Intermediate nodes are capable of caching and transforming data. The interest and data propagation and aggregation are determined locally. The sink refreshes and reinforces the interest when it starts to receive data from the source. One efficient example explaining this paradigm is given in [27] as follows “A human operator’s query would be transformed into an interest that is *diffused* towards nodes in regions X or Y. When a node in that region receives an interest, it activates its sensors which begin collecting information about pedestrians. When the sensors report the presence of pedestrians, this information returns along the reverse path of interest propagation. Intermediate nodes can *aggregate* the data, *e.g.*, more accurately pinpoint the pedestrian’s location by combining reports from several sensors. An important feature of directed diffusion is that interest and data propagation and aggregation are determined by *localized interactions*.” The novel features constituted in diffusion are [27]:

- Data-centric dissemination;
- Reinforcement-based adaptation to empirically best path; and
- In-network data aggregation and caching.

1.7 Contribution of Research

This thesis concentrates on efficient implementation of the mobile-agent based computing paradigm for applications like target tracking and classification in wireless sensor networks [43, 47, 48]. The computing paradigm is built on top of Directed Diffusion. This new paradigm makes use of Diffusion's application programmer's interface (API) and the object streaming capability of the Python language [29] to design the *mobile-agent* based computing paradigm for collaborative processing in wireless sensor networks. The paradigm has been mapped to real-time experimental demonstrations for target classification in three field demos.

1.8 Structure of the Thesis

The organization of the thesis is as follows:

Chapter 2 describes the mobile agent paradigm in detail. It further discusses the implementation of mobile agent framework over Directed Diffusion in wireless sensor networks.

Chapter 3 shows the experiments conducted in three field demonstrations using MA paradigm for target classification.

Chapter 4 concludes the thesis and points out future work in the usage of MA in wireless sensor networks.

Chapter 2

Development of a Mobile Agent Framework on Directed Diffusion

As sensors of various type acquire networking and local processing capabilities, it is important to collaborate the spatially distributed sensing phenomenon between multiple sensor nodes to provide both reliable and comprehensive results. Multiple sensor nodes can perform functions previously impossible for any of the devices independently [49]. This kind of collaborative sensing and distributed processing environment demands an efficient computing paradigm. The client/server-based paradigm is a typical computing model in distributed processing. In this model the data or processed results are moved to a central server for further processing. Since the number of sensors in WSNs is very large, which causes large data transfers from each node to the central server node, the client/server-based paradigm not a justifiable option. In addition, the resource limitations on WSNs not support this kind of computing paradigm either.

The mobile-agent-based computing is proposed in order to support the collaborative processing applications in WSNs. In this approach, instead of each sensor node sending local data or results to a central node for processing, the information processing code is moved from sensor node to sensor node using mobile agents. These agents, as described earlier, will carry out local computation on each node and carry the result from node to node, integrating as it progresses. Agents by themselves are autonomous software programs specifically designed to handle a volatile network environment [13].

For this kind of computing paradigm to support a distributed sensing environment, it should be built upon an efficient routing protocol. The routing protocol should support the application-specific and data-centric nature of wireless sensor networks. Directed diffusion is one such protocol. It is data-centric and all the communication is for named data. All nodes in a directed-diffusion based network are application aware enabling it to achieve energy savings by selecting empirically good paths and by caching and processing data in-network [27].

This chapter discusses the development of the agent framework starting from the architectural overview of system in Sec. 2.1, with emphasis on different layers of the architecture described in Sec 2.2 and Sec 2.3. Finally the analysis of the architecture is done in Sec. 2.4

2.1 Architectural Overview

The architecture of the mobile agent framework (MAF) is shown in Fig. 2.1. Different layers in this architecture perform different tasks and provide upward layer

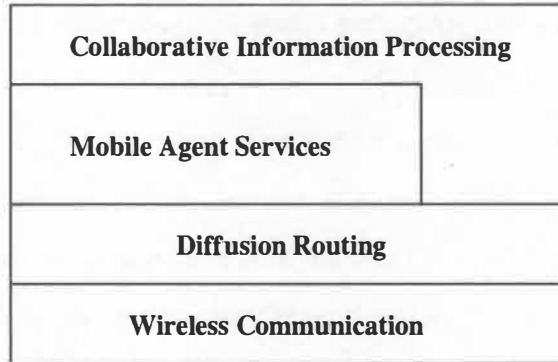


Figure 2.1: Architectural overview of the system.

support. At each layer the algorithms and methods are designed with concern towards power-efficiency and fault-tolerance [27, 34]. Compared to the traditional 7-layered network architecture this is a unique integrated cross-layer design that is application-oriented and data-centric. The collaborative information processing layer hosts algorithms for the integration of information derived from multiple sensors. The mobile agent services layer provides the mobile-agent-based framework to achieve the collaborative signal and information processing (CSIP) task. This mobile agent framework (MAF) is realized using the directed diffusion protocol which facilitates communication over the wireless link.

2.2 Directed Diffusion API

Designing protocols on a traditional layered architecture may not be optimal in all situations. Researchers have argued the need to collapse the protocol stack and design completely integrated protocol architectures for the application-specific requirements [16, 22, 30]. Exposing the application requirement to the lower layers

of the network will enhance the application-perceived quality of the network [22]. Both SPIN [23, 28] and Directed Diffusion [27, 35] protocols use application-specific data naming and routing to achieve energy efficiency in WSN. Directed diffusion is open source software and is well supported by the ISI research group. A network simulation environment of this protocol has also been added as an NS-2 (Network Simulator) extension which makes performance evaluation more convenient. Therefore, we choose to implement our MAF framework on top of directed diffusion. The basic mechanism of the Directed diffusion is discussed in chapter 1. This section describes the API usage and how MAF is developed using the API.

In directed diffusion the data are named using *attribute-value* pairs. For example, a target classification task can be described using the following set of attribute-value pairs:

```
type      = wheeled vehicle
instance  = Dragon Wagon
location  = [30,40]
confidence = 0.90$
timestamp = 02:30:45$
```

The task description specifies an *interest* for data matching [27]. The data sent in response to this interest request conforms to the same naming scheme making it easy at both the receiving side and transmitting side.

After conforming to a particular naming scheme, the *interests* or the task descriptions are diffused through the network. The node that diffuses these *interests* is called the *sink*. The *sink* node periodically broadcasts the *interest* messages onto the network watching to see if any node on the network detects an event (described in

the interest message). The interest is periodically refreshed by the sink by re-sending it from time to time. This repeated transmission is required for reliable transmission through the network. The refresh rate is protocol-dependent and trades off overhead for increased robustness to lost interests [27].

After these interests are broadcasted through the network each node maintains an interest cache. Each item in the cache corresponds to a different interest message. The interest entries in the cache do not contain any information about the sink. The cache has several entries like *timestamp*. It also contains *gradient* fields up to one per neighbor. These *gradient* fields contain information about the neighbors like the data rate requested by the neighbor, the expiration time of the interest sent by the neighbor (approximate lifetime of the interest), etc. When a new interest is received by a node it matches with the existing cache entries and updates them accordingly. Each entry will have a single gradient toward the neighbor from which the interest is received and with the specified event data rate [27]. If a node receives an interest message, it in turn broadcasts only to a subset of neighbors determined by whether it has re-sent a matching interest recently.

When an event occurs on the network, the sensor node searches its interest cache for matching interest entry. It then sends the event description to each neighbor to whom it has a gradient. The *data* message is unicast individually to the relevant neighbors. The *sink* finally will receive data from multiple paths. The sink then *reinforces* one particular path from the neighbor in order to “draw” data at high data rates. This feature of diffusion is achieved by *data driven* local rules [27]. Nodes rely on the data cache to do reinforcement. Fig. 2.2 shows the above-described mechanism of the diffusion. Diffusion is a receiver-initiated protocol. The matching

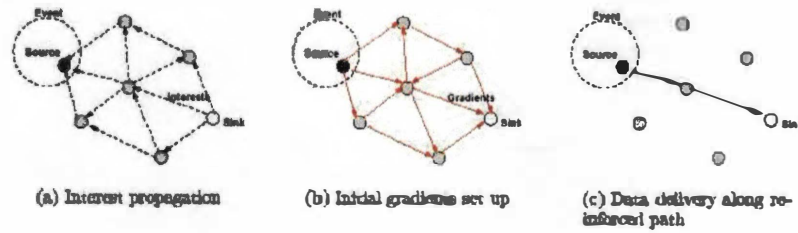


Figure 2.2: A simplified schematic of directed diffusion [27].

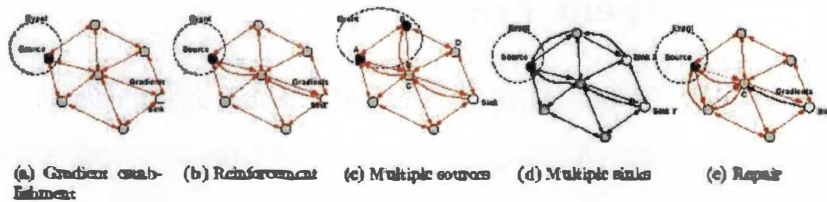


Figure 2.3: Illustration of different aspects of diffusion [27].

data are “drawn down” towards the sink. Fig. 2.3 illustrates different aspects of diffusion such as multiple sources and multiple sinks.

Diffusion is started on each node by running a diffusion daemon. Each node is setup with a unique node ID either randomly or by the user. After all the nodes are running, diffusion daemons, the gradient daemon is run on all of them to support the gradient formation. The diffusion discovers the other members of the network by periodically sending out exploratory messages over the network. It usually takes a few minutes for all the nodes to settle down with proper identity. The applications using diffusion utilize the API provided to talk to the diffusion layer on the network.

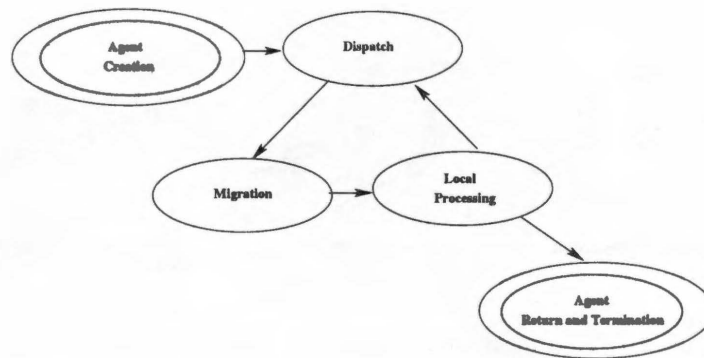


Figure 2.4: MAF - finite state machine.

2.3 Mobile Agent Framework

The mobile agent framework (MAF) consists of a mobile agent daemon or server that runs on each node and acts as a host to the incoming *mobile agent*. The mobile-agent itself is a program that flies from one node to another collecting and integrating local results from each node. The mobile agent server hosts the agent and completes the required processes and passes the agent to the diffusion layer to be relayed through the radios. All the agent services provided by the server are tightly coupled with the diffusion API for communication purposes. Initially all the nodes subscribe to diffusion with the respective node IDs as attributes. This section describes the design flow of MAF.

2.3.1 MAF - The Finite State Machine

Fig. 2.4 illustrates the life cycle of an MAF using a finite state machine. The agent is created when an event is detected. The agent is then dispatched to the neighboring node in a pre-defined itinerary. The mobile agent server running on the node dispatches the agent to its destination by accessing the diffusion layer which

in turn accesses the RF wireless communication interface. Since the agent carries the execution code and the results from a previous migration, it carries out local processing on each node and integrates the result with the earlier result. This is how the collaborative processing task is achieved. When the agent attains a certain threshold of accuracy or reaches the last node of migration it terminates and displays the final result.

2.3.2 Mobile Agent Creation

The agent is created when an event is detected. The mobile agent daemon on the node creates the agent encompassing the local results and the current state of the execution code. As described earlier, the mobile agent, upon creation, is defined as an entity of four attributes:

- **Identification:** Identification uniquely identifies the agent in the network. Different mobile agents dispatched from different places of the network are uniquely numbered such that they can be differentiated from one another.
- **Itinerary:** The itinerary is the route of the mobile agent. The route of the agent is either derived on-the-fly dynamically or pre-defined by the user.
- **Data Space:** Data space is the agent's buffer carrying the agent's partially integrated result. As the agent progresses, it integrates the result from each node. The local result is derived, based on the data captured on each node, which is fused with the partial result contained in the agent from the previous migration.

- **Service know-how:** Service know-how is the processing task or the execution code carried by the agent.

2.3.3 Mobile Agent Dispatch

Once the agent is created it carries with it its itinerary and migrates to the next node using diffusion API. The agent is streamlined into a string literal type allowing it to keep the current values of variables and execution status which will be recovered when arriving at the next node. This process is called *object serialization*. During dispatch the agent accesses the diffusion API and publishes its interest over the network. Since the agent constitutes the itinerary, the matching interest the agent sees the network is its next destination address. The following code segment shows the agent setting up publication in a network, where the `TargetAttr` specifies the *nextip* or the next node address of the agent.

```
handle setupPublication(NR *dr, char *nextip){
    NRAttrVec attrs;
    attrs.push_back(NRClassAttr.make(NRAttribute::IS, NRAttribute::DATA_CLASS));
    attrs.push_back(LatitudeAttr.make(NRAttribute::IS, 60.00));
    attrs.push_back(LongitudeAttr.make(NRAttribute::IS, 54.00));
    attrs.push_back(TargetAttr.make(NRAttribute::IS, nextip));
    handle h = dr->publish(&attrs);
    ClearAttrs(&attrs);
    return h; }
```

2.3.4 Mobile Agent Migration

When a node with data matching the interests of the agent is discovered on the network, the agent migrates to the specified node. The itinerary of the agent is currently stored in the `/tmp/maf.conf` file of the node. An example of the itinerary file is shown as below. The variable “hops” represents the number of nodes the agent should migrate before reaching destination. The node names or the corresponding IP addresses specify the order of migration. The nodal names or IP addresses belong to each local node and are the same as the `TargetAttr` described earlier. Each node’s MAF server subscribes to the Diffusion using this name/address. When the agent needs to migrate to a particular node (say `node0` with address `160.36.31.2`) it publishes the interest on the diffusion looking for this name/address (`160.36.31.2` in this case). When a matching subscription is discovered the agent migrates to the corresponding nodes. `np` indicates the number of parameters the agent carries with it representing the integration result.

```
[itinerary]
hops=2
node0=160.36.31.2
node1=160.36.31.2
[parameter]
np=3
```

The MAF implemented in this thesis uses a static itinerary pre-defined in each cluster. However, the ideal mobile agent should migrate deciding its destination on-the-fly.

2.3.5 Local Processing

As described earlier the MAF is used for multi sensor fusion to produce reliable information. The local processing of the result is carried out on each node. Upon arrival the agent is hosted by the mobile agent server running on each node. A multi resolution integration (MRI) [34] function is used here to integrate the result from one node to another. The agent will progressively carry the result as it migrates and integrates the local confidence range from each node finally arriving at a multi-sensor collaborative confidence range depicting the target type.

2.3.6 MAF - Implementation

The MAF is coded using the Python scripting language. The interface with Diffusion API and application services, which provide C/C++ API, is done by generating shared library modules accessible both by the Python and C++ code. These shared libraries are generated using a code development tool called Simplified Wrapper and Interface Generator (SWIG) [10]. SWIG is an open-source software development tool that connects programs written in C and C++ with a variety of high-level programming languages. It is primarily used with common scripting languages such as Python, Tcl/Tk to create high-level interpreted programming environments, and as a tool for testing, prototyping and interfacing C/C++ API and software.

Fig. 2.5 shows the stages in the implementation of MAF. Python is used for coding the agent and server-side software for the MAF. Python is an interpreted, interactive, object-oriented programming language. This language is chosen for its ability of *object-serialization* which is used to generate the agent. This process variously called as *pickling*, *serializing*, *marshalling* or *flattening*, allows the *Python*

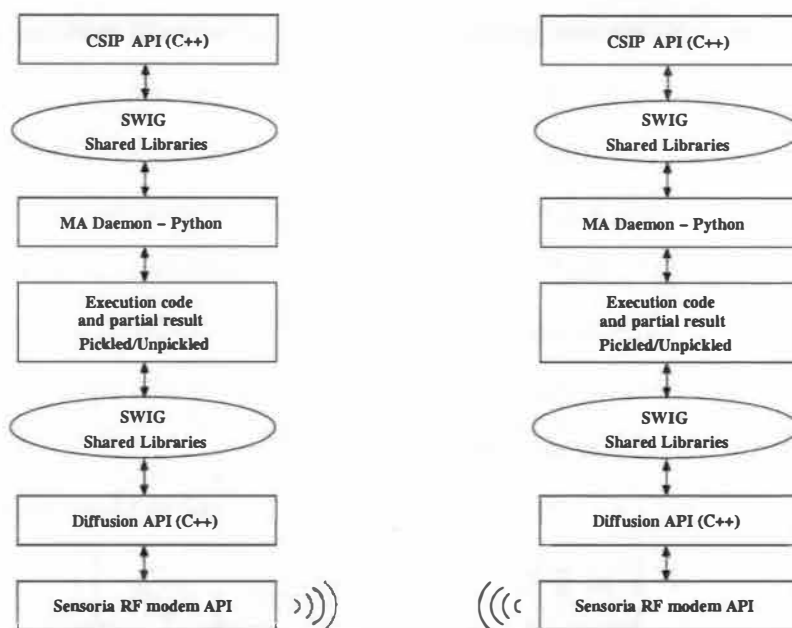


Figure 2.5: Implementation of MAF

in-memory objects to form a single linear string format, suitable for storing in flat file, shipping across the network sockets and so on [29]. The resulting stream file can be converted back into *Python* objects by the unpickling process. Python has modules, classes, exceptions, very-high-level dynamic data types, and dynamic typing. It contains interfaces to system calls and libraries, as well as to various windowing systems [8]. The Python implementation is portable and is an open source software written in C/C++. Unlike compiled codes like C++, where a full-fledged executable requiring no dependencies can be generated using cross compilers, Python code is an interpreted code and an interpreter is required on the system where the code needs to be interpreted/executed. For implementing MAF on Hitachi SH-4 based sensor nodes Python source code is cross-compiled using the SH-4 cross-compilers to generate an SH-4 version of the Python interpreter. The resultant interpreter is

configured so as to occupy the least space with only the required modules built into it. This is due to the limited availability of memory on the node.

POSIX threads are used to keep the node in a wait state while waiting for mobile agent to arrive. The two primitive thread-synchronization primitives are *mutex* and *condition*. Mutexes are simple lock primitives that can be used to control access to shared variables. The *flag_mutex* variable is locked until triggered by a signal thus maintaining the status of the loop making it a light weight process (making it consume a lower percentage of CPU time while idle).

Fig. 2.6 and Fig 2.7 show the complete flow chart of the MAF server and agent execution respectively.

Fig. 2.8 shows the processor usage of the MAF server during idle state. The last three lines in the screen shot with PIDs (second column) 269, 270, 271 are the corresponding processor usage lines for the MAF server while idling. The Hitachi SH-4 averages 1.5 - 3.0% CPU usage for running the MAF server daemon. Fig. 2.9 shows the typical screen shot of the server running in a sensor node.

2.4 Analysis of the Architecture

The MAF has many advantages compared to the client/server model. The different aspects of the mobile agent are summarized below.

- **Scalability:** In sensor networks, the number of nodes can be hundreds or thousands. Agent architectures are adaptive to scaling of the network by changing the number of nodes the agent migrates through [37, 21]. Unlike in client/server architectures, the agent does not create any queue as the number

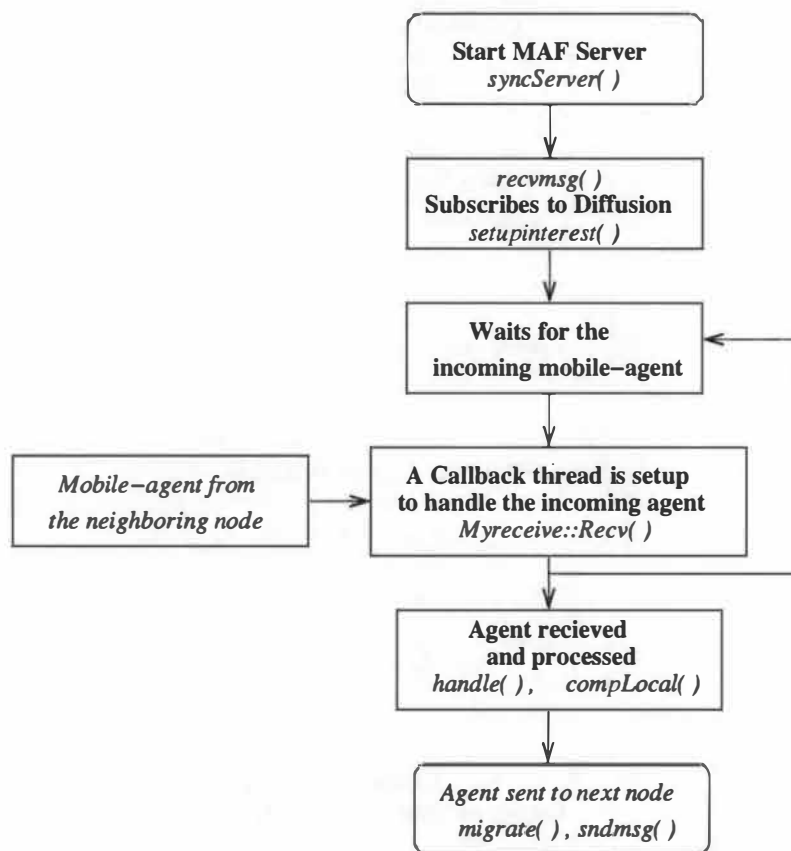


Figure 2.6: Flow chart of MAF server.

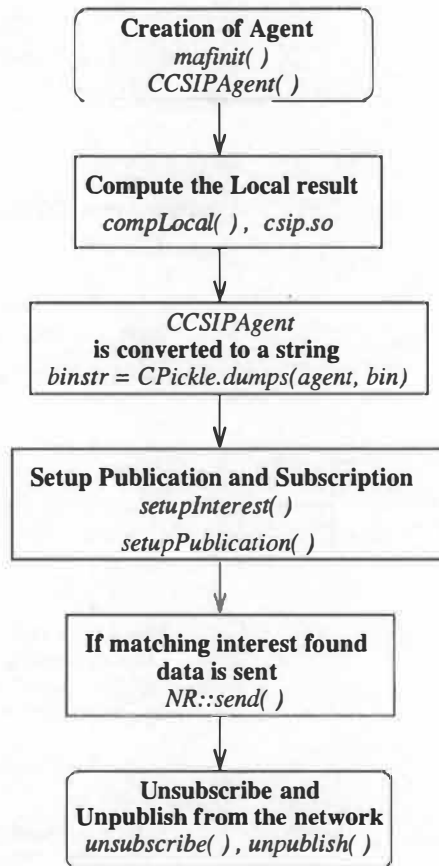


Figure 2.7: Flow chart of mobile-agent.

```

root@node1.ece.utk.edu: /mnt/piranha/run
11:04pm up 23:04. 1 user, load average: 0.10, 0.04, 0.01
procs
r b w swpd free buff cache si so bi bo in cs us sy id
0 0 1 0 42852 38 11928 0 0 0 0 0 115 16 0 1 99
USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND
root 1 0.0 0.8 1448 540 ? S 00:00 0:02 init [3]
root 2 0.0 0.0 0 0 ? SW 00:00 0:00 [keventd]
root 3 0.0 0.0 0 0 ? SWN 00:00 0:00 [ksftirqd_CPU0]
root 4 0.0 0.0 0 0 ? SW 00:00 0:00 [kswapd]
root 5 0.0 0.0 0 0 ? SW 00:00 0:00 [bdflush]
root 6 0.0 0.0 0 0 ? SW 00:00 0:00 [kupdated]
root 8 0.0 0.0 0 0 ? SW 00:00 0:00 [mtdblockd]
root 12 0.0 0.0 0 0 ? SWN 00:00 0:00 [jffs2_gcd_mtd4]
root 26 0.0 1.0 1688 632 ? S 00:00 0:00 /sbin/devfsd /dev
blu 64 0.0 0.7 1608 480 ? S 00:00 0:00 putmap
root 80 0.0 1.0 1828 668 ? S 00:00 0:00 syslogd -m 0
root 84 0.0 0.8 1496 552 ? S 00:00 0:00 klogd
root 109 0.0 1.1 1560 704 ? S 00:00 0:00 /sbin/cardmgr
root 127 0.0 1.2 1816 756 ? S 00:00 0:00 /sbin/pump -l eth0
root 136 0.0 1.0 1820 676 ? S 00:00 0:00 inetd
root 169 0.0 1.3 2240 820 ? S 00:00 0:00 rfmodemd -r 0 -p sensit 1 --daemon
root 174 0.0 1.1 2084 728 ? S 00:00 0:00 heartd -r 0 --daemon
root 179 0.0 1.1 2088 728 ? S 00:00 0:00 streamd -r 0 --daemon
root 182 0.0 1.3 2240 820 ? S 00:00 0:00 rfmodemd -r 1 -p sensit 2 --daemon
root 187 0.0 1.1 2084 724 ? S 00:00 0:00 heartd -r 1 --daemon
root 189 0.0 1.1 2088 724 ? S 00:00 0:00 streamd -r 1 --daemon
root 196 0.0 1.1 2032 684 ? S 00:00 0:00 staled
root 200 0.0 0.8 1428 512 ttsc/1 S 00:00 0:00 /sbin/agetty 115200 ttsc/1 linux
root 231 0.0 0.0 0 0 ? SW 00:04 0:00 [rpciod]
root 232 0.0 0.0 0 0 ? SW 00:04 0:00 [lockd]
root 238 0.2 1.1 1784 708 ? S 22:57 0:01 lr.telnetd: piranha.ece.utk.edu
root 240 0.2 2.1 2520 1356 pty/s0 S 22:57 0:01 -bash
root 269 1.3 4.0 6900 2492 pty/s0 S 23:01 0:02 Python/python maf/Cmafserv.py
root 270 0.0 4.0 6900 2492 pty/e0 S 23:01 0:00 Python/python maf/Cmafserv.py
root 271 0.0 4.0 6900 2492 pty/s0 S 23:01 0:00 Python/python maf/Cmafserv.py
root 456 0.0 1.3 2916 856 pty/s0 R 23:04 0:00 ps -aux

```

Figure 2.8: Screen shot of processor usage, the last three lines (PIDs 269, 270, 271) show the processor usage by the mobile agent daemon in idle state.

```

root@node1.ece.utk.edu: /mnt/piranha/run
File Edit View Terminal Go Help
root@node1 /mnt/piranha/run
Python/python maf/Cmafserv.py
Initialize MAF service, waiting for incoming agent ...
node1.ece.utk.edu
Diffusion Routing Agent initializing... Agent Id = 1028
waiting for something to come :^

```

Figure 2.9: Screen shot of MAF server.

of nodes increase because of the absence of a centralized server.

- **Energy Efficiency:** The agent technology reduces the total amount of data to be transmitted over the network. This helps the MAF to distribute the energy consumption of the nodes evenly upon the network, enabling longer network life, since on a sensor node most of the energy is consumed by the radio transceiver [18]. The minimal size of the agent also reduces the network traffic and utilizes the network bandwidth efficiently. This is important considering the low bandwidth of the wireless connections.
- **Progressive Accuracy:** The MAF provides progressive accuracy to the task performed. As mentioned earlier the agent migrates from one node to another carrying the partially integrated result. At each node the agent integrates the new result with all the previous results, which has the potential to gain accuracy as it migrates along the network. A user-defined threshold of accuracy can be set to terminate the migration of the agent at any stage.
- **Autonomy of the Agent:** Sensor networks are usually deployed in unmanned terrains to help in unexpected event detection. In such cases mobile agents provide the biggest advantage due to their autonomy. The agent can be automatically triggered upon an event detected by the sensors and continue its computation, migration and conclusion of result. This, compared to each node reporting to a server or main sub-system as in the client/server paradigm, is advantageous and provides practical real-time performance.
- **Reliability:** The agents are autonomous programs and can function independent of the network status. An agent can be deployed when the network

connection is alive and the results can be collected when the network connection is alive without loss of information. It is common in sensor networks for certain nodes to die, in such cases the agent can be programmed to be adaptive to the connection status and bypass the nodes that are not alive in its path. This increases the fault-tolerance and real-time nature of the applications built on this framework. In addition, Diffusion uses the unreliable UDP style packet messaging, thus this feature of agent can provide more stability and reliability of the sensor network applications. This facilitates a pervasive, open, generalized framework for the personalization of network services [21].

Considering the above features of the agents, MAF approach provides important qualitative services for collaborative processing in wireless sensor networks. It reduces the amount of data moving on the network and is a promising approach for distributed computing in wireless sensor network. However we should also realize its limitations. The agent framework requires a highly secure agent execution environment since there is no built-in security in the agent. The MAF also requires a balance between number of nodes the agent migrates through before returning and the efficiency and redundancy of the information agent carries. The MAF does not provide any re-transmission capability once the agent is dropped for some reason in the network.

Chapter 3

Experimental Demonstrations

The MAF discussed in Chapter 2 has been applied in multi-sensor collaborative processing for ground-target classification in wireless sensor networks. This application has been tested and validated in three field demos under different network configuration and scenario designs. This chapter describes the sensor node setup and experimental results obtained from these demos.

3.1 System Overview

The project is a collaborative effort involving participation from different universities and research institutions. The systems configuration at a sensor node is shown in Fig. 3.1. At the lowest level, Sensoria provides a micro-sensor platform, which performs sensing independently. The raw data collected is pre-processed using low-level signal processing functions provided by BAE Austin. Upon event detection, UTK's mobile agent based collaborative target classification routine will be invoked. The mobile agent framework is realized on top of ISI's diffusion routing.

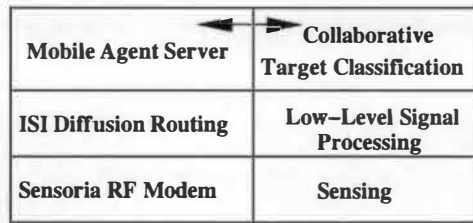


Figure 3.1: Overview of nodal architecture for collaborative classification.

3.1.1 Sensor Node Platform

The Sensoria WINS (Wireless Integrated Networked Sensors) NG 2.0 node is chosen as the platform for implementing the algorithms. The WINS NG platform is capable of monitoring multiple sensors, carrying out local processing and communicating the results with other nodes using wireless radios. Physically the WINS NG node contains a high performance analog sensor sampling unit, a sensor digital signal processing, a dual channel spread spectrum wireless network solution, a 32-bit application processor and a POSIX-compliant real-time operating system.

The WINS NG 2.0 node platform architecture includes a real-time interface processor. This processor supports high-speed multi-port sampling integrated with both a high speed DSP and digital I/O. The interface processor accesses the sensor data and helps in DSP of the sensor data. This dedicated programmable digital signal processor enables local processing of high-bandwidth sensor data (upto 20K samples/sec per channel input) in addition to providing upto a total of 580K-samples/sec throughput from the DSP to the application processor [11]. DSP control and processing allows a scalable output word rate ranging from 156Hz up to 20KHz. However the DSP provides only the data sampling and the unprocessed sensor data is supplied to the processor and available to the application developer through the

Platform Processor Core	Hitachi SH-4 7751
Platform Memory	16MB RAM, 16MB Flash
Processor Architecture	<ol style="list-style-type: none"> 1. 32bit RISC architecture 2. Superscalar dual issue architecture 3. Low power consumption with 1.8V internal, 3.3V I/O bias 4. Branch operations: folding, prediction, conditional prefetch 5. 16K data cache and 8K instruction cache 6. Memory Management Units (MMU) 7. Floating Point Unit with Single and Double Precision
Processor Performance	<ol style="list-style-type: none"> 1. Core: 300 MIPS 2. Floating Point Unit: 1.1 GFLOPS
Processor Power Dissipation	400 mW

Figure 3.2: Processor specification.

Sensor API (API provided by Sensoria) for further processing. The architecture of the node includes a 32-bit RISC Hitachi SH-4 processor including an FPU and accompanied by RAM, ROM and flash memory. Fig. 3.2 shows the detailed information on the processor. The detailed description of different sections of the node are shown in Figs. 3.2, 3.3, 3.4, 3.5, 3.6, and 3.7.

The node is equipped with an embedded GPS device, which provides geo-location and timing data. It can be accessed as a device file by the software through /dev/ttyS2 of the node. This device gives the GPS data in standard National Marine Electronics Association (NMEA) format. Detailed FAQ about NMEA is available from [7]. The NMEA message set can be parsed to obtain the geographical position of the node. The specifications of the GPS module are given in Fig. 3.6.

The wireless network interface of this node includes a dual-mode RF modem system that enables a solution for scalable, multi-hop networking with spread spectrum signaling. The RF modems are discussed in detail in Sec. 3.1.2.

Processor	Low power digital signal processor control system
Frequency (MHz)	80
	80
Data & Program Memory	16k internal 128k external words (x16 bit addresses per word) External memory shared between host and DSP
Arithmetic and Logic Unit	
Software System	Software implemented for sampling management only. Unprocessed sensor data provided to Processor.

Figure 3.3: High speed analog sampling system.

Battery Lifetime	3 hour continuous use 20 hour standby operation
Battery charging time	5 hours from uncharged.
Adapter	120V 60Hz to 12V DC in

Figure 3.4: Power specification.

High Speed S/H ADC	
4-ch differential front-end	
Resolution	16bits*
Sample rate	20kHz
Selectable gains	2, 11, 101, 1001V/V (6dB, 20.8dB, 40.1dB, and 60.01dB)
Selectable output word rates	20K, 10K, 5K, 2.5K, 1.25K, 625, 312.5, 156.25Hz
Digital LP filtering and decimation for word rates below 20kHz	
Fixed 8th order Butterworth anti-alias filter	-20dB @ 5kHz -3dB @ 6kHz -35.5dB @ 10kHz
Input noise voltage	15 nV/rt(Hz)
Linearity metric to front-end (Independent Linearity Error)	
Gain non-linearity	20ppm**
ADC Integral non-linearity	Max ± 6LSB
*Lower resolution at higher output word rates.	
4ch @ 20kHz	ENOB=12 bits
1ch @ 20kHz	ENOB=13.5 bits
1ch @ 1.25kHz	ENOB=15.5 bits
(ENOB effective number of bits)	
** Maximum deviation from best fit line	

Figure 3.5: High speed analog front-end specification.

Received codes	L1, C/A code
Channels	12
Max. Solution update	1 Hz
Satellite Reacquisition Time	100ms
Snap Start	< 30 seconds average
Hot Start	< 8 seconds average
Warm Start	< 30 seconds average
Cold Start	< 45 seconds average
Minimum signal tracked	-175 dBm
Maximum altitude	< 60,000 feet
Maximum velocity	< 1000 knots
Protocols	NMEA v2.2
Position Accuracy	10 meter 2dRMS, WAAS enabled 1-5 meter, DGPS corrected
GPS antenna cable length	6 feet

Figure 3.6: GPS specification.

Frequency	2.4-2.483GHz
Coding	Frequency Hopped Spread Spectrum (FHSS)
FCC Certification	FCC part 15.247, ETS 300-328 and RSS210 rules, license free
Channel Data Rate	56kbps node-to-node
Number of frequency channels	79
Independent networks	64 (for both modes)
RF Bandwidth	750kHz
Transmit power	10mW or 100mW
Outdoor operating Range	25m worst case (zero elevation, cluttered, no LOS) 100m surface-to-surface LOS 500+m elevated with line of sight
Indoor operating range	25m-100m

Figure 3.7: RF modem specification.

Apart from the hardware, the node contains a fully POSIX-compliant operating system - Linux 2.4.16. This provides the developer with all the API of the sensor node devices. The node does not come with compilers but cross-compilers for the Intel machines are provided to do software development for these nodes. All the sensor channels, GPS module, RF modems are available to the developer as device files from /dev directory of this Linux-like directory structure. The node provides the user with 16 MB of flash memory to load the executables and 16 MB of RAM for computation. The operating system supports the network file system (NFS) services so the developers can create a drive space mounted over the network to the node. The node supports a wired 10 Mbps Ethernet and can be used for all communications to the nodes, i.e for loading/unloading of software/data. The volatile memory space is available in the /tmp (RAM disk) directory of the node and all the network mounts and the local data acquisition or file storage are done into this directory.

3.1.2 Sensoria RF Modems

The embedded RF modems in Sensoria nodes provide a low-power networking solution, however at low data rates. The RF modems can be accessed both by the command-line utilities and by using the API provided by Sensoria. The RF modem API is based on the open source Framework for User Space Devices (FUSD) interface which makes them accessible via standard device files interfaces promoting portability and accessibility [11].

As mentioned earlier, each node consists of two RF modems, each supporting 2.4 GHz frequency-hopped spread-spectrum communication. A network is composed of one modem operating as base and one or more modems operating as remotes. Each

remote can only *unicast* to the base. The base can *unicast* to a single remote or *broadcast* to all remotes. The two modems can be operated simultaneously and can put the node simultaneously into two different networks. The modems can be accessed by a device file interface. The node having two modems allows the implementation of high performance, multi-cluster, low-latency multi-hop networking applications [11]. The WINS NG 2.0 RF modem implements a star topology where one modem acts as *base* and some other modems acts as *remotes*, each connected to the *base* by a separate logical channel. Within the WINS NG 2.0 modem, the link synchronization is based on an underlying TDMA frequency-hopped implementation, in which the base serves to synchronize the TDMA frames and coordinate the allocation of slots to remotes.

Each of the two modems on the network should select a *network number*, an integer between 0 and 63. Each of the modems can be a *base* or *remote*. There can be two *remotes* on a particular node but there can only be one base on either of the nodes. The network numbers correspond to separate hopping sequences that the modems follow when communicating with their peers. Because the sequences are separate, two modems that have selected different network numbers will rarely interfere with each other. This is similar to two modems selecting different fixed channels for communications, but in addition, frequency hopping avoids consistent loss when a portion of the spectrum is experiencing high interference [11].

The WINS NG 2.0 node provides the C language API and is used by ISI Diffusion to transmit and receive packets over the network. Diffusion in turn provides a more user-friendly API to the developer to port applications on to it.

3.1.3 BAE Low Level Signal Processing

This is a BAE repository environment used for low-level signal processing. The Sensoria nodes can accommodate four different kinds of sensors on four different channels. We use three sensor channels for data acquisition namely passive infrared sensor (PIR), seismic sensor and acoustic sensor. The Sensoria node provides application programmer's interface (API) to access the sensor hardware. BAE uses this API to access the sensors. Each node will perform sensing independently and the raw sensor data are processed by the BAE low-level signal processing daemons running on the nodes. The raw sensor data are pre-processed, sampled and made available as a repository (flat file format) placed at a specific location in node's memory to be available for other application developers. BAE uses a sampling rate of 4960.32 Hz on the time-series data from each sensor with a buffer size of 256 samples and an FFT window of the size 1024, to pre-process the data. The time-series repository can be accessed by developers using subscription methods provided by BAE's API. Each node consists of a `/tmp/config.rep` file. This is a configuration file for the BAE's repository indicating which type of sensor is connected to which channel of the node.

3.1.4 UTK Target Classification

The UTK classification services access the data generated by the BAE and perform the classification task using supervised learning techniques like kNN [39, 41, 42]. The classification services use MAF and the lower layers to communicate its result with other nodes in the network. The UTK classification services consist of a daemon running on each node subscribing to BAE's repository. This daemon is responsible

for performing local processing of the repository data upon event detection. The event detection is determined by a pre-defined threshold of the signal level, beyond which an event is assumed to have happened and classification is run on the available data of that event from the repository. The classification result from each node is a confidence range of the target existence [42]. The classifier output is placed in the `/tmp/range.dat` file of each node. Upon completion of the classification the classification daemon invokes the mobile-agent, starting the collaborative signal and information processing (CSIP).

3.1.5 MAF based Collaborative Processing

The MAF server runs on each node hosting the incoming agent. The mobile-agent initiated by the classification daemon picks up the local confidence range and travels to the next node according to its itinerary. At the next node the agent further integrates the corresponding result from the `/tmp/range.dat` file. This migration continues until it reaches the last node in its itinerary or achieves required accuracy. The integrated result from each node is generated using the multi-resolution integration algorithm (MRI) [34, 42]. Fig. 3.8 shows the mobile-agent migration and integration of the result among a cluster of three sensor nodes. Initially an event is detected and each node consists of the local confidence range written out. The agent flies from node 1 to node 2 and finally to node 3 where the final result is displayed based on the deduction from the integrated result.

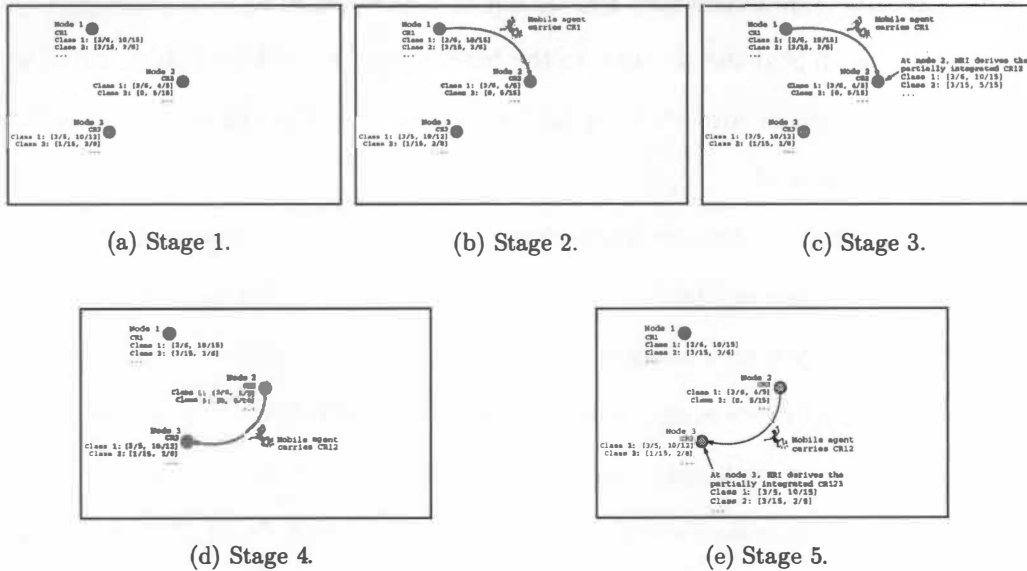


Figure 3.8: Mobile-agent-based multi-sensor fusion [42].

3.2 SITEX02 Demo

This field demo sponsored by the DARPA sensor information technology (SensIT) program took place at Twentynine Palms, California in November, 2001. The purpose of this demo was to test MAF-based CSIP in realistic environment. The UTK AICIP (Advanced Imaging and Collaborative Information Processing) lab is one of the participants. Others include Auburn University, Applied Research Laboratory/Penn State, BAE systems, BBN (Integrators of the demo), University of Wisconsin, Xerox-PARC, MIT-Lincoln Labs, ISI/USC, Rutgers, UCLA, University of Maryland, Fantastic-Data, Cornell, Duke and Virginia Tech. BBN technologies is the integrator for the demo and has the central command for all the nodal administration and setup. The setup is done at the Marine Corps Air and Ground Combat

Center (MCAGCC), Twentynine Palms, CA at a location 30 miles northeast of the “Mainside” which is at the entrance to the base in the Twentynine Palms, CA. This is located in the most arid parts of the Mojave desert. Fig. 3.9 shows the sensor node setup in the field.

In this demo 70 nodes are deployed in close proximity for improved density of coverage. The nodes are laid out in the field as shown in Fig. 3.10. The nodes are placed in an open area along side three roads and their intersection point. The north-south leg lay down is shown in Fig. 3.11, which also shows radio connectivity. As explained earlier each node has two radios and each radio participates in two different networks forming a complete linkage in the network. In Fig. 3.11 the nodes in yellow indicate they have one of the radios as base and the nodes in green have both of their radios as remotes. The network number of channel 1 and channel 2 of each node is depicted in the parenthesis beside each node number. Fig. 3.12 shows the similar node lay down and radio diagram in the east-west direction and the center of intersection. This experimental setup provided us with a dense field of sensor nodes from Sensoria with real-world, coherent signals from targets related to battlefield, supporting development of target classification and tracking applications.

The vehicles used in the demo are shown in Fig. 3.13. The targets primarily consist of:

- **HMMWV** : This is a diesel engaged light-weight wheeled vehicle.
- **Dragon Wagon (DW)**: This is a heavy, twin-axled, wheeled logistics vehicle system with good acoustic and seismic signatures.
- **Amphibious Assault Vehicle (AAV)**: This is a heavy amphibious full-

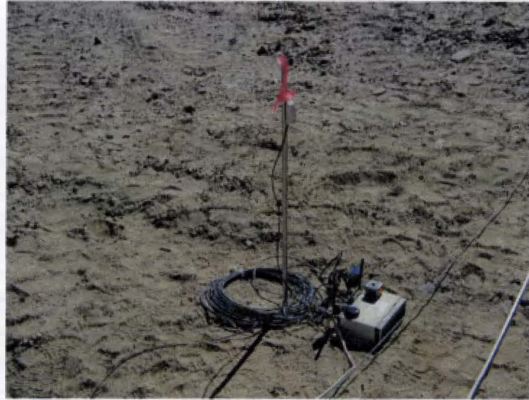


Figure 3.9: Sensor node in the field with all sensors.

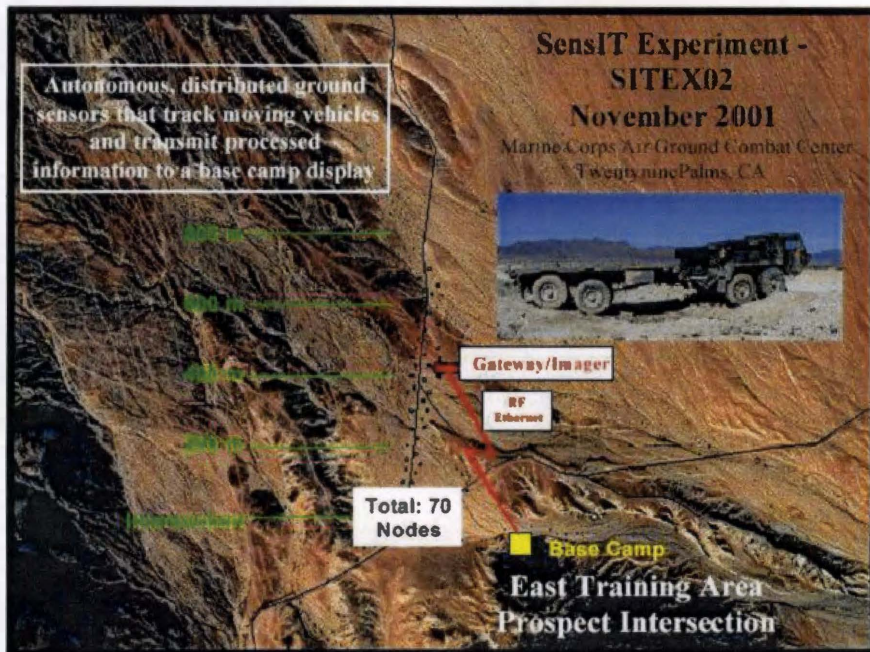


Figure 3.10: Plan of the DARPA SensIT experimental demo [38].

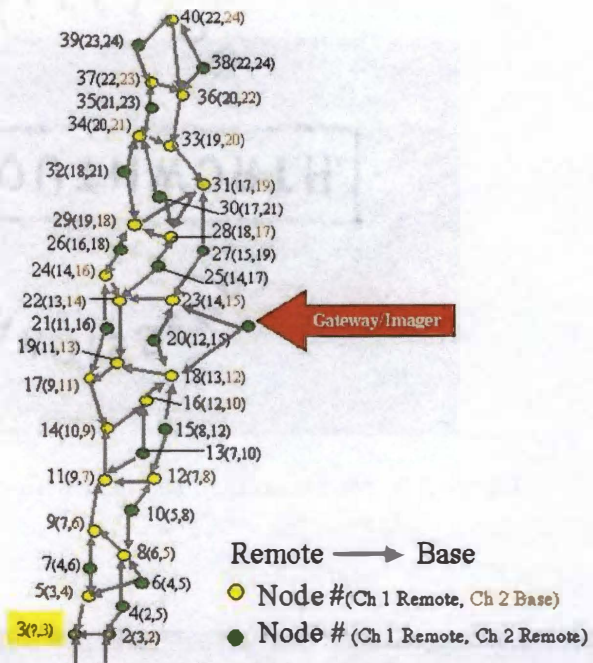
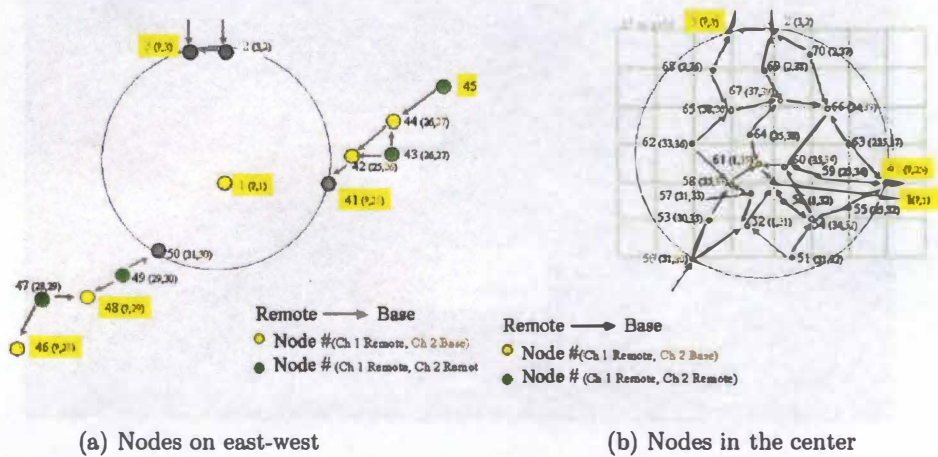


Figure 3.11: Node distribution and radio configuration on north-South leg [38].



(a) Nodes on east-west

(b) Nodes in the center

Figure 3.12: Nodes on east-west and center [38].



Figure 3.13: Vehicles used for classification at 29 Palms.

tracked vehicle with distinctive acoustic and seismic signatures.

Each node in the field is wired with a 10Mbps Ethernet connection to a central server. Each node is mounted on this central server through NFS (Network File System). Thus placing software on this mount point will make it accessible locally at each node. The Ethernet allows fast data archiving. Each node is run with the following modules in the background all the time:

- BAE low-level signal processing daemon.
- ISI Diffusion and gradient over RF modem.
- BBN Logger for logging each node information, traffic on the networks, etc.

Apart from these, UTK's classifier daemon and mobile agent daemon are also run in the background. Three tri-axle seismic sensors are provided in all the three directions of the testbed to support sensitive seismic detections apart from the sensors on each node. The MAF used in this demo is implemented on a traditional TCP layer and uses the wired Ethernet for transferring the agent. Five clusters of four nodes each are setup in each direction (See node layout figures for location):

- Cluster 1: North-South leg, Nodes:2, 3, 4, 5.
- Cluster 2: North-South leg, Nodes:11, 12, 13, 14.
- Cluster 3: East-West leg, Nodes:47, 48, 49, 50.
- Cluster 4: East-West leg, Nodes:42, 43, 44, 45.
- Cluster 5: Center, Nodes:58, 57, 56, 59

In each node, the path of the agent in the cluster is setup in the *maf.conf* file of the respective cluster. The three targets specified above are run independently in the field in north-south and east-west direction. The mobile-agent is triggered manually in each node to check the collaborative classification result. The MAF based classification experiment is *successful* in identifying the target type. The results of each run are collected from the file *result.dat* written at the last node of each mobile-agent migration.

The experiments of MAF based classification in Twentynine Palms were successful. However, the MAF is implemented on a TCP layer using a wired network. For realistic WSN environment the MAF is required to be implemented in the wireless domain. This field demo proved the correct functionality of the MAF and helped us to proceed through the next phase of implementing MAF on Directed Diffusion using wireless RF modems.

3.3 BAE Austin Demo

This field demo is setup by BAE, at Austin, TX in August 2002. This demo is aimed at running collaborative classification and localization algorithms on civilian

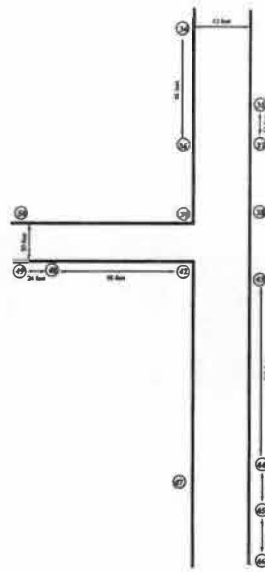


Figure 3.14: Node lay down at BAE Austin - T-Junction.

vehicles in urban environment. The participants of this demo include UT, Auburn University, Applied Research Laboratory/Pennsylvania State University, and BAE systems, Austin. BAE systems is the integrator for this demo providing the software and hardware support.

Two different node laydowns are used. In one the nodes are deployed along side the roads of a T-junction as shown in Fig. 3.14. In the other, perimeter security arrangement of the nodes in a parking lot is laid as shown in Fig. 3.15. Fig. 3.16 shows the node in a weather proof box at BAE, Austin. The testbed consists of 15 Sensoria nodes. All the field nodes are equipped with 10Mbps wired and 11Mbps wireless Ethernet to facilitate communication between nodes for loading/unloading of software, debugging, logging and for sensor data collection.

Four different vehicles as shown in Fig. 3.17 are used for classification and localization experiments. The targets primarily consist of:

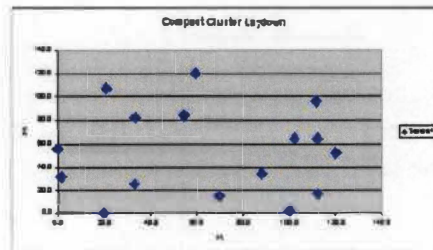


Figure 3.15: Node lay down at BAE Austin - Parking lot.

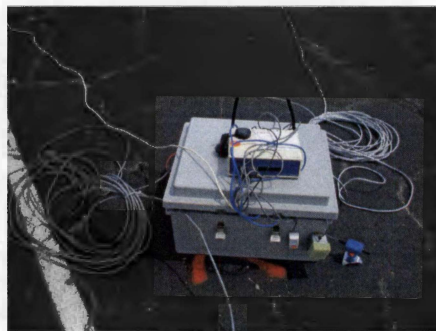


Figure 3.16: Node in weather proof box, BAE, Austin, TX.



(a) Pickup Truck



(b) Diesel Truck



(c) SUV



(d) Motorcycle

Figure 3.17: Vehicles used for classification.

- A heavy diesel truck with a distinctive seismic signature.
- A pickup truck with a distinctive seismic signature.
- A Harley-Davidson motorcycle with a distinctive acoustic signature.
- An SUV as a target of opportunity and discrimination tests.

There is a central server that is NFS mounted onto all the nodes. Loading software on this server's NFS directory makes available locally at each node. At each node the following elements are run at boot-up in the background:

- BAE low-level signal-processing.

- Directed Diffusion and gradient over RF modem.
- Logger and data acquisition (Only during data acquisition experiments).

Apart from these, UTK's classifier daemon and mobile agent daemon are run in the background for classification and MAF services respectively. The MAF in this demo is implemented on top of wireless Diffusion as discussed in Chapter 2. Upon event detection, the classification server triggers the mobile-agent automatically. The agent migrates within the cluster according to a pre-defined itinerary from the */tmp/maf.conf*. The four different clusters of the T-junction layout shown in Fig. 3.14 are:

- Cluster 1: Nodes - 34, 35, 36, 37
- Cluster 2: Nodes - 39, 38, 42, 43
- Cluster 3: Nodes - 46, 47, 45, 44
- Cluster 4: Nodes - 49, 50, 48

The different experiments conducted at the site and its results [32] are discussed below:

- Experiment 1: Mobile-agent based multi-modality multi-sensor (MAMMMS) classification on fixed itinerary in T-junction layout. The itinerary of the agent is setup in each cluster and all the daemons described above are run on each node. The vehicles are run from one end to other in the T-shaped road. The classification result from each cluster is observed confirming with the ground truth.

- Classification using one cluster of 4 nodes (*Success*).
- Classification using one cluster of 7 nodes (*Success*).
- Experiment 2: Mobile-agent based multi-sensor localization using a fixed itinerary. This experiment is only *partially Successful*. The agent migrated within nodes and integrated the localization result from some nodes but could not migrate further possibly due to some integration problem in the code. However the partial integration result proves the agent development to be viable for such kind of applications.

The MAF framework is *successfully* integrated with classification and localization applications using Diffusion over RF modems. This version of MAF is more robust and can be used in the true context of wireless sensor networks. The results from each node are collected from the log files located in the /tmp directory of each node.

The improvements from this version of MAF should go in the direction of adaptive itinerary of the agent since the current MAF uses a pre-defined, fixed itinerary.

3.4 BBN Waltham Demo

Unlike the previous experimental sites described above, this is a real-time live demonstration at DARPA PI meeting at BBN Technologies, Waltham in November 2002. This demonstration is aimed at presenting live MAF based classification and tracking of the targets using a graphical user interface (GUI). The participants in this demo include all the PIs funded through DARPA SensIT and other WSN based researchers and companies.



Figure 3.18: Node lay down at BBN, Waltham.

The setup consists of 27 nodes laid out across a road with sensors tied to observe the ground phenomenon. The layout of all the nodes is shown in Fig. 3.18. The radio network diagram is shown in the Fig. 3.19. The targets in this live demonstration primarily consists of:

- Walker through the sensor field.
- A passenger car.
- A pickup truck.

The BAE low-level signal-processing daemon and the ISI Diffusion are run in all the nodes in the background. During UTK's experiment the classifier daemon and the mobile agent daemon were also run in the background. All the nodes are provided with 10Mbps wired and 11Mbps wireless Ethernet for communication purposes. The radio connectivity of all the nodes is monitored using the radio

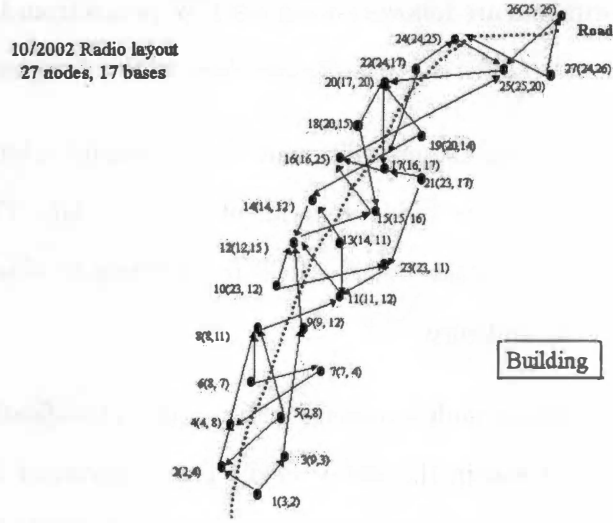


Figure 3.19: Radio configuration at BBN, Waltham.

heartbeat monitor provided by Sensoria. The sensor testbed is divided into 5 clusters namely:

- Cluster 1: Nodes - 1, 2, 3, 4, 5
- Cluster 2: Nodes - 7, 6, 8, 9, 10
- Cluster 3: Nodes - 13, 11, 12, 15, 14
- Cluster 4: Nodes - 21, 17, 16, 18, 19
- Cluster 5: Nodes - 20, 25, 22, 24, 27, 26

The MAF used in this demonstration is implemented over wireless diffusion. The *MAFserver* sends classification and partial integration result of the classification to the GUI using wired Ethernet on the nodes. This provides real-time display of the agent migration and classification task in partial/full. The real-time ground

truth of the experiments are followed using the CW radios from the target and the demonstrator. The two different experiments done in this live demonstration are:

- Mobile-agent based multi-modality multi-sensor classification of a walker walking in the sensor field. This experiment is *successful*. The walker icon is displayed progressing through the field in real time as observed on the GUI projected to the audience.
- Mobile-agent based multi-modality multi-sensor classification of a compact passenger car driven in the sensor field. This experiment is *successful*. The compact car icon is displayed progressing through the field in real time as observed on the GUI.

Chapter 4

Conclusion and Future Work

The wireless sensor network is a rapidly growing field and the diminishing size of the integrated circuits makes it a prospective research area in the coming years. These networks are inherently distributed and are tightly coupled with the physical surroundings through sensors attached to these nodes. The sensor nodes are rich in number however they are resource-constrained individually with limitations ranging from computing ability to battery power. This kind of environment will call for fundamental research in new kinds of paradigms for computing and collaborative signal and information processing. A research agenda for networked systems of embedded computers published by the National Research Council [19] describes the future of these kinds of networks which couple the physical world with the information space. They will virtually change all spheres of life through developments like swallowable health monitors and automated buildings. However this change of PCs to smart sensor nodes will require tailored analysis of their scalability and robustness.

This thesis concentrates on the development of a mobile-agent framework (MAF)

on top of a network routing layer for collaborative processing in wireless sensor networks, supporting applications like target classification, tracking and surveillance. The framework is flexible to support several different applications and can be configured accordingly. The framework provides several advantages and the implementation of this framework in WSN has been tested in three field demos. The advantages of the MAF are:

- The MAF requires less bandwidth compared to the peer client/server architecture since this framework does not transfer large amounts of data over the network.
- It conserves energy at the nodes since the total amount of data transmitted by the node is limited. Most of the nodal energy is consumed usually on the communication using radios. This in addition helps in prolonging the lifetime of the sensor network.
- The MAF provides progressive accuracy since the agent migrates from node to node carrying a partially integrated result and all the processing is done locally on each node.
- This kind of autonomous agent framework is reliable and can be fault-tolerant to unreliable network connectivity.
- The diffusion routing layer provides a unique data-dissemination paradigm for significant energy efficiency. Even with relatively unoptimized path selection, it outperforms an idealized traditional dissemination scheme like omniscient multicast [27]. With careful design of the radio MAC layer, the diffusion mechanisms are stable under a wide range of network dynamics.

- The data-centric dissemination of the diffusion along with the execution-code-based agent provide a unique data distribution mechanism with very less network traffic.

Future work is still needed in following aspects:

- The agent route is currently pre-defined. However the agent can be automated to choose its path on-the-fly based on the event detection and predicting the direction of target and adaptively migrating to the nodes in that direction. This requires the use of techniques like Doppler effect to initially estimate the direction of arrival (DoA).
- The clustering of the nodes can be automated. This will allow MAF to be highly scalable and robust. However, designing a scalable, distributed and robust clustering algorithm is a challenging problem.

Bibliography

Bibliography

- [1] Sensoria sgate wireless sensor gateway.
- [2] <http://webs.cs.berkeley.edu/tos/>.
- [3] <http://www.atmel.com/atmel/products/prod23.htm>.
- [4] <http://www.controlled.com/pc104/consp5.html>.
- [5] <http://www.controlled.com/pc104faq/#pc104>.
- [6] <http://www.isi.edu/scadds/pc104testbed>.
- [7] <http://www.kh-gps.de/nmea.faq>.
- [8] <http://www.python.org>.
- [9] <http://www.superh.com/products/sh4.htm>.
- [10] <http://www.swig.org>.
- [11] Winsng 2.0 user manual and api specification.
- [12] A.Fugetta, G.P.Picco, and G.Vigna. Understanding code mobility. *IEEE Transactions on Software Engineering*, 24(5):342–361, 1998.

- [13] Wilmer Carpie, George Cybenko, Katsuhiro Moizumi, and Robert Gary. Network awareness and mobile agent systems. *IEEE Communication Magazine*, 1998.
- [14] Alberto Cerpa, Jeremy Elson, Deborah Estrin, Lewis Girod, Michael Hamilton, and Jerry Zhao. Habitat monitoring: Application driver for wireless communications technology. *First ACM SIGCOMM Workshop on Data Communications in Latin America and the Caribbean*, 2001.
- [15] C.Perkins. *Ad Hoc Networks*. Addison-Wesley, Reading, MA, 2000.
- [16] D.Clarke and D.Tennenhouse. Architectural considerations for a new generation of protocols. *Proceedings of ACM Symposium on Communications Architectures and Protocols*, pages 200–208, 1990.
- [17] Deborah Estrin, Ramesh Govindan, John Heidemann, and Satish Kumar. Next century challenges: Scalable coordination in sensor networks.
- [18] Deborah Estrin, Akbar Sayeed, and Mani Srivastava. Mobicom 2002 tutorial: Wireless sensor networks. *MOBICOM*, 2002.
- [19] Deborah Estrin et al. Embedded everywhere, a research agenda for networked systems of embedded computers. *Computer Science and Telecommunications Board, National Research Council*, 2001.
- [20] Leonidas J. Guibas. Sensing, tracking, and reasoning with relations.
- [21] Colin G. Harrison, David M. Chess, and Aaon Kershenbaum. Mobile agents: Are they a good idea. *IBM Research Division*, March 13, 1995.

- [22] Wendi Beth Heinzelman. Application-specific protocol architectures for wireless sensor networks. *Ph.D Thesis*, Massachusetts Institute of Technology.
- [23] Wendi R. Heinzelman, Joanna Kulik, and Hari Balakrishnan. Adaptive protocols for information dissemination in wireless sensor networks. In *Fifth ACM/IEEE MobiCom*, pages 1–12, Seattle, WA, August 1999. ACM/IEEE.
- [24] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister. System architecture directions for networked sensors.
- [25] Ian.F.Akyildiz, Y.Sankarasubramanian, and E.Cayirci. Wireless sensor networks: A survey.
- [26] Research Challenges in Wireless Networks of Biomedical Sensors. Loren schwiebert and sandeep k.s. gupta and jennifer weinmann. *The seventh annual international conference on Mobile computing and networking*, pages 151–165, 2001.
- [27] Chalermek Intanagonwiwat, Ramesh Govindan, and Deborah Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks.
- [28] J. Kulik, R. B. Heinzelman, and H. Balakrishnan. Negotiation-based protocols for disseminating information in wireless sensor networks. *Submitted to ACM Wireless Networks*, 2000.
- [29] Mark Lutz. *Programming Python*. O'Reily, 1996.
- [30] M.Abbott and L.Peterson. Increasing network throughput by integrating protocol layers. *IEEE/ACM Transactions on Networking*, 1(5):600–610, 1993.

- [31] Rex Min, Manish Bharadwaj, Seong-Hwan Cho, Eugene Shih, Amit Sinha, Alice Wang, and Anantha Chandrakasan. Low-power wireless sensor networks.
- [32] University of Tennessee, Auburn University, and BAE SYSTEMS Austin. Sensit collaborative processing experiments. *BAE-Austin Test Site*, August 19-21, 2002.
- [33] G.J. Pottie and W.J. Kaiser. Wireless integrated network sensors. *Communications of the ACM*, 43:551–8, 2000.
- [34] Hairong Qi, S.S.Iyengar, and K.Chakrabarty. Multi-resolution data integration using mobile agents in distributed sensor networks. *IEEE Trans. on Syst., Man, and Cybern. Part C:Applications and Reviews*, 31(3):383–391, 2001.
- [35] Fabio Silva, John Heidemann, and Ramesh Govindan. Network routing application programmer’s interface (api) and walk through 9.0.1.
- [36] Mani Srivastava, Richard Muntz, and Miodrag Potkonjak. Smart kindergarten: Sensor-based wireless networks for smart developmental problem-solving environments. *The seventh annual international conference on Mobile computing and networking 2001*, pages 132–138, 2001.
- [37] Todd Sundsted. An introduction to agents. *Java world*, 1998.
- [38] BBN Technologies. Sensit experimental plan (sitex02). *Version 1.0*, 2001.
- [39] Y. Tian and H. Qi. Target detection and classification using seismic signal processing in unattended ground sensor systems. *International Conference on Acoustics Speech and Signal Processing (ICASSP)*, 2002.

- [40] Jeffrey T. Russell and Margarida F. Jacome. Software power estimation and optimization for high performance of 32-bit embedded processors. *Proceedings of ICCD'98, Austin, Texas*, 1998.
- [41] X. Wang and H. Qi. Acoustic target classification using distributed sensor arrays. *International Conference on Acoustics Speech and Signal Processing (ICASSP)*, 2002.
- [42] X. Wang, H. Qi, and S. S. Iyengar. Collaborative multi-modality target classification in distributed sensor networks. *Information Fusion*, 2002.
- [43] X. Wang, H. Qi, and S. S. Iyengar. Collaborative multi-modality target classification in distributed sensor networks. *Information Fusion*, 2002.
- [44] Jay Warrior. Smart sensor networks of the future. *Sensors Magazine*, 1997.
- [45] Yingue Xu and Hairong Qi. Performance evaluation of distributed computing paradigms in mobile ad-hoc sensor networks. *ICPADS*, 2002.
- [46] Yingue Xu, Hairong Qi, and Phani Teja Kuruganti. Computing paradigms for collaborative processing in sensor networks. *Globecom*, 2003.
- [47] X. Wang and H. Qi. Acoustic target classification using distributed sensor arrays. *ICASSP*, 2002.
- [48] Y. Tian and H. Qi. Target detection and classification using seismic signals processing in unattended ground sensor systems. *ICASSP*, 2002.
- [49] Feng Zhao, Jaewon Shin, and James Reich. Information-driven dynamic sensor collaboration for tracking applications. *IEEE Signal Processing Magazine*, 2002.

Appendix

Appendix A

The MAF can be downloaded from the Advanced Imaging and Collaborative Information Processing Lab (AICIP), UTK website (<http://aicip.ece.utk.edu>). The latest version is MAF-1.3 which includes the mobile agent framework, low-level signal processing and graphical user interface (GUI) for visualization. This appendix explains installing and running of the MAF framework on Directed Diffusion.

A.1 Dependencies to Build MAF

1. **Cross-Compilers:** Cross-compiling for the Sensoria nodes requires the newest SH4 tools. In order to use the Intel based Linux machine to generate binaries for the Hitachi SH4 processor based sensor nodes, you need to install the cross-compilers to compile the code. These cross-compilers can be obtained from the AICIP's MAF web page:

<http://aicip.ece.utk.edu/research/mufashion.htm>

The RPM available there installs the tools in the `/usr/sh4-linux` directory. The complete set of compilers and the required header files can be found in

this directory. Add the `/usr/sh4-linux/bin` directory to your path:

```
export PATH=/usr/sh4-linux/bin:$PATH$
```

To build executables for the SH4 target running Linux, simply replace the build tools environment variables used in your makefile (i.e. `CC`, `AS`, `LD`, `RANLIB`, etc.) with their SH4 version. For example, replace “make all” with: “make `CC=sh4-linux LD=sh4-linux-ld AS=sh4-linux-as` all”, or you can directly change the compilers inside the Makefile accordingly. See `/usr/sh4-linux/bin` for a complete list of the SH4 cross compiler tools. For a complete set of RPMs for SH4 processor based Linux visit the RPM repository at :

<http://www.sh-linux.org/rpm-index/index.html>

2. **Python Interpreter on SH4:** The Hitachi SH4 processor compatible Python interpreter binary should be generated to run Python code on the sensor node. The Python-1.5.2 for SH4 can be obtained from the AICIP's MAF web page:

<http://aicip.ece.utk.edu/research/mufashion.htm>

Alternatively you can generate the executable yourself by downloading Python from “<http://www.python.org>” and making the following modifications to generate the Python - 1.5.2 on Hitachi SuperH4 processor:

- (a) Obtain and install cross-compilers and check if they are available from the command line i.e the path is set correctly.
- (b) Obtain the source of Python 1.5.2 from <http://www.python.org>.
- (c) From the terminal in which the cross compiling of the source code is done, create the following variables in the environment as follows:


```
CC=sh4-linux-gcc
export CC
AR=sh4-linux-ar
export AR
RANLIB=sh4-linux-ranlib
export RANLIB
GCC=sh4-linux-gcc
export GCC
LD=sh4-linux-ld
export LD
AS=sh4-linux-as
export AS
```

- (d) Edit the "configure.in" file to generate the appropriate configure script and Makefile. Remove or modify lines in the file "configure.in" which are irrelevant for the sh4 processor and also some "try - compile" and "run - test" kind of code. The changes described in the rest of the section should be done on the "configure.in" file
- (e) Comment all the lines following the "NEXTSTEP stuff" until the end of the loop. This removes the machine dependency of the generated configure script
- (f) Comment all the lines in the section starting with the following line and till the end of the loop.

```
# checks for UNIX variants that set C preprocessor variables
```

- (g) Comment all the lines starting with AC_CHECK_SIZEOF and replace

the following lines. These lines define the size of the data types that are defined for the platform.

```
AC_CHECK_SIZEOF(int,4)
AC_CHECK_SIZEOF(long,4)
AC_CHECK_SIZEOF(void *,4)
AC_CHECK_SIZEOF(char,1)
AC_CHECK_SIZEOF(short,2)
AC_CHECK_SIZEOF(float,4)
AC_CHECK_SIZEOF(double,8)
AC_CHECK_SIZEOF(long long,8)
```

(h) Comment on all the following lines of the code:

```
# Hmph. AC_CHECK_SIZEOF() doesn't include <sys/types.h>.
.....
.....
# AC_MSG_RESULT(no)
#fi
```

(i) In the section of setting compiler characteristics set

```
bad_forward = no
bad_prototypes = no
and comment on the following lines:
if test "$have_prototypes" = yes; then
bad_prototypes=no
AC_MSG_CHECKING(for bad exec* prototypes)
#AC_TRY_COMPILE([#include <unistd.h>], [char **t;execve("\0",t,t);], ,
```

```

#AC_DEFINE(BAD_EXEC_PROTOTYPES) bad_prototypes=yes)
#AC_MSG_RESULT($bad_prototypes)
fi

bad_forward=no
#AC_MSG_CHECKING(for bad static forward)
#AC_TRY_RUN([
#struct s { int a; int b; };
#static struct s foo;
#int foobar() {
# static int random;
# random = (int) &foo;
# return random;
#}
#static struct s foo = { 1, 2 };
#main() {
# exit(!((int)&foo == foobar()));
#}
#], , AC_DEFINE(BAD_STATIC_FORWARD) bad_forward=yes)
#AC_MSG_RESULT($bad_forward)

```

(j) Comment on the following lines starting with:

```

# check whether malloc(0) returns NULL or not
.....
.....
# AC_DEFINE(MALLOC_ZERO_RETURNS_NULL)
#fi

```

- (k) After all the modification, save the “configure.in” and from the same directory and terminal where you have exported the variable, generate the “configure” script using the command:

```
#bash: autoconf
```

- (l) Run the configure file from the current directory by “./configure --with-thread” or just “./configure” if thread support is not required.
- (m) In the Makefile generated do the following modifications:

```
VERSION=1.5  
srcdir= .  
CC=sh4-linux-gcc -m4  
AR=sh4-linux-ar  
RANLIB=sh4-linux-ranlib -m4  
DEFS=-DHAVE_CONFIG_H
```

- (n) After all the modifications run the make command and it should generate the Python binary required for the SH4.

3. Simplified Wrapper and Interface Generator (SWIG): The MAF uses SWIG to generate shared libraries to be accessible from different languages. Download SWIG from <http://www.swig.org/> and install it. Make sure the swig compiler is accessible from the command line.

4. Directed Diffusion: The MAF uses ISI west’s Directed Diffusion for network routing. MAF-1.3 uses Diffusion-3.1 and can be obtained from <http://www.isi.edu/scadds/software/>

The Diffusion home page can be found at

<http://www.isi.edu/scadds/projects/diffusion.html>

The Directed Diffusion can be compiled for Intel-based systems or Hitachi based WINSNG systems. Follow the Diffusion README file to generate diffusion for Hitachi SH4 processor based WINSNG nodes.

5. **Graphical User Interface (GUI)** The GUI uses the Java for display. Download the latest version of Java from Sun and install the path to be accessible from the command line.

A.2 Compilation of MAF

After setting up all the dependencies download the latest release MAF-1.3 from the AICIP lab's MAF web page:

<http://aicip.ece.utk.edu/research/mufashion.htm>

Untar the MAF-1.3 into the current directory. The directory structure is explained in the README file. Open the "config.mk" file and make the required modifications as follows:

- Set the compilers to the platform you use (default is SH4).
- Specify the complete diffusion-3.1 directory as "diff_dir". Do not specify a relative path.

```
diff_dir = /mnt/piranha/pkurugan/package/diffusion-3.1
```

- Specify the MAF_DIR, the directory where you have untarred this software. Give the complete path.

MAF_DIR = /mnt/piranha/pkurugan/package/MAF-1.2

- At the `mafstart_dir` specify the complete path to the subdirectory `maf` in MAF-1.x. This specifies the classifier the path from which the agent should be executed on the node.
- At the `python_exec` directory specify the complete path to the Python interpreter on the node.

Type "make" in the MAF-1.x/ directory to compile all the code.

The `utkclassifier/` directory consists of the low-level signal processing code and classification code. To compile change to MAF-1.3/`utkclassifier` directory and type `./makeall.sh`. This generates the detector-classifier "utkClassify" and copies the binary to `bin` directory (the required *.dat files are read from /tmp directory so when running, copy the dat files from `utkclassifier/` dir to /tmp of node).

A.3 Running the MAF

1. Setup the radio configuration on each node and setup the appropriate network numbers with corresponding bases and remotes. For example, to setup a radio 0 of a node on network 10 as remote on the terminal to the node run the following command

```
echo remote:network=10 > /dev/rf/0/command
```

Similarly, to setup as base change the "remote" as "base". There can be two remotes on one node but there cannot be more than one base on a node.

2. Run diffusion and gradient on each node.
3. Run "utkClassify" which does both detection and classification. Run "utk-Classify -server" on the leader node (that initiates the mobile agent upon event detection) and "utkClassify" on the rest of the nodes. When running the server take care of the system command in `utkclassifier/utk/UTKClassifierServer.cpp` (this is responsible for calling the agent and Python).
4. Set the mobile agent itinerary in the `maf.conf`. This determines the itinerary of the mobile agent with a list of IP addresses of the nodes the mobile agent will migrate. Store this file in the `/tmp` directory of the node. Typical `maf.conf` looks as follows:

```
[itinerary]
hops=3
node0=sensit3
node1=sensit1
node2=sensit2
```

```
[parameter]
np=3
res=0.05
thresh=0.8
```

5. Start the Mobile Agent daemon at all the nodes listed in the configuration file.

For example, to run `cd` into the `MAF-1.x/maf` directory and type

```
bash# /MAF-1.2/Python/python Cmafsvr.py
```

you will see a server waiting message. “ /MAF-1.2/Python/python” is the directory where you have the Python interpreter for SH4 platform.

6. The "utkClassify -server" automatically sends the mobile agent to the group of nodes specified in the maf.conf. Copy the same maf.conf in all the group of nodes.
7. The final integration is written in the result.dat file after the agent returns back to the initial node.
8. Finally the classification output can be seen on the GUI available. Each node currently sends the information to the GUI using the Ethernet. The IP address of the machine where the GUI is running should be setup in the CC-SIPAgent.py (line 142) as a variable passed on to the sndTCPmsg function. Edit this file to reflect the respective changes. To bring up the GUI change to the gui/ directory and run "make install" (make sure java is properly installed at the command prompt). Click on the listen button to start listening, the status is shown on the status bar below.
9. To setup a particular image file as background for the GUI, obtain the corresponding pixel positions of the sensor nodes in the image and write them in the sensorpositions.txt file in gui/ directory.

Vita

Phani Teja, Kuruganti was born in Eluru, India in 1980. His major interests and research areas are signal processing, digital system design and networking. He graduated from Chaitanya Bharathi Institute of Technology, Osmania University, India in 2001 with a Bachelor of Engineering degree in Electronics and Communication Engineering. During his undergraduate study, he implemented a 1024-point Fast Fourier Transform using VHDL to carry out FFT analysis in FPGA for in-flight data compression (vibration and acoustic data). This work is done at Defence Research Development Laboratories, Hyderabad, India. In Fall 2001 he came to The University of Tennessee at Knoxville as a graduate student in Electrical and Computer Engineering. He then joined the Advanced Imaging and Collaborative Information Processing lab of Dr. Hairong Qi with a major topic on wireless sensor networks. He developed mobile agent based computing paradigm on top of ISI Diffusion routing layer for collaborative signal and information processing tasks in wireless sensor networks which has been successfully implemented in three field demos. Apart from this he also worked on efficient algorithms for early detection of breast cancer using non-invasive infrared thermography. Phani Teja's Publications include:

Yingue Xu, H. Qi, Phani Teja Kuruganti, " Computing paradigms for collaborative processing in sensor networks ", accepted by GLOBECOM 2003.

H. Qi, Phani Teja Kuruganti, Yingue Xu, " The development of localized algorithms in wireless sensor networks ", Sensors Journal, 2(7): 270-285, July 2002.

H. Qi, Phani Teja Kuruganti, Z. Liu, " Early detection of breast cancer using thermal texture maps ", IEEE International Symposium on Biomedical Imaging:

Macro to Nano, Washington, D.C., July, 2002.

Phani Teja Kuruganti, H. Qi, " Asymmetry analysis in breast cancer detection using thermal infrared images ", IEEE EMBS'02, Houston, 2002.