BioSENSE: Biologically-inspired Secure Elastic Networked Sensor Environment

Ramy M. Eltarras

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Computer Engineering

Mohamed Eltoweissy, Chair
Scott Midkiff
Luiz DaSilva
Ing-Ray Chen
Sedki Riad
Moustafa Youssef

Jan 25th, 2011
Blacksburg, Virginia

Keywords: Sensor Networks, Biologically-Inspired Design, Routing, Middleware, Software Diversity, Adaptability, Resilient Architecture

BioSENSE: Biologically-inspired Secure Elastic Networked Sensor Environment

Ramy M. Eltarras

ABSTRACT

The essence of smart pervasive Cyber-Physical Environments (CPEs) is to enhance the dependability, security and efficiency of their encompassing systems and infrastructures and their services. In CPEs, interactive information resources are integrated and coordinated with physical resources to better serve human users. To bridge the interaction gap between users and the physical environment, a CPE is instrumented with a large number of small devices, called sensors, that are capable of sensing, computing and communicating. Sensors with heterogeneous capabilities should autonomously organize on-demand and interact to furnish real-time, high fidelity information serving a wide variety of user applications with dynamic and evolving requirements. CPEs with their associated networked sensors promise aware services for smart systems and infrastructures with the potential to improve the quality of numerous application domains, in particular mission-critical infrastructure domains. Examples include healthcare, environment protection, transportation, energy, homeland security, and national defense.

To build smart CPEs, Networked Sensor Environments (NSEs) are needed to manage demand-driven sharing of large-scale federated heterogeneous resources among multiple applications and users. We informally define NSE as a tailorable, application agnostic, distributed platform with the purpose of managing a massive number of federated resources with heterogeneous computing, communication, and monitoring capabilities. We perceive the need to develop scalable, trustworthy, cost-effective NSEs. A NSE should be endowed with dynamic and adaptable computing and communication services capable of efficiently running diverse applications with evolving QoS requirements on top of federated distributed resources. NSEs should also enable the development of applications independent of the underlying system and device concerns. To our knowledge, a NSE with the aforementioned capabilities does not currently exist.

The large scale of NSEs, the heterogeneous node capabilities, the highly dynamic topology, and the likelihood of being deployed in inhospitable environments pose formidable challenges for the construction of resilient shared NSE platforms. Additionally, nodes in NSE are often resource challenged and therefore trustworthy node cooperation is required to provide useful services. Furthermore, the failure of NSE nodes due to malicious or non-malicious conditions represents a major threat to the trustworthiness of NSEs. Applications should be able to survive failure of nodes and change their runtime structure while preserving their operational integrity. It is also worth noting that the decoupling of application programming concerns from system and device concerns has not received the appropriate attention in most existing wireless sensor network platforms.

In this dissertation, we present a Biologically-inspired Secure Elastic Networked Sensor En-

vironment (BioSENSE) that synergistically integrates: (1) a novel bio-inspired construction of adaptable system building components, (2) associative routing framework with extensible adaptable criteria-based addressing of resources, and (3) management of multi-dimensional software diversity and trust-based variant hot shuffling. The outcome is that an application using BioSENSE is able to allocate, at runtime, a dynamic taskforce, running over a federated resource pool that would satisfy its evolving mission requirements. BioSENSE perceives both applications and the NSE itself to be elastic, and allows them to grow or shrink based upon needs and conditions.

BioSENSE adopts Cell-Oriented-Architecture (COA), a novel architecture that supports the development, deployment, execution, maintenance, and evolution of NSE software. COA employs mission-oriented application design and inline code distribution to enable adaptability, dynamic re-tasking, and re-programmability. The *cell*, the basic building block in COA, is the abstraction of a mission-oriented autonomously active resource. Generic cells are spontaneously created by the middleware, then participate in emerging tasks through a process called *specialization*. Once specialized, *cells* exhibit application specific behavior. Specialized *cells* have *mission objective* that are being continuously sought, and *sensors* that are used to monitor performance parameters, mission objectives, and other phenomena of interest.

Due to the inherent anonymous nature of sensor nodes, associative routing enables dynamic semantically-rich descriptive identification of NSE resources. As such, associative routing presents a clear departure from most current network addressing schemes. Associative routing combines resource discovery and path discovery into a single coherent role, leading to significant reduction in traffic load and communication latency without any loss of generality. We also propose Adaptive Multi-Criteria Routing (AMCR) protocol as a realization of associative routing for NSEs. AMCR exploits application-specific message semantics, represented as generic criteria, and adapts its operation according to observed traffic patterns.

BioSENSE intrinsically exploits software diversity, runtime implementation shuffling, and fault recovery to achieve security and resilience required for mission-critical NSEs. BioSENSE makes NSE software a resilient moving target that : 1) confuses the attacker by non-determinism through shuffling of software component implementations; 2) improves the availability of NSE by providing means to gracefully recover from implementation flaws at runtime; and 3) enhances the software system by survival of the fittest through trust-based component selection in an online software component marketplace.

In summary, BioSENSE touts the following advantages: (1) on-demand, online distribution and adaptive allocation of services and physical resources shared among multiple long-lived applications with dynamic missions and quality of service requirements, (2) structural, functional, and performance adaptation to dynamic network scales, contexts and topologies, (3) moving target defense of system software, and (4) autonomic failure recovery.

This work is sponsored in part by NSF award 0721523.

Dedication

To my wonderful parents,
for their encouragement, wisdom and guidance,
To my loving wife,
for her endless support and patience,
To Jana and Hussein
for shining my days with their smiles,
To the souls of the martyrs of the Egyptian revolution,
for the freedom and dignity they gave to my home country,
With Love and devotion
I dedicate to thee.

Acknowledgments

Finishing this dissertation has been a long journey, but not one that I did alone. I would like to express my gratitude and appreciation to all those people who helped, inspired, and supported me along the way.

First and foremost, I would like to thank Dr. Mohamed Eltoweissy for his exceptional mentorship and guidance. I consider myself very fortunate to have had Dr. Eltoweissy as my advisor. I have immensely gained from his research experiences, insightful viewpoints, and inspiring ideas. Our scientific discussions were mind openers to me. He has greatly impacted my research skills and enlightened my perspectives. I would also like to thank him for his belief in my potential and continuous support and encouragement.

I would like to thank my committee members, Dr. Scott Midkiff, Dr. Luiz DaSilva, Dr. Ing-Ray Chen, Dr. Sedki Riad, and Dr. Moustafa Youssef, for reviewing my manuscript and providing me with their excellent feedback and valuable remarks that greatly helped me to improve the quality of this dissertation. I appreciate the time and effort they dedicated to me, in spite of their busy schedule.

Pursuing a PhD degree from a respectable university like Virginia Tech has always been a dream to me. This dream could have never come true without the efforts of Dr. Sedki Riad and Dr. Yasser Hanafy in establishing the VT-MENA program. I would like to thank them for making this program successful and giving me the chance to enjoy Virginia Tech education and research environment from Egypt.

Last, but not least, I would also like to extend my deepest gratitude to my family, colleagues and friends for the encouragement that I have received during this enriching journey. Special thanks to my parents, wonderful wife, and children for their continuous support and bearing with me throughout this journey.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

"Research is to see what everybody else has seen, and to think what nobody else has thought" .. Albert Szent-Gyorgi

1.1 Overview

The essence of smart pervasive Cyber-Physical Environments (CPEs) is to enhance the dependability, security and efficiency of their encompassing systems and infrastructures and their services[42][73]. In CPEs, interactive information resources are gracefully integrated and coordinated with physical resources to better serve human users. To bridge the interaction gap between users and the physical environment, a CPE is instrumented with a large number of small devices, called sensor nodes (or sensors, for short) that are capable of sensing, computing and communicating. Sensors should autonomously organize to furnish real-time, high-fidelity information to their users. CPEs with their associated networked sensors promise aware services for smart systems and infrastructures with the potential to improve the quality of numerous application domains, in particular mission-critical domains ranging from healthcare and transportation to national defense, law enforcement and public safety. These application require very large number of networked sensors, imposing serious challenges on all aspects of network and software management beyond the capabilities of contemporary Wireless Sensor Networks (WSN)[50].

We define NSE as a tailorable, application agnostic, distributed platform managing a massive number of resources with heterogeneous computing, communication, and monitoring capabilities. A major differentiator between NSEs and contemporary WSNs is that NSEs

Table 1.1: NSE vs WSN

WSN	NSE
Special purpose deployment	General purpose deployment
Programmed before deployment in field	Programmed on-demand after deployment in field
All nodes run the same program image	Different nodes may have different different programs to run
No or limited re-programmability	Programmability is a first class feature
Owned and controlled by a single administrative entity	Shared ownership and federated administrative control
Single user (usually same as owner), and single mission	Multiple users (usually different from owners), and multiple missions
Networking is usually coupled with application	General purpose networking decoupled from applications

are programmed and organized on-demand according to dynamic evolving needs and situational requirements, while WSNs are pre-programmed and deployed for specific application or mission. Current WSN architectures are based on the assumption that all sensor nodes are participating in a single global task[74]. Table 1.1 summarizes the major differences between our vision for NSE and contemporary WSN. Although the literature contains a number of research efforts that address some of the limitations of existing WSN, these efforts are point solutions that addresses specific WSN limitations. For example, a generic communication structure for WSN was proposed in [33], and the scoping concept was proposed to enable multi-purpose WSN in [74]. More details are provided in chapter 2.

NSEs are intended to bridge computation and physical processes within CPEs and provide in-situ users with diverse services from multiple providers while preserving their changing QoS requirements. Due to the very limited computational and communications capabilities of individual sensor nodes, NSEs rely on the collaboration among large number of sensor nodes to perform desired functions and achieve its goals. It is very common to have a large number of sensors that are completely identical in hardware or play the same role. Also, the network resource management and maintenance functions becomes more costly as number of nodes in the networks increases if nodes are individually addressed. This makes the unique identification of individual sensor node both unnecessary and undesirable in large scale NSEs.

Changes in the network's underlying resources, connectivity, mission, or QoS requirements necessitate the design of autonomous adaptable NSE architectures and protocols. The network should be able to change its structure and behavior at run-time in response to change in requirements or environment. The ability to autonomously adapt to situational changes is essential specially in hazardous and unattended environment where human physical access to the network may not be feasible.

BioSENSE attempts to realize a scalable, trustworthy, secure, flexible, cost-effective and dynamic computing and communication infrastructure capable of efficiently running multiple applications, designed by different entities for different purposes, on top of federated NSE resources. The outcome is that an application using BioSENSE is able to allocate, at runtime, a dynamic taskforce, running over a federated resource pool that would satisfy its evolving mission requirements. BioSENSE allows sharing of federated resource pools among multiple applications. BioSENSE exploits application semantics and adapts to observed workload to enhance the efficiency of NSEs. BioSENSE perceives both applications and the NSE itself

to be elastic, and allows them to grow or shrink based on needs and conditions.

BioSENSE adopts Cell-Oriented-Architecture (COA), a novel biologically-inspired software architecture that supports the development, deployment, execution, maintenance, and evolution of NSE software. COA employs mission-oriented application design and inline code distribution to enable adaptability, dynamic re-tasking, and re-programmability. COA provides ability to seamlessly add new resources to the existing network. BioSENSE provide the programming framework and middleware to support COA based NSE application.

The *cell*, the basic building block in COA, is the abstraction of a mission-oriented autonomously active resource. Cells undergo a life-cycle that starts with the encapsulation of passive resources in stem cells. Stem cells are unspecialized cell that exhibit common application-agnostic behavior that enables to participate in emerging tasks through a process called *specialization*. The *specialization* process automatically and transparently triggers *code discovery* process to locate and download the code modules that implements the functionality and behavior of the desired cell type. BioSENSE middleware provides in-network storage of NSE code to allow fast and efficient inline code distribution. Once specialized, *cells* exhibit application specific behavior. A number of cells specialized cells can be organized to compose an *organism* that can take different shapes and sizes according to need. Specialized *cells* have *mission objective* that are being continuously sought, and *sensors* that are used to monitor performance parameters, mission objectives, and other phenomena of interest. Specialized cells can be re-specialized into a different *cell* type during its lifetime. Cells die when the physical sensor nodes that hosts them die. When new NSE nodes are added they spontaneously form new stem cells. BioSENSE provides an application development framework based on COA. The design elements and aspects of BioSENSE applications are shown in figure 3.1.

We propose associative routing as a class of routing protocols that enables dynamic semantically-rich descriptive identification of network resources and services. As such, associative routing presents a clear departure from most current network addressing schemes, eliminating the need for a separate phase of resource/service discovery. We hypothesize that since, in ad-hoc computing environments, resource discovery operates similarly to path discovery then both can be performed in a single phase, leading to significant reduction in traffic load and communication latency without any loss of generality. We also propose a framework for associative routing and present Adaptive Multi-Criteria Routing (AMCR) protocol as a realization of associative routing for sensor networks. AMCR exploits application-specific message semantics, represented as generic criteria, and adapts its operation according to observed traffic patterns. Analytical results demonstrate the scalability and efficiency of AMCR.

The development of secure, robust NSE software is highly challenged by implementation flaws that could lead to serious security vulnerabilities despite careful auditing and analysis of the software specifications. The implementation flaws are usually difficult to detect and fix before the system is operational due to the prohibitive cost of auditing the implementation or testing all possible inputs and system states, specially in distributed ad-hoc environments.

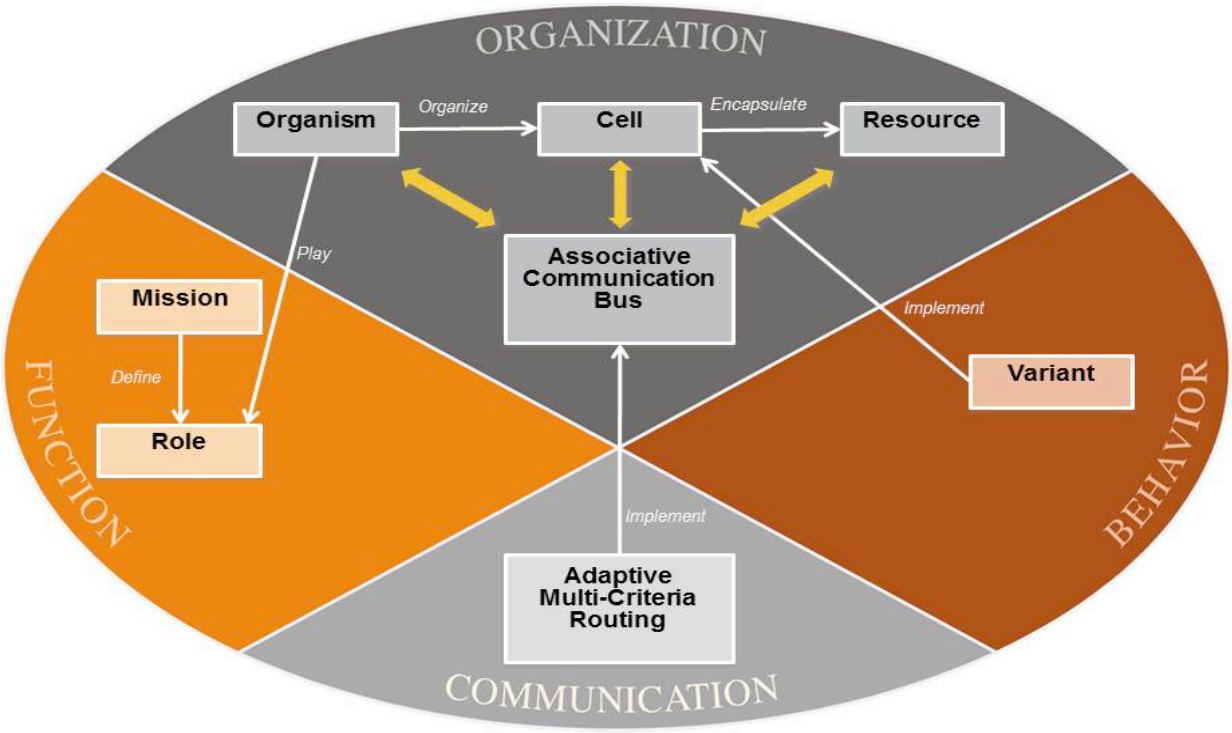

Figure 1.1: BioSENSE Design Elements

Anonymous intruders could exploit implementation flaws to gain partial or total control of the NSE, cause the software to crash leading to undesirable malfunction, or disclose private or sensitive information. To address this class of vulnerably, BioSENSE intrinsically exploits software diversity[57], runtime implementation shuffling, and fault recovery to achieve security and dependability in mission critical NSEs. BioSENSE decouples software specifications from system implementations and performs "hot" shuffle among a multiplicity of diverse implementations for the same specifications. BioSENSE is able to tolerate the existence of implementation flaws, while considerably enhancing the security and robustness of NSEs. BioSENSE makes NSE software a resilient moving target that : 1) confuses the attacker by non-determinism through shuffling of software component implementations; 2) improves the availability of NSE by providing means to gracefully recover from implementation flaws at runtime; and 3) enhances the software system by survival of the fittest through trust-based component selection in an online software component marketplace.

1.2 Smart Border Protection: A Scenario

Clive is a U.S. border patrol officer who leads a team responsible for mitigating illegal border crossings, trafficking, smuggling or other similar criminal activities. The team deploys large array of networked sensors such as electro-optical cameras, infrared cameras, vibration sensors and explosives detection sensors that aids the detection of these illegal activities. The team uses High Mobility Multipurpose Vehicles (HMMWVs) to respond to the detected illegal activities. Each HMMWV is equipped with high performance computers and high power communication devices. The team also deploys Unmanned Ground Vehicles (UGVs) to facilitate the mission.

Though the sensor network is of great value in the detection phase, the single-purpose nature of the sensor network limits the usefulness of the sensor network during the pursuit-and-x1 mission. That is, the sensor network can only provide very little help in prioritizing the responses to multiple concurrent attacks.

Problem Space

Multiple criminal activities can happen in a relatively short time at the border. In this scenario, a group of foreigners tries to cross the border without proper identification, and soon after a smaller group of armed drug dealers also attempt to cross the border. The sensor network detects both illegal intrusions and notifies Clive the size of the groups. However, the group of drug dealers is able to use a sensor-key extractor to compromise a sensor node. The drug dealers then carry the compromised node and instruct it to advertise itself as a local sink and then drop all traversing multihop packets, thereby obfuscating the exact location of the group of drug dealers.

Clive dispatches his team in the HMMWVs in a pursuit-and-capture mission, aiming to capture the larger group based on the information provided by the sensor network as shown in figure 1.2. Clive also dispatches a number of UGVs to aid the team's mission. During the mission, Clive's team spends great effort in contacting and collaborating with the UGVs as the UGVs move in and out of the range of the HMMWVs. Clive's team was able to successfully capture the large group of illegal travelers; however, the drug dealers were able to deceive the sensor network and escape. Additionally, during the escape, the group of drug dealers may be able to compromise a large number of sensors. Thus, Clive's team may not be able to rely on the sensor network to effectively detect future illegal activities.

If the sensor network can autonomously switch to using path redundancy, then it could mitigate the described local-sink attack when suspicion is raised. Additionally, if the sensor network can incorporate the additional computation ability of the HMMWVs and the UGVs, then the network may change its sensing task from mere intrusion detection to more sophisticated target tracking and activity identification. The sensor network, on the other hand, can provide navigation support for the HMMWVs and the UGVs to minimize the time needed to

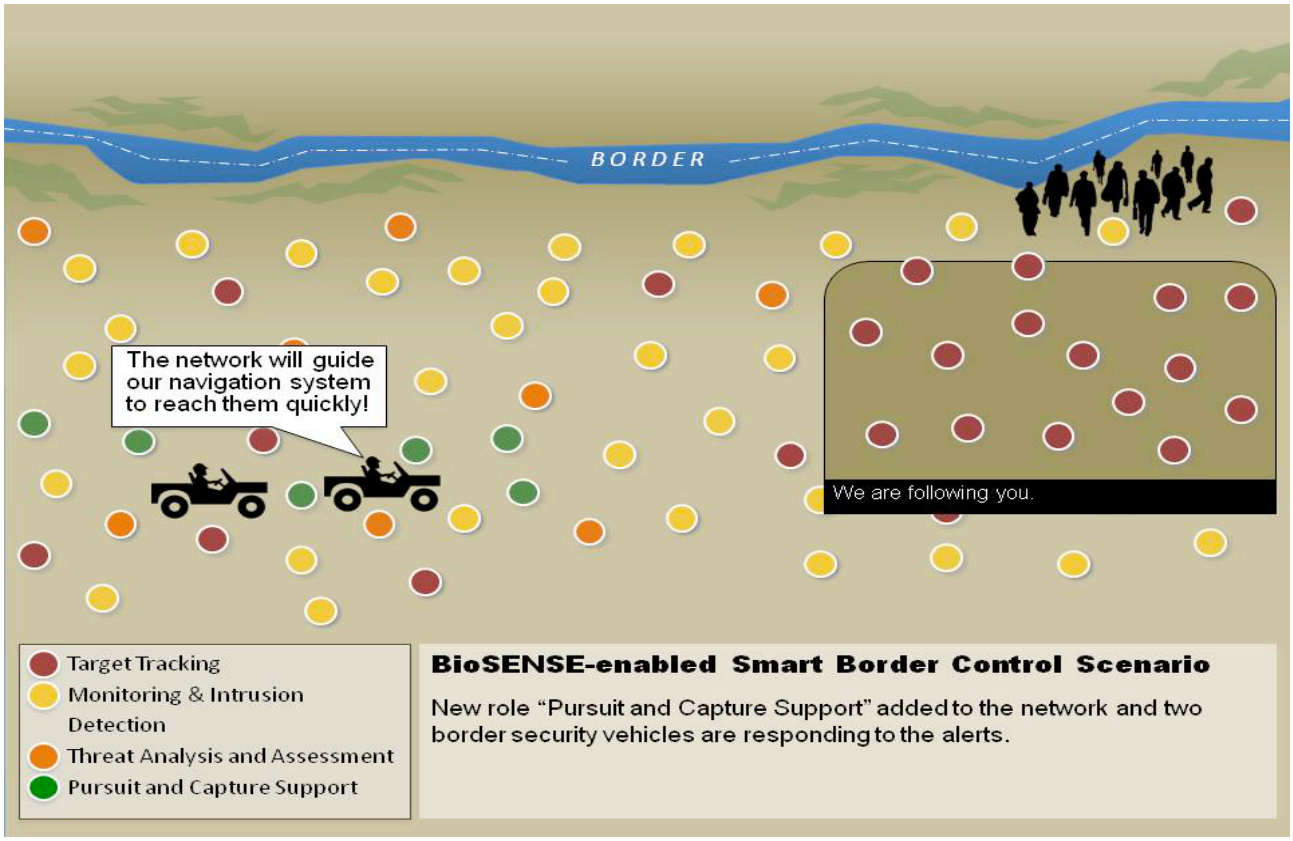

Figure 1.2: Border Protection Scenario: NSE Support for Pursuit-and-Capture

capture the intruders. Clive's team can dynamically update their mission and focus on the group of intruders that are more malicious. The network of sensors, HMMWVs, and UGVs can also partition themselves into two autonomous groups with appropriate capabilities so that the group with better firepower focuses on tracking the armed drug dealers while the other group focuses on tracking the illegal travelers as shown in figure 1.3. Moreover, if the sensor network can incorporate the HMMWVs and the UGVs into the network, then Clive's team can timely react to the malicious tampering of sensors and avoid costly sensor maintenance.

Research Challenges

This scenario illustrate the following research challenges:

1. NSE should be able to seamlessly assimilate the HMMWVs and the UGVs as new nodes with more powerful capabilities with exiting sensor nodes.

2. NSE should also be able to divide into groups and handle multiple concurrent tasks.

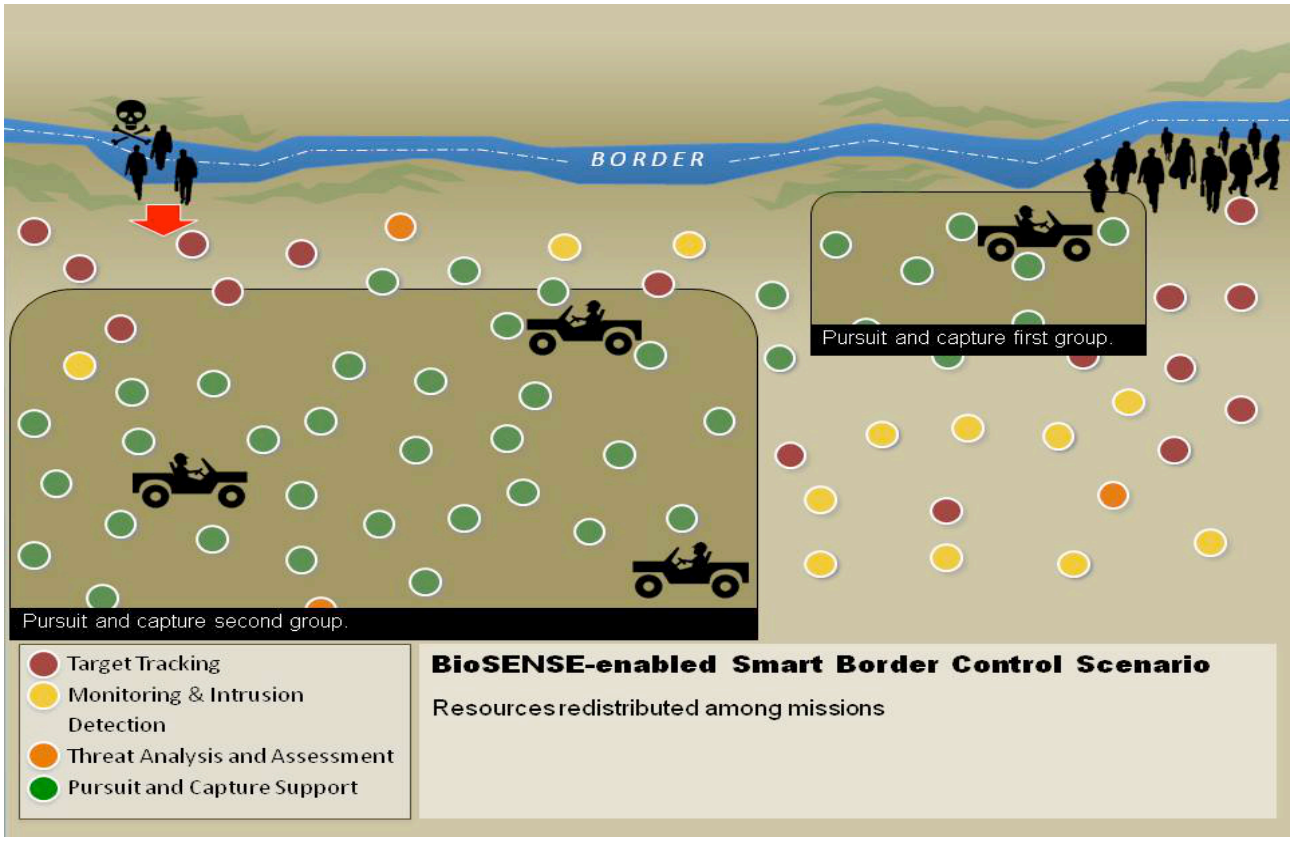

Figure 1.3: Border Protection Scenario: Multiple Concurrent Prioritized Missions

3. NSE should be able to dynamically update missions and their objectives.
4. NSE should minimize the impact of compromising individual nodes by the intruders.
5. NSE should be able to prioritize multiple concurrent tasks.

Mission Impact

Clive's team is able to form collaboration between the HMMWVs, the UGVs, and the NSE by incorporating the vehicles into the sensor system; the team can timely obtain the big picture of the field. During the pursuit-and-capture mission, the NSE is able to partition its resources into two autonomous groups, one responsible for tracking the drug dealers, and one for tracking the illegal travelers. The NSE was able to detect the local-sink attack, and switches to incorporate path redundancy, thereby mitigating the local-sink attack. The first group then can provide Clive's team with the whereabouts of the drug dealers and any necessary navigation assistance. The second group provides Clive's team with information about the path of the illegal travelers; the information is then forwarded to local authorities.

1.3 Motivation

The research presented in this dissertation is motivated by the following needs:

1. Seamlessly bridge the physical and cyber worlds.
    - Monitor and control physical environments.
    - Operate in harsh and hostile environments.
    - Enable smart cyber-physical applications.
2. NSE platform as a service
    - Sharing of NSE resources among multiple long-lived applications
    - Enable utility computing model
    - Lower setup and operation cost of NSE application
    - Better utilization of NSE resources through sharing network services as well as hardware devices
    - Enable inter-application collaboration

1.4 Research Statement

Our research focuses on the challenges facing the adoption of NSE-based solutions. Despite the tremendous amount of research in the fields of ad-hoc and wireless sensor networks, the world literature has yet to address the challenges of building a scalable, secure, resilient, and integrated computing platform to support the design, development, deploying, maintenance, and evolution of NSE software.

1.4.1 Problem Statement

We identify research gaps that impede the realization of our vision for future NSEs. We explore the gaps in both software architecture and networking realms in the following subsections.

Architectural Gap

Although the literature is filled with proposed software architectures and platforms targeting peer-to-peer, self-organizing, mobile, ad-hoc, and wireless sensor networks[56][34][39][46],

the ability to efficiently and securely share NSE resource among multiple applications serious challenge that has been recently highlighted as a missing link in state-of-the-art WSN research in [31]. Applications like traffic monitoring, parking management, fire alarming, and security enforcement could share common NSE resources to achieve multiple distinct goals in a more economic way. The evolution from single-purpose WSN deployments to more generic NSE platforms became inevitable for cost-effective production of NSE technologies. Common limitations of exiting WSN software architectures include:

Single purpose Most existing WSN software platforms are based on the limiting assumption that all nodes are participating in a single global task. Consequently software modularity, reconfigurability, and specialization were not considered design objective in such platforms.

Short lived The short lifetime of battery powered NSE nodes represent a primary obstacle against using NSE for long lived applications. Most existing WSN platforms fail to provide a systemic mechanism to extend lifetime of NSE applications beyond that of individual nodes, and consequently suffer from limited life time. Existing NSE research strive to maximize the network lifetime by being very conservative in energy usage, instead of providing the necessary mechanisms to incorporate newly deployed nodes to replace dead nodes.

Prohibitively Costly Existing WSN platforms are prohibitively costly and complex to deploy for many potential applications specially for large scale heterogeneous networks.

Insecure Most existing WSN platforms use preinstalled monolithic software programs on all nodes, such that the compromise of a single node in the network leads to the compromise of the entire network.

To fill such a gap, there is a need for new integrated NSE platform that:

1. Enables network command and control to quickly change network missions, priorities, software, and resource assignments without interrupting network operation or requiring any physical interaction with network nodes.

2. Enables structural and functional adaptation of software to suite different scales, environments, contexts, and topologies.

Networking Gap

Although many routing protocols have been proposed for WSNs, most of the proposed protocols either do not exploit the query semantics or are designed with a specific application or load pattern in mind[32][29][40]. A truly generic, flexible, and adaptive routing protocol

that can exploit the query semantics is essential for the efficient operation of large scale NSEs.

Routing is a challenging problem in WSNs due to its infrastructure-less nature and unreliability of nodes. Despite the anonymous nature of sensor devices, most WSN routing protocols still rely on unique node identifiers that applications are expected to provide. For applications to use such routing protocols, they need another resource discovery protocol that allows senders to query the network for the identifiers of the nodes they want to communicate with. This is currently the responsibility of the middle-ware. Such separation between resource or service discovery and route discovery results in unnecessary control traffic in the network that could drain the sensor batteries faster. For example, in AODV [62], broadcast is used to discover routes on demand. The same route control packets used to discover routes could serve a dual purpose by discovering resources too; saving the network from another broadcast storm.

1.4.2 Research Objectives

Efficiency Share resources among multiple long-lived applications. Allocate resources/services adopting demand-driven paradigm. Efficiently discover resources/services.

Adaptability Adapt to changes in mission, topology and context. Deploy software selectively at run-time over-the-air to allocated node. Allow applications to shrink and grow their allocated resources when needed.

Resilience Tolerate node/software failure. Enable live evolution of application and system software. BioSENSE exploits implementation diversity, variant shuffling, and automatic fault recovery to mitigate vulnerabilities resulting from implementation errors in any NSE software and consequently improve the resilience and fault tolerance of NSE software.

Security Reduce attack surface by transforming NSE software to a co-operatively moving target and provide trust-based operation. Diversity through shuffling is highly effective to slow down automated attacks like worms. In networks with no software diversity, the compromise of a single node most likely leads to the compromise of the whole network since same attack can be repeated successfully for all nodes at all time. In BioSENSE, if a nodes is compromised, the same attack is unlikely to succeed on different nodes or at different times due to software shuffling.

Elasticity Elasticity is achieved using the dynamism in forming organisms from cells at runtime. An organism is a directed graph of cells that supports the runtime binding of nodes and links resulting in a malleable computing platform that supports topology changes while preserving the operational integrity of running applications. The system manages the binding between cells within an organism. A replica of a cell on another

resource may be replace the master cell when its resource becomes unavailable. Elasticity is important for reliable operation of NSEs networks subject to node failures, mobility, and changing missions. It also allows NSE application to adapt to changing environment or context.

Programming Simplicity Isolate the programmer from the underlying system details and inherent complexities of NSE.

Scalability NSE software should not assume a specific network scale. The runtime system is responsible to run the software efficiently on networks of different sizes.

1.5 Contributions

BioSENSE enables QoS-sensitive, long-lived applications and services to be deployed over scalable, trustworthy, survivable NSE platforms and promotes wider adoption of NSE technology in our daily lives enhancing (micro-level) interaction with our physical surroundings and enabling unprecedented breakthroughs in critical areas like healthcare, sustainable environments, transportation, emergency response and public safety. The mapping between BioSENSE features and research objectives is illustrated in figure 7.1. Our contributions are summarized in the following subsections:

1.5.1 Intellectual Merits

Biologically-Inspired Design

First, we propose the Cell-oriented architecture that provides a simple and effective approach to design, develop, deploy, and maintain software systems with autonomous and adaptive characteristic. The cell-oriented architecture provides the following benefits:

1. Decouples software roles from the underlying role-players
2. Enhances the resilience of NSE software
3. Enables redundancy and fault-tolerance
4. Enables application and middleware evolution

Second, we propose a middleware design that exploits associative communication to provide the following benefits:

1. Selective on-demand distribution of application and middleware code

2. Software deployment is a first class runtime operation
3. Applications grow or shrink based on needs and conditions

Associative Communication Bus with Adaptive Criteria-based Routing

First, we propose associative addressing for NSE routing protocols, combining service/resource discovery with route discovery roles into a single coherent role and decoupling node identification from addressing using extensible criteria-based addressing of resources. We show that associative routing offers the following benefits:

1. Improves energy efficiency.
2. Minimize communication latency.
3. Enhances network scalability and simplifies management and maintenance operation through dynamic identification and addressing.
4. Improves the network resilience by allowing transparent fail-over to nodes with similar attributes.
5. Enables routing protocols to exploit application layer semantics.
6. Supports on demand multi-cast by eliminating the need of joining multicast groups.
7. Allows routing protocol to preserve the anonymity and privacy of nodes when needed.

Second, we present AMCR, a generic criteria-based routing protocol, exploiting associative addressing to allow destinations to be specified as a qualitative reference to node capabilities, administrative settings, and/or application published criteria. We show that AMCR offers the following benefits:

1. enables sharing of resources among multiple heterogeneous applications and allows the network to adapt to changes in application behaviour, operating environment, and QoS requirements.
2. provides seamlessly routing of unicast/multicast/anycast/broadcast traffic.
3. employs a novel criteria indexing mechanism to optimize communication efficiency and adapts to the observed application's semantics and load patterns.
4. preserves genericness by not imposing any restrictions on the application's choice of the desired communication patterns.

Figure 1.4: Resilience in BioSENSE: Moving Target Environment

Moving Target Environment

BioSENSE induces non-determinism in software behavior at runtime, resulting in attacker confusion. BioSENSE middleware attempts to systemically impede behavior discovery of NSE software implantations and mitigates exploiting vulnerabilities resulting from implementation flaws. Consequently BioSENSE reduces the attack surface and improve the security and resilience of the system. BioSENSE middleware encapsulates and autonomously manages semantically equivalent software component variants with dissimilar implementation behavior. BioSENSE is designed to enable trust-based hot shuffling of implementation variants at software component level to maximize the spatio-temporal software diversity in NSE. It also provides a systemic mechanism to exploit software redundancy to automatically recover from failures resulting from implementation flaws. See figure 1.4.

1.5.2 Broader Impact

BioSENSE, our proposed platform, has the potential to profoundly impact the NSE industry. By enabling the "platform-as-a-service" model for provisioning NSE applications, BioSENSE, will lower the market entry barriers that prevent wider adoption of NSE based solutions.

By providing a uniform architecture that supports the development, deployment, execution,

maintenance, and evolution of NSE software, BioSENSE will redefine the software engineering practices and processes used throughout the lifecycle of NSE applications leading to decreased cost and time to market. The wide adoption of BioSENSE will also accelerate the evolution of NSE software through trust-based component marketplace.

By enabling spatio-temporal software diversity and autonomous on-demand computing, BioSENSE lays the foundation for new research concerned with cooperative resilient autonomous systems and situation based control in cyber-physical environments [73].

The generic properties of associative routing, and AMCR, extends their applicability to wide range of computing environments other than NSE. The associative routing framework enables the integration of data-centric, service-centric, resource-centric, and communication centric networking paradigms, possibly leading to the evolution of new hybrid networking paradigms with unique properties.

1.6 Document Organization

The remainder of the dissertation is organized as follows. Chapter 2 presents an overview of the state of the art in the field of ad-hoc and wireless sensor networks focusing on architectural and networking concerns. Chapter 3 details BioSENSE design which includes programming framework, and enabling middleware. Chapter 4 describes our associative routing framework and adaptive multi-criteria routing protocol. Chapter 5 presents our evaluation approach, simulation framework, and evaluation results. Chapter 6 concludes the dissertation and highlights future work.

Chapter 2

Background and Related Work

"Good judgement comes from experience; experience comes from bad judgement" .. Jim Horning

2.1 Overview

NSEs, as we defined in this dissertation, share common characteristics with a number of actively-researched network classes like Wireless Sensor Networks (WSNs)[78], Mobile Ad-hoc Networks (MANETs)[51], Peer-to-Peer networks (P2P)[60], and Programmable Networks[14]. Before we address the challenges of NSE design, we need to better understand the limitations of existing network classes and the fundamentals for achieving the desired characteristics of NSEs. In this chapter, we explore the different aspects of NSEs and survey the state of art in the systems and networking research addressing these aspects within the contexts of various network classes. Our survey is guided by taxonomies for the programming, networking, and resilience aspects of NSEs.

2.2 Taxonomy

The goal of our taxonomy is to lay down the foundation for our review of the related work in literature. We provide taxonomy for three fundamental NSEs aspects; programming, networking, and resilience, then explore each aspect by following a top-down approach in research landscape partitioning, moving from general concepts to more concrete concepts.

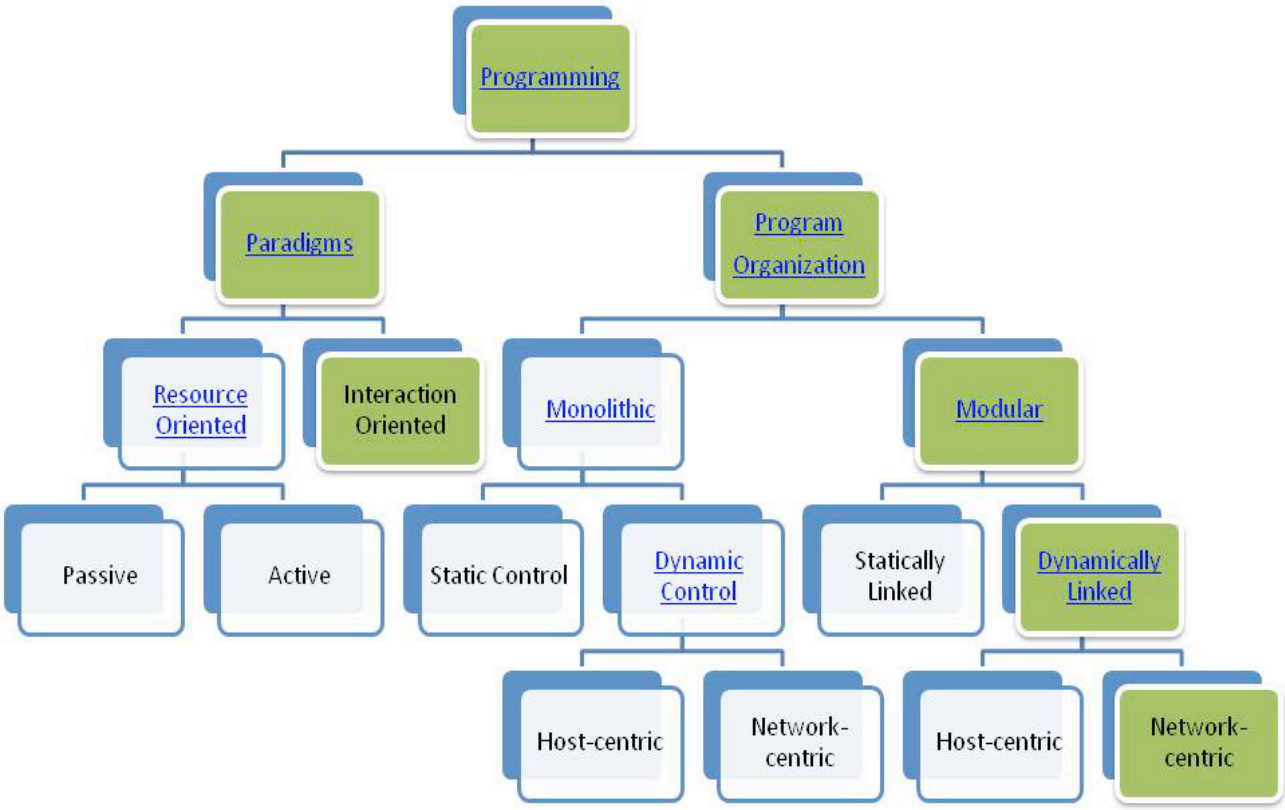

Figure 2.1: Taxonomy: Programming Landscape

2.2.1 Programming Landscape

We divide the programming landscape into two aspects; programming paradigms, and program organization. Figure 2.1 shows our taxonomy for the programming landscape.

Programming Paradigms

Programming paradigms describe the style and methodologies used to solve software engineering problems. Paradigms differ in the concepts and abstractions used to represent the elements of a program. As new technologies, platform, and applications evolve, new paradigms may be needed to maximize the efficiency of the software development process.

We classify programming paradigms landscape in two classes; resource-oriented paradigms, and interaction-oriented paradigms.

a) Resource Oriented

Resource-oriented paradigms directs the focus of software developers to resources when designing the application. Principles of resource abstraction, encapsulation, and composition guide the design process of resource oriented applications. This class of programming paradigms encourages the modelling of application entities as hierarchical families of abstract resources. We classify the resource-oriented paradigms into two subclasses according to the type of the resource.

I) Passive Resource

In this class of programming paradigms, the software system is composed of passive resources. Passive resource do not have an independent control or execution context. For example, data-oriented paradigm[71], and object-oriented paradigm[23]. In WSNs, a number of data-centric architecture has been proposed where the network is viewed as a distributed database management system. Applications uses a high-level 4GL language to access the network. This greatly simplifies application programming, although it lacks flexibility and control over the network resources and application performance. Examples are TinyDB [49], and Cougar [22].

II) Active Resource

In this class of programming paradigms, the software system is composed of active resources. Active resources are have independent control and execution context. The Service-Oriented Paradigm[48] is an example of a programming paradigm that relies on active resources. In recent years, integrating the Service Oriented Architecture (SOA) with sensor networks gained much attention from researchers around the world [38][18][44]. SOA realizes a loosely coupled software architecture that allows software resources to be plugged in and out on demand, and allows composition of complex services from simple ones. It is suitable for resource-centric environments where services represent a convenient abstraction of resources. Service Oriented SANETs [66] was introduced as an attempt to build open, inter-operable SANETs where nodes expose their capabilities to applications in the form of service profiles. OASIS is another programming framework and middleware for service oriented sensor network[41][4]. The *REpresentational State Transfer* (REST) architectural style, introduced by Fielding[27], defines a uniform connector interface for globally identified resources. REST has been applied successful in the world wide web. It is characterized by *interface genericity* for making

the system components expose the same minimal interface. This allows all interactions to be resource-generic rather than resource-specific[27].

b) Interaction Oriented

Interaction-oriented paradigms directs the focus of software developers to the interactions among distributed nodes to collect, organize and comprehensively utilize available resources. In this class of paradigms, the resource aggregation logic is effectively separated from the business logic. The programming constructs provide abstractions for interaction aspects like collaboration processes, team plans, roles, agents, contracts and missions.

Interaction-Oriented Programming (IOP) was originally proposed to construct complex multi-agent systems. IOP attempts to realize a more tractable and practical approach to orchestrate the interactions among agents than the general agent programming approach. It enables expressive declarative specification of agent interactions to channel the intellectual energies of designers into the most effective design tasks[72].

The Role-Oriented Adaptive Design (ROAD) was proposed as a general-purpose paradigm for adaptive software systems[20]. ROAD provides composite applications the ability to adaptively and continuously reconfigure itself to remain viable as environment and run-time conditions undergo changes. ROAD exploits contracts to realize adaptive service compositions that can be dynamically constructed by connecting and configuring other services.

Program Organization

Program organization refers to the scheme used to organize different sections of an executable programs for execution. We classify the program organization landscape into two classes; monolithic, and modular.

a) Monolithic

The term monolithic is traditionally used to describe programs that do not have any form of modularity, however, for sake of this taxonomy we use the term to describe program runtime organizations that have a single program image executed by all nodes. Monolithic programs are built as a single, unstructured, self-contained software unit. Most existing wireless sensor networks are based on monolithic applications. We classify the monolithic programs into two subclasses according to the type of control.

I) Static Control

In this class, the logic control is statically specified in the monolithic program image. Programs in this category are usually designed for a very specific task that is unlikely to change. It provides the least flexibility of all types of program organizations.

II) Dynamic Control

In this class, the logic control is is external to the monolithic program image. The program is designed to provide a number relatively generic commands, or services that can be invoked in the order and configuration specified by an external program. For example, an interpreter maybe implemented as monolithic program image, but the control flow is determined by the interpreted scripts. The same interpreter can run multiple scripts with different control flows. We classify monolithic program with dynamic control into two classes according to the scope of control.

- Host Centric

In this class, the program *control* is remains on a single host, such that control script is completely executed on that host. Virtual machine based platforms for WSNs belong to this class. In such platform, a bytecode interpreter supports the execution of applications. It achieves energy efficiency by providing a concise way to represent the application logic, although it suffers from instruction interpretation overhead.

- Network Centric

In this class, the program *control* can be transferred from a host to another using code mobility. Agent-based platforms for WSNs belong to this class. Agent are interpreted programs that can migrate from one host to another during its lifetime. Application modules are presented to the network as autonomous mobile agents. This approach supports the dynamic evolution of applications, although it suffers from high overhead making it less suitable for resource constraint environment. Agilla [28] and Impala [45] are examples of agent-based platforms for WSNs.

b) Modular

In this class, programs are constructed from smaller standardized units called modules that are linked together. The decomposition of the program structure into modules improves flexibility, reusability, and maintainable of software. We classify the modular programs into two subclasses according to the time modules are linked to each others.

I) Statically Linked

In this class, modules are linked to each others at system integration time. Once integrated, the program structure can not be altered. At runtime, statically linked modular program have similar characteristic to monolithic programs, however they allow more flexibility at development and maintenance times. In WSN, applications built for TinyOS belong to this category[52].

II) Dynamically Linked

In this class, modules are linked to each others at runtime. Programs can change the bindings between modules when needed. This enables structural plasticity of programs and minimizes the overhead of software updates. We classify modular program with dynamic control into two classes according to the scope of linking.

- Host Centric

In this class of dynamically linked modular programs, all program modules must exist on the same host (NSE node). In WSNs, applications built for RETOS belong to this category[17].

- Network Centric

In this class of dynamically linked modular programs, different modules composing a program could exist at different hosts. This allows programs to scale beyond the storage and computational capabilities of a single host (NSE node). The BioSENSE platform, presented in this dissertation, belongs to this class. In BioSENSE, *cells*, hosted at different nodes, can be bound together into multi-cellular organisms that play application roles. To our knowledge, no other platform provide such functionality for networked sensors.

2.2.2 Networking Landscape

Networking paradigms are concerned with communication patterns, addressing, and routing. Our taxonomy for the networking landscape focuses on networking paradigms for ad-hoc environments like WSNs and MANETs. We classify networking paradigms according to their addressing schemes to two major classes; *resource-centric* and *communication-centric*. Figure 2.2 shows our taxonomy for the networking landscape.

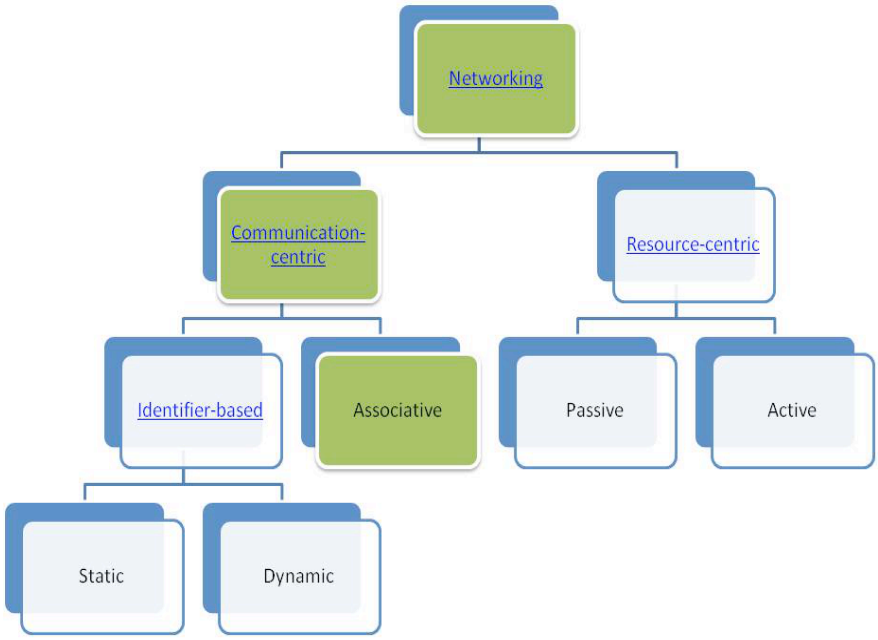

Figure 2.2: Taxonomy: Networking Landscape

Resource Centric

In this class of paradigms, resources are focus of the programmers attention. We classify resource centric paradigms according to the type of resource to two classes; *passive* and *active*.

a) Passive Resource

In this class, the network is abstracted as a distributed database. Routing is based on the requested data content. Applications are completely oblivious of the communication endpoints. The network layer is usually inseparable from the application layer. Data-centric routing is also referred to as as content-based routing. TinyDB [49] and EnviroTrack [2] are examples of WSN middleware systems that adopts content-based routing.

b) Active Resource

In this class, the network is abstracted as a service environment composed of multiple service providers. Routing is based on the requested service profile. Like data-centric routing, applications in service-centric routing are completely oblivious of communication endpoints and the network layer is usually inseparable from the application layer. Service-centric routing realizes service oriented architecture (SOA) in MANETs and WSNs. For example, the Reliable Adaptable serviCe-driven Efficient Routing (µRacer) [67] suite of protocols adopted a novel service-driven routing approach [66] for WSNs that exposes the node capabilities to applications as service profiles. µRACER translates service profiles into efficient paths for service requests.

Communication Centric

The network is viewed as a substrate enabling communication among end points. Routing is based on destination addresses of end points. The network layer has clear boundaries and usually separable from application layer. Communication-centric routing protocols are the most generic protocols that do not restrict the application programming paradigm or communication patterns. We further classify communication-centric routing into two subclasses:

a) Identifier Based

Destinations are identified using unique identifiers to nodes or groups. There is always a direct mapping between identifiers and nodes.

I) Static

A globally unique network address is used to identify the node. The network address of a node does not change as it changes its location. Most MANET routing protocols belong to this class. For example, AODV [62];

II) Dynamic

The node identifier is decoupled from its network address, allowing network address to change as node changes its location while providing a way to dynamically map node identifiers to their current network addresses. DART [25] and its multi-path variant M-DART [13] use distributed hash tables for this purpose.

b) Associative

Destinations are identified using descriptive addressing that does not directly map to specific nodes or multicast groups. AMCR is an example of an associative routing protocol designed for shared sensor networks. Addressing in AMCR is based on generic resource attributes. We developed AMCR as an example of a generic associative routing protocol capable of adaptation to observed workloads.

2.2.3 Resilience Landscape

We define resilience of as the ability of a system to preserve, or autonomously restore, its operational integrity in situations of external attack or hitting internal (software or hardware) faults. We divide the resilience landscape into two aspects; redundancy, and recovery. Figure 2.3 shows our taxonomy for the resilience landscape.

Redundancy

Redundancy is essential to eliminate single points of failures. System components, whether being software or hardware, are subject to failure due to many unpredictable factors ranging from manufacturing faults to damage due to cyber or physical attacks. By having redundant system components fault tolerance and recovery become possible. We classify redundancy into two main classes; resource redundancy, and interaction redundancy.

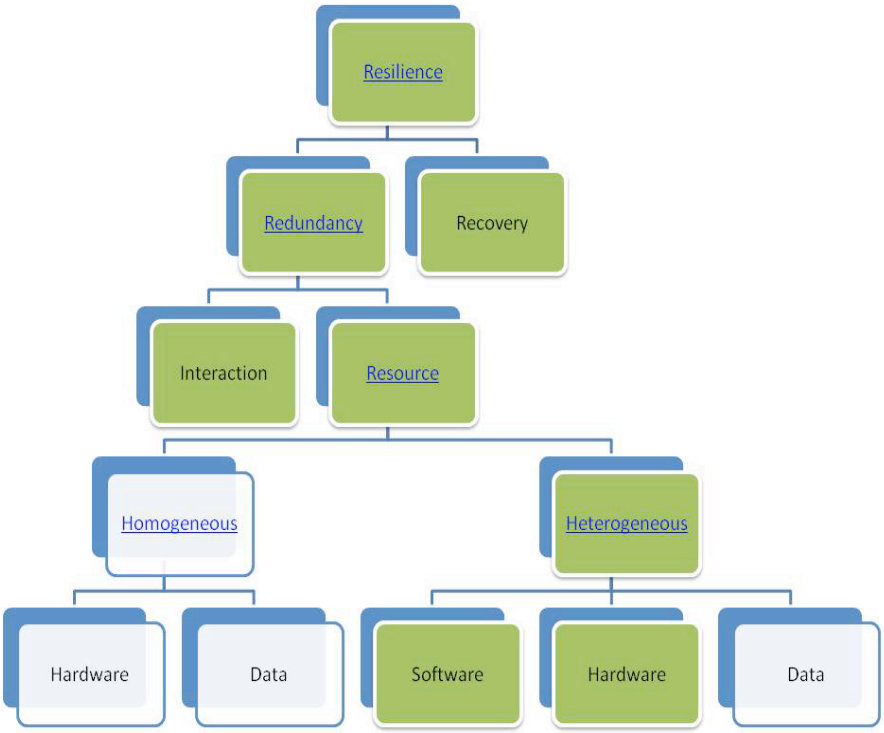

Figure 2.3: Taxonomy: Resilience Landscape

a) Resource Redundancy

In this class, the system employs multiple equivalent alternative for critical resources. We classify resource redundancy according to the similarity between the alternatives to two classes; *homogeneous* and *heterogeneous*.

I) Homogeneous Resource Redundancy

In this class, resource redundancy is achieved using identical clones of critical resources. While homogeneous resource redundancy can be useful for arbitrary failure that affect individual resource clone, it can not help if the failures is due to inherent flaw in the resource design or replicated state, since the redundant clones will posses the same flaw and cause the same failure. We classify homogeneous resource redundancy according to the resource type into two classes; *hardware* and *data*.

- Homogeneous Hardware Redundancy

In this class, identical clones of hardware components are used. To allow recovery the internal state of all hardware clones should be synchronized so that when one component fails, a clone can take over its responsibilities.

- Homogeneous Data Redundancy

In this class, multiple identical copies application data are stored in different places to recover from failures causing data loss.

II) Heterogeneous Resource Redundancy

In this class, resource redundancy is achieved using dissimilar alternatives of critical resources. Heterogeneous resource redundancy exploits resource diversity to improve system resilience. Unlike, homogeneous redundancy, heterogeneous redundancy can be useful in cases of failures due to inherent flaws in the resource design, since alternatives may not posses the same from due to diversity. We classify homogeneous resource redundancy according to the resource type into two classes; *hardware* and *data*.

- Heterogeneous Software Redundancy

Unlike the case with homogeneous redundancy, the software, as a resource, can benefit from heterogenous redundancy using implementation diversity. Software implementation diversity

has been proven to be the most general way of coping with residual software faults.

Software systems are comprised of sets of interconnected components that act and interact together to achieve a desired functionality. Each component adheres to certain specifications defining its explicit behavior. At the same time, each component exhibits an inherent implicit behavior due to the potential individuality and variability across different execution environments and different implementations of the same specifications. Accordingly, we can classify the types of vulnerabilities of a software system into:

Specification vulnerabilities Those exist due to flaws in the system specifications. We are not concerned with this type of vulnerabilities as they are likely to be detected early on by the careful analysis of the specifications documents, which is a common and necessary process in systems engineering; and

Implementation vulnerabilities Those result from implementation flaws as a side effect of implementing the specifications of the system or any other component of the execution environment. This class of vulnerabilities is much harder to detect, since auditing the implementation requires much more effort than auditing the specifications and is usually infeasible, especially for commercial off-the-shelf components. Obviously, this type of vulnerabilities is implementation dependent and almost unavoidable.

Different implementations of the same software specifications preserve the explicit behavior of the system, but exhibit variations in the implicit behavior, and hence have different sets of implementation vulnerabilities. Attacking a system through implementation vulnerability involves two steps:

a) Discovering the vulnerability by observing and analyzing the implicit behavior of the system when given a specific input; and

b) Exploiting the vulnerability by providing the input learned from Step 1 to instances of the system having the same implementation.

It can be noticed that both steps desire the system to possess a deterministic implicit behavior for an attack to succeed. In N-version programming (NVP)[5], multiple versions of a program are independently developed by different programming teams. N-version relies on the independence in the development of program versions to achieve diversity. In NVP, multiple versions run concurrently and their results are compared by an adjudicator that usually uses majority vote to select the result assumed to be correct. BioSENSE exploits software redundancy to improve NSE resilience and introduce behavior non-determinism to confuse attackers.

- Heterogeneous Hardware Redundancy

In this class, redundant hardware of different brands or models are used. If a specific task triggered an inherent flaw in the hardware design or manufacturing, restarting the task using different hardware may be helpful (unlike in case of homogeneous hardware redundancy).

- Heterogeneous Data Redundancy

In this class, redundant copies of application data are stored with minor perturbation. This is motivated by the observation that in some cases software failures occur for certain data values in the input space that could be avoided with if the minor perturbation of input data which is acceptable to the software. Data diversity was first introduced in [3].

b) Interaction Redundancy

In this class, the reliability of interaction over unreliable communication substrate can be enhanced using redundancy in the interaction behavior. For example, *path redundancy* is exploited to enhancing the reachability to the destination node and localize the effects of route failures in [37].

Recovery

The ability of a software system to automatically recover from software failure is key to enhance it resilience. Failure recovery techniques allow continued operation of system if some types of faults are encountered. A fault tolerant framework that supports automatic fault recovery in WSN was proposed in [68].

2.3 Related Work

The WSN architecture research is influenced by many sources in literature. Many of the related work has been done in the context of self-organizing, mobile, ad-hoc networks[56][34]. Although the intended purposes of these networks are different, they share some of the most distinctive characteristics of wireless sensor networks like the infrastrastructureless nature, need for decentralized control and distributed organization. The WSN architecture research is also influenced by peer-to-peer computing[7][16][46], grid computing[65], active and programmable networks[15][11][12], mobile agents[9][77], and swarm intelligence[8].

2.3.1 Software Engineering Principles

In this subsection we explore some of the software engineering principles that inspired the design of BioSENSE.

Separation Of Concerns

Separation of concerns is a general problem-solving idiom that enables us to break the complexity of a problem into loosely-coupled, easier to solve, subproblems[54]. The concept has been extensively used in software engineering to define component modularization boundaries in harmony with the natural modularisation boundaries of requirements, and consequently simplifies the software development process. Traditionally, Aspect Oriented Requirement Engineering (AORE) approaches, like viewpoint oriented, use case/scenario, and goal oriented approaches, has be successfully used in the system analysis phase, however, the design and implementation phases could not directly map these requirements into design and implementation artefacts before Aspect-Oriented Programming paradigm (AOP) was first introduced [35]. AOP exploits the separation of concerns principle to separate cross-cutting programming concerns starting from the requirements phase to the implementation phase, passing through the design phase. AOP simplifies the capture of some important design decisions that procedural and object oriented paradigms can not clearly capture due to the fact that they coss-cut the system basic system functionality[36]. BioSENSE also exploits the separation of concerns principle to preserve programming simplicity and enable platform managed services while effectively achieving sophisticated design objectives.

Information Hiding

Information hiding is the principle that users of a software component need to know only the essential details of how to initialize and access the component, and do not need to know the details of the implementation or internal data structure [10]. According to this principle software modules are used via their specifications, not their implementations. The segregation of the program logic, subject to frequent change from other, from static parts of the program is encouraged to protect data structure and core methods from extensive modification if the design decision is changed. The internal implementation of modules is protected by providing a stable interface which protects the remainder of the program from the details that are most likely to change. Modules may provide optional mechanisms to prevent certain aspects of modules from being accessible by other modules, hence enforcing information hiding instead of merely encouraging it. The cell-oriented architecture, proposed in this dissertation, adopts information hiding through strict separation of data from logic to enable run-time implementation shuffling and transactional state management. This allows BioSENSE platform to provide automatic failure recovery and other resilience services.

Role-Oriented Design

The role is a commonly used design concept that facilitates the separation between roles from role players for both system design and permission management. The Role-Based Access Control (RBAC), a standardized access control mechanism commonly used in enterprise systems, is a flexible approach to restricting system access based on roles[26]. RBAC provides role oriented programming paradigms a convenient and proven security mechanism that could considerably harden the security and improve accountability of role-oriented software programs. Role engineering refers to the engineering methodology for designing, implementing, deploying, securing, managing, and evolving role oriented software systems.

In NSEs, where different application components are likely to exist on different nodes, the relationships between application components can be non-deterministic and unreliable necessitating adaptive reconfiguration to continuously satisfy the system requirements. BioSENSE employs role engineering to enable dynamic self reconfiguration of shared NSE. The application provides the implementations of roles and the NSE provides the role players. The assignment/reassignment of roles to players is dynamic and maintained by the runtime environment.

2.3.2 WSN vs Mobile Ad-hoc Networks

WSNs share a number of common features with Mobile Ad-hoc NETworks (MANETs), essentially, due to the infrastructure-less nature of both network classes. In both classes, routing may involve intermediate nodes to relay network packets from source to destination, a technique known as multi-hop routing. Self-organization and self-management are essential properties of software systems and protocols in both classes. The dynamic topology is another common features, although the main reasons for topology changes differ. In MANETs, topology changes are often due to node mobility, while in WSN, they are mainly due to node failures. Also, nodes in both types of networks are usually battery-powered requiring routing protocols to be energy efficient. Despite, the many common features, routing protocols for WSNs were differentiated from MANETs according to their communication patterns. In MANETs, all nodes are assumed to be peers and communication is usually node-to-node, while in WSNs, communication usually involves a sink that plays the most important role in the network. Also, unlike MANET nodes, sensors do not usually have globally unique identifiers, challenging address-centric communication and motivating data-centric communication instead. In WSNs, node-to-node communication was traditionally assumed a rare communication pattern, making WSN routing protocols focus on sensor-to-sink, and sink-to-sensor patterns.

While there are many active research efforts in the direction of WSN programming, most of the proposed models and software architectures are not application to large-scale multi-purpose NSEs. The lack of general-purpose flexibility is one of the very common limitations.

The trade off between essential software quality factors is usually influenced by the needs of a specific type of application making the work less suitable for different types of applications. Few systems attempted to leverage the potential of in-network storage to confine the processing and storage of the entire application inside the network [58].

2.3.3 Dynamic Reconfiguration in WSN

The problem of dynamic reconfiguration in WSN has recently attracted the attention of researchers. Research efforts are addressing the problem from two different perspectives:

1. Dynamic topology reconfiguration.
2. Dynamic software reconfiguration.

Our focus is on the latter more recent category. In [6] a composition based approach was proposed to address dynamic reconfiguration of software in WSN. The proposed approach is based on the Fractal component based software engineering[21] focusing on the application level component rewiring and employing -calculus for communication semantics. Applications are built using the Insense language. Important aspects of WSN reconfiguration like code distribution and change location specification were not addressed. In [55] a fine grained software reconfiguration in WSN was proposed whereby a programming model and a runtime system were proposed to solve the dynamic reconfiguration problem. Code distribution is addressed, although the distribution is performed eagerly from a code injection point to the whole network requiring the code dissemination node to stay in network till the whole network is updated. Such eager update approach does not scale well for large scale WSN. The above-mentioned solutions to software reconfiguration in WSN are addressing the problem at the fine grained component level shifting the focus of application developers away from the global mission of the network and how to model network level mission changes to the details of updating components on individual nodes. To the best of our knowledge, none of the existing solutions to the dynamic reconfiguration problem in WSN provide a way to model the network mission or enable role oriented application design and re-specialization of nodes. Also, none of the solution provide a technique for on-demand (lazy) code distribution that can benefit from multi-level in-network storage and caching of code.

The role-oriented approach to adaptive software design has been explored in [76] and [20].A role-based approach for the hierarchical self-organization of sensor nodes was presented in [39]. Federated and shared use of sensor networks was addressed in [31]. In [58] an autonomous NSE platform was proposed to enable secure dynamic task-based networking and in-network storage.

2.3.4 Ad-hoc Routing

Routing is one of the most challenging problems in NSEs due to its infrastructure-less nature and unreliability of nodes. Although many routing protocols were proposed for NSEs, most of them still rely on unique node identifiers that applications are expected to provide. For applications to use such routing protocols, they need another resource discovery protocol that allows senders to query the network for the identifiers of the nodes they want to communicate with. This is currently the responsibility of the middle-ware. Such separation between resource or service discovery and route discovery results in unnecessary control traffic in the network that could drain the sensor batteries faster. For example, in AODV [62], broadcast is used to discover routes on demand. The same route control packets used to discover routes could serve a dual purpose by discovering resources too; saving the network from another broadcast storm.

Despite the classical distinction between WSNs and MANETs routing protocols, the differences between the technical requirements in both classes of protocols are getting blurry, due the evolution of shared multi-purpose WSNs and the need to integrate WSNs and MANETs into large scale cyber-physical systems. Most of the assumptions underlying the differentiation between the communication patterns used for WSNs and MANETs are no longer considered valid for such environment. For example, content-based routing was proposed as a better alternative to address-based routing in MANETs [63]. Secure node-to-node communication was also extensively studied for WSNs [64]. A combined routing Layer for WSNs and MANETs was proposed in [70] motivated by the need to integrate heterogeneous networks of both classes.

Routing protocols targeting WSN platforms usually attempt to achieve routing efficiency through exploiting the application layer query semantics by proposing an all-in-one solution that weave the routing concern with other application layer concerns, such as data-centric and service-centric routing. They also tend to optimize routing performance for a specific communication pattern inspired by a specific a class of WSN applications. A number of routing protocols for sensor networks assume a flat network where sensed data flow from multiple sources to a particular base station and focus on minimizing redundancy to achieve energy efficiency as in [32][29][40]. Such protocols often assume identical roles for all nodes, which is not a practical assumption in large scale heterogeneous networks. Data reporting, mobility, and role assignment models are often dictated by WSN protocols, resulting in interdependence between the application and routing concerns disqualifying those protocols from being general-purpose. Associative routing addresses these limitation by providing a generic routing layer capable of integrating multiple different application layers and still provide the means to exploit the application layer semantics through descriptive addressing.

2.3.5 Distributed Architectures

A distributed system consists of a collection of independent computers that interact together through computer network to collaboratively provide single system view to applications allowing them to utilize distributed hardware resources and software services. A distributed system architecture is often influenced by a specific architectural style. The architecture style describes a class of architectures or significant architecture pieces that are commonly used in practice and bundles a coherent set of design decisions and architectural patterns[19]. An architectural pattern defines reusable system structures, interconnections and interactions among them. It provides a template for architects to describe subsystems and communication among them. The four most commonly used distributed architectural styles are: *object-oriented*, *service-oriented*, *resource-oriented*, and *agent-oriented*. In this subsection, we compare and contrast these four architectural styles in addition to our proposed *cell-oriented* architecture. The comparison is summarized in table 2.1.

Table 2.1: Architectures comparison chart

Criterion	Object-Oriented	Service-Oriented	Resource-Oriented	Agent-Oriented	Cell-Oriented
Building blocks	Objects	Services	Resources	Agents	Cells
Communication Style	synchronous (call-based)	synchronous(call-based)	synchronous (call-based)	hybrid synchronous / asynchronous(AGI[69] communication models)	hybrid synchronous / asynchronous (contract defined)
Addressing	References to unique object instances	Single end-point address for each service	Unique address (URL) for each resource	Unique global identifier for each agent	Descriptive criteria-based addressing of cells
Request routing	unicast to unique object	unicast to a unique service endpoint	unicast to a unique resource instance	unicast to a unique agent instance	anycast to a number of cells satisfying required criteria
Responses Caching	No	No	Yes	No	No
Application Interface	Specific to this object / class. Description is middleware specific (e.g. IDL)	Specific to this service. Description is protocol specific (e.g. WSDL)	Generic to the request mechanism (e.g. HTTP verbs)	Specific to agent platform (e.g Jade) and the agent specifications	Generic in case of stem cells and contract-specific in case of specialized cells
Data format description	Yes. Specific to middleware (e.g. IDL in CORBA)	Yes. Part of service description (e.g. XML Schema in WSDL)	No. Nothing directly linked to address/URL	Yes, communication protocol specific	Yes, specified in cell contracts
Design Focus	Abstracting application domain entities as object classes and capturing their attributes and behaviour	Defining remotely accessible functions for consumers based on request / reply semantics	Mapping application resources into unique addresses (e.g. URLs in REST)	Agent work breakdown structure	Mission, goals, and roles
Behavioural Determinism	Deterministic	Deterministic	Deterministic	Deterministic unless soft computing is explicitly employed	Support architectural non-determinism
Platform Elasticity	undefined	undefined	undefined	undefined	inherently supported
Failure recovery	undefined	undefined	undefined	undefined	inherently supported
Composability	Yes. Through aggregation	Yes. Through service composition	Yes. Through proxy resources	undefined	Yes. through wiring cell ports into muti-cellular organisms (structural), and through role composition composition (functional)
Autonomic Components	No	No	No	Yes	Yes

Object-Oiented Architecture (OOA)

The object-oriented paradigm is one of the most popular programming paradigms in software industry. In this paradigm, the responsibilities for providing the application functionality is divided among individual reusable and self-sufficient objects, each containing the attributes and the behavior needed to model an application-domain object. During the last couple of decades, research in the programming languages area has shifted focus from the procedural paradigm to the object-oriented paradigm, resulting in the domination of the object-oriented paradigm over the programming language landscape. As a result, the majority of today's software developers are familiar with object oriented-programming and tools for object-oriented analysis and design because available and affordable. Accordingly, distributed architectures that exploit the object-oriented paradigm, to allow remote objects to be transparently accessed and invoked over a computer network, can significantly reduce the cost of developing distributed applications. They also provide a simple and straightforward path to convert existing object-oriented applications, designed for a centralized computing environment, to scalable distributed applications. The object-oriented architecture facilitates synchronous communication with remote objects based on call/return semantics. Unlike message passing systems, the caller has to block waiting for the callee to finish executing the invoked method and sends back the methods return value. An object-oriented middleware facilitates the remote method invocation transparently from programmers. This is usually achieved with the assistance of computer generated client *stubs* and server *skeletons* that make method invocation for remote and local objects appear identical. Applications maintain references to remote objects as if they were local references. Each object reference is tightly coupled with a unique object instances either local or remote to which a method invocation is directly routed. The caller needs to have design time knowledge of the callee interface to be able to invoke its methods. Such interface is specific to the object class. An object do not maintain an execution context of its own. Instead, object methods are executed in the same execution context as the caller, hence, objects can not operate autonomously. The most popular middleware solutions adopting the distributed object-oriented architectural style is the Common Object Request Broker Architecture (CORBA) [59] standardized by the Object Management Group (OMG™) and the Java Remote Method Invocation (JavaRMI)[75]. CORBA allows distributed applications to be constructed using different object-oriented programming languages and utilize computers with heterogeneous architectures and internal data representations. This required object interfaces to be defined in a programming language independent way. For this reason CORBA employs Interface Definition Language (IDL) as a universal language to unambiguously define interfaces for objects implemented in different object-oriented programming language. The IDL compiler reads class definitions in IDL and generate language specific stubs and skeletons that are later linked with the client and server applications respectively. The Java RMI provide a more simple alternative of CORBA in case the application is entirely written in Java. The object serialization and reflection features of Java programming language eliminate the need for external data representation formats or separate interface definition language when using

Java RMI.

Service-Oriented Architecture (SOA)

Service Oriented Architecture (SOA) is a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains [48]. Organizations can create capabilities needed to solve a specific problem, and share these capabilities to service other organizations in need for these capabilities. SOA realizes a loosely coupled architecture for service providers and consumers, jointly referred to as service participants, to interact together. The webservice architecture[1] is the most popular architecture representing the service-oriented architectural style. Like OOA, SOA provides synchronous communication facility with remote services based on call/return semantics. Each service has a single end-point address. Accordingly communication with service is implicitly stateless. This stateless nature makes SOA generally more scalable than OOA. SOA does not define any operations for service lifecycle management. The service interface is specific to each service end-point and describes the messages and payload formats. For example, in webservice architecture a Web Service Definition Language (WSDL) is to define service interfaces. Different architectures and communication protocols adopting the service-oriented architectural style may use different languages and/or formats to describe service interfaces. Although SOA is usually associated with course grained capabilities, the granularity of the functions performed by services is completely up to application designers. Course grained services can be composed from fine-grained services. Recently, web services composition became a popular practice to support business-to-business or enterprise application integration.

Resource-Oiented Architecture (ROA)

The resource-oriented architectural style channels the intellectual energy of software designers towards identification and naming/addressing of application domain resources. The *resource* is an abstract concept representing every application component deserving to be directly accessed and represented in the application. Resources in ROA are handled through a uniform interface common for all types of resources. The *REpresentational State Transfer* (REST) architecture[27] is the most popular architecture based on the resource-oriented style employing only web technologies and using HTTP as a transport. REST defines a uniform connector interface for resources based on the HTTP verbs; GET, PUT, POST, HEAD, and DELETE. This common interface for all types of resources allows all resources to be resource-generic rather than resource-specific[27]. The REST uniform interface is described in table 2.2.

One of the advantages of REST is that it supports response caching for GET requests. Course grained resources can be composed of fine grained resources using proxy resources.

Table 2.2: REST unified interface methods

HTTP verb	Method name	Description
GET	Get resource	Retrieves the representation of the current state of a resource
PUT	Create resource	Creates a new resource
POST	Update resource	Modifies an existing resource
HEAD	Get resource metadata	Retrieves the metadata of a resource
DELETE	Delete resource	Deletes a resource

Agent-Oriented Architecture (AOA)

The agent-oriented architectural style defines the structures and interactions to support the execution of applications adopting the mobile multi-agent paradigm. Multi-agent systems are composed of autonomous components, namely agents, that collaborate to complete a task by breaking down the task into subtasks. Each subtask is assigned to an agent that can either complete the task independently or with co-operation of other agents. Code mobility is an important feature of AOA allowing agents to travel from one host to another to find resources of interest. Agent mobility involves the transfer of both code and internal state from over the network resulting in significant overhead for frequently travelling agents specially in resource constraint environments. Agent usually implement domain specific protocol to facilitate communication. For example, the Asynchronous, Group-Oriented, and Inter-Agent (AGI) communication architecture provides abstract models from which new communication types can be defined. The first model defines the asynchronous events and their application-specific handlers. The second model defines group communication to facilitate collaboration among agent groups. The third model defines direct agent-to-agent communication. Individual agents are assigned global identifiers in a multi-agent system.

2.4 Conclusion

In this chapter we presented taxonomies exploring the programming, networking, and resilience aspects of NSE design. We overviewed the factors that influenced the design of existing software systems for wireless sensor networks, mobile adhoc networks, and various other distributed platforms and surveyed the related work in literature. We observed that despite the existence of solid and concrete research base addressing various design aspects of platforms targeting peer-to-peer, self-organizing, mobile, ad-hoc, and wireless sensor networks, the ability to efficiently and securely realize large scale shared NSEs is still a missing link in literature.

Chapter 3

BioSENSE Platform

"Simple things should be simple and complex things should be possible" .. Alan Kay

3.1 Overview

BioSENSE allows multiple networked systems designed by different entities for different purposes to be morphed into a single shared network to collaborate and achieve multiple dynamic goals. BioSENSE enables applications to allocate, at runtime, a dynamic taskforce, running over a shared pool of federated resources that adapts to its evolving mission requirements. While NSE applications are oblivious of the BioSENSE resource management, BioSENSE exploits application semantics for efficiency. BioSENSE perceives both applications and the NSE itself to be elastic, and allows them to grow or shrink based on needs and conditions. Our general approach is to support dynamic node re-tasking and network reconfiguration, efficient addressing and routing over large-scale NSE with anonymous nodes, redundancy management, and dynamic QoS provisioning and resource allocation.

The most compelling design solutions are ones that succeed to hide system complexities from programmers while effectively satisfying the design objectives. However, this goal is difficult to achieve in inherently complex distributed computing platform, encompassing a large number of design objectives, unless the programming framework effectively separates design concerns and the runtime platform provides a comprehensive set of fully managed services like, state replication, resource management, topology management, resource re-specialization, and indexing. Accordingly, BioSENSE design exploits separation of concerns and platform managed services to preserve programming simplicity while effectively achieving

sophisticated design objectives. BioSENSE is realized in three major components; programming framework, middleware, and application server. BioSENSE middleware functionality is split among a minimalistic kernel installed on all NSE nodes and dynamic middle-ware roles that are selectively deployed at runtime. We refer to the middleware kernel as BS-DNA.

In this chapter we illustrate and discuss the design of BioSENSE major components. We start with introducing the cell-oriented architecture and programming framework, then finally, illustrate the middleware architecture.

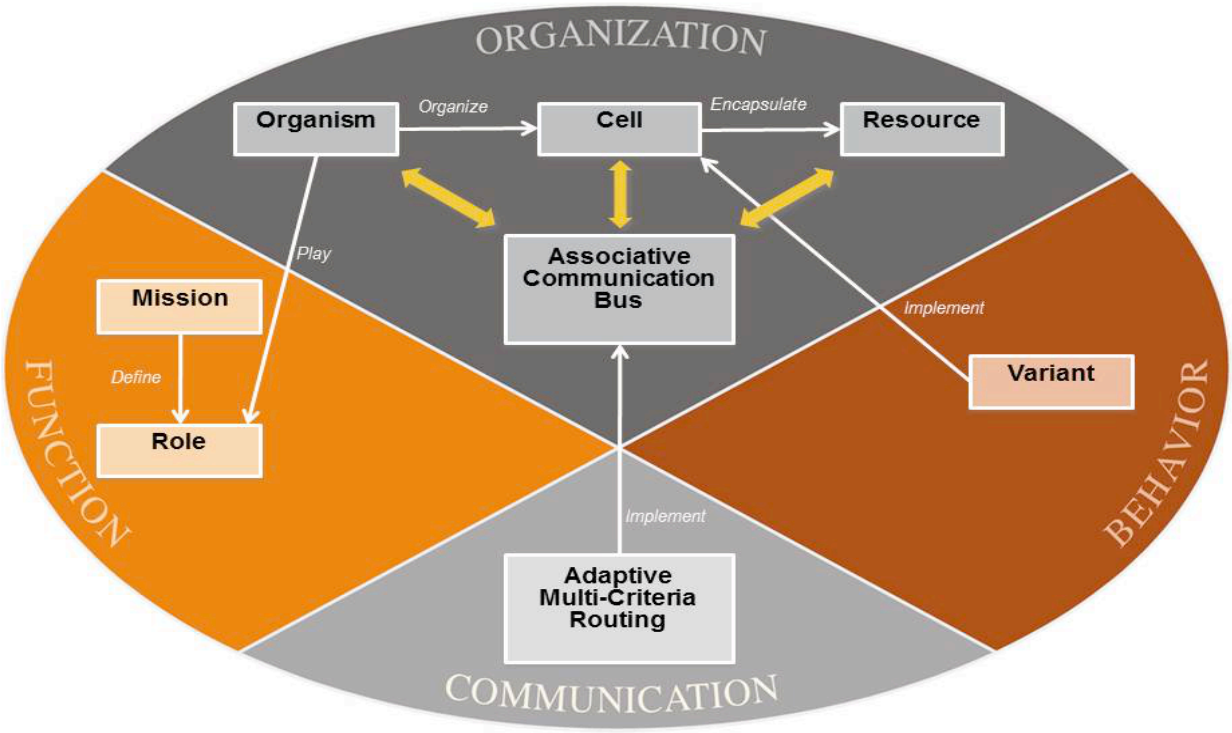

Figure 3.1: Design Elements and Aspects

3.2 Cell-Oriented Architecture

We propose a new architecture called Cell-Oriented Architecture (COA) to support the development, deployment, execution, maintenance, and evolution of NSE software.

Conceptually, the cell is the smallest *active resource* in a distributed computing platform. Cells are instantiated as generic virtual computational units that acquire application specific functionality over-the-air through a process known as *specialization*. NSE physical nodes

can host one or more cells, providing a flexible way to share the physical resources of a NSE node among multiple applications.

A cell can either operate independently as a *unicellular organism* that possesses autonomous existence, or become part of larger structure that resembles multicellular organisms. The software designer views NSE as a set of interacting roles executed ubiquitously on top of a set of role-playing organisms. Inspired by the biological stem cells, the COA cells have the ability to specialize into a specific organism cell that have specific characteristics. Unlike stem cells that can only be specialized once in its lifetime, BioSENSE cells can re-specialize any number of times. Accordingly, COA achieves a high degree of decoupling between functional roles and underlying active resources (cells), by allowing the reassignment of roles at runtime and by providing a convenient and efficient mechanism to dynamically reconfigure NSE nodes. Not only that cells are capable to change their functionality, they can also change their observable behavior performing the same functionality.

COA exploits the separation of data from control to enable platform managed services like replication, failure recovery, and implementation diversity. The runtime binding of cells is facilitated by contracts. Cells use virtual sensors to monitor performance parameters and other phenomena of interest. These monitoring parameters are subject to adjustments/amendments as the mission of the cell changes.

Cells posses autonomous control based on it mission. Missions define objectives, priorities, and mechanisms to quantitatively measure how well are the objectives satisfied, while roles define the tactics to maximize mission objective satisfaction.

Sensors are used by cells to regulate its behavior according to its mission objectives. Cells are also capable of morphing and possessing non-deterministic behavior to confuse attackers and transform NSE software to a moving target.

Cells are formally defined as an ordered 7-tuple of *bootstrap functions*, *contracts*, *ports*, *criteria*, *variants*, *sensors*, and *objectives*. A cell $C_i \in C$, where C is the set of all Cells defined as:

$C = \{(B_c, t, p, r, v, s, j) : ((t \subset T_c) \wedge (p \subset P_c) \wedge (r \subset R_c) \wedge (v \in V_c) \wedge (contract(v) \in T_c) \wedge (s \subset S_m) \wedge (j \in J_m) \wedge (\cup contract(v) = T_c))\}$, where B_c is the set of bootstrap functions that represent the genetic material of the cell, T_c is the set of all cell contracts, P_c is the set of all cell I/O ports, R_c is the set of criteria describing cell capabilities and properties, V_c is the set of implementation variants, S_m is set of all mission sensors, and J_m is the set of mission objectives. The union of all variant contracts should be equal to the cell contract.

Stem cells are generic, unspecialized, cells that are ready to specialize. The stem cell C_s is defined as the tuple $(B_c, \emptyset, \emptyset, r, \emptyset, \emptyset, \emptyset)$, where r is the set of criteria describing the physical resource hosting the stem cell. Stem cells are considered free resources that can be allocated to application or middleware organisms.

3.2.1 Bootstrap Functions

The bootstrap functions of cells are generic inherent operations that all cells must support despite their state of specialization.

specialize(t) Change the function/behavior of the cell according to the contracts specified in the set t, where $t \subset T_c$. The cell automatically discover and loads the code variants that implement contracts in t using universal contract identifiers. Variant loading is explained in details later in this chapter.

invoke(c,m) Invoke method m from contract c.

portConnect(l,b) Connects local cell port l to the virtual communication bus b.

replicate(c) Listen to state transition events from cell c and replicate them.

updateSensors(s) Update the cell's set of sensors to s.

updateObjectives(j) Update the cell's goal to j.

3.2.2 Contracts

Contracts are key elements in COA that separate what cells do from how they are implemented by specifying the causality of data communicated between cell but not the specific cell instances that participate in the communication. Contracts provides a systematic way to verify, monitor, configure, and regulate the interaction between cells. Consequently, contract effectively enables runtime plaggability of cells and automates testing.

The set of cell contracts $T_c = \{(g, p, s, i, c, d) : ((g \subset G_t) \wedge (p \subset P_t) \wedge (s \subset S_t) \wedge (i \subset I_t) \wedge (c \subset C_t) \wedge (d \in D_t))\}$, where G_t is set of all operation signatures, P_t is the set of preconditions for interaction, S_t is the set of postconditions for correct results, I_t is the set of invariants, C_t is the set of internal state constraints, and D_t is the data model of communicated information.

Pluggability

The ability of a cell C to conform to port P if another cell is determined by the predicate:

$signatures(I_p) \subset signatures(I_c) \wedge \forall o((o \subset signatures(I_p)) \wedge (precond(I_p.o) \wedge invariants(I_p) \wedge precond(I_c.o)) \wedge (postcond(I_p.o) \wedge invariants(I_p) \wedge postcond(I_c.o))$, where I_c , I_p are the interfaces of C, and P respectively, signatures(I) is a function that returns the set of all signature in interface I, precond(o) is a predicate that returns true only if all preconditions for invoking operation o are satisfied, invariants(I) is a predicate that returns true only if invariants I are satisfied, and postcond(o) is a predicate that returns true only if all postconditions for invoking operation o are satisfied.

3.2.3 I/O Ports

Cell uses I/O ports as plug points for communication. The cell where a port resides is called the host cell of the port. Two cells are connected by wiring their ports together. When a cell sends data at one of its ports, the port relays the data to the port(s) that connect to it. When data arrives at a port, the cell that owns the port processes the data and may produce output at certain ports according to the contract. In BioSENSE, the wiring of ports is dynamic and managed by the middleware.

The set of ports, $P_c = \{(n, y, t) : ((n \in L) \wedge (y \in Input, Output) \wedge (ct \in T_p))\}$, where L is a regular language defined over the alphabet {a,..,z,A,..Z,_,0,..9} and described by the regular expression $[a - zA - Z\_][a - zA - Z0 - 9\_]*$, d is the port type, and T_p is the set all port contracts.

3.2.4 Criteria

Cells are identified and described using a set of criteria. Each criterion is key/value pair. Values are either boolean, real numbers, or text strings. A stem cell inherits its criteria from the host node. When a cell is specialized it can add application specific criteria to the set. Application can construct criteria-based queries to address cells.

The set of cell *criteria*, $R_c = \{(k, v) : ((n \in L) \wedge (v \in U))\}$, where L is a regular language defined over the alphabet {a,..,z,A,..Z,_,0,..9} and described by the regular expression $[a - zA - Z\_][a - zA - Z0 - 9\_]*$, and U is the universal set.

3.2.5 Implementation Variants

A cell provides its functionality by loading one or more implementation variants. Variants provide implementations to the contracts published by the cell. Functionally compatible variant are shuffled by the cell to exploit implementation diversity. Variant shuffling is explained in details later in this chapter. The variant is the smallest unit of loadable binary code executed by cells. A variant implements one or more cell contracts. A cell provides its functionality by loading one or more components at a time that provide implementations to the contracts published by the cell. Different component versions, that are functionally compatible, are shuffled by the cell to exploit implementation diversity and behavior non-determinism. Variants shuffling is explained in details later in this chapter.

The set of cell *variants*, $V_c = \{(r, u, t) : ((r \in R_v) \wedge (u \in R) \wedge (t \in T_c))\}$, where R_v is set of variant criteria, and T_c is the set of cell contracts.

R_v, is defined similar to R_c, however, the records in the R_v identify and describe implementation variant properties like *version number*, *code maturity*, *code size*, and *trust level*.

3.2.6 Sensors

Sensors provides a systematic mechanism to monitor the environment for phenomena of interest. Each sensor is identified by a unique name. The reading value of a sensor at any instance of time is an application-specific numeric quantification of the phenomena represented by the sensor. Missions define their sensor sets and provide the technique to evaluate and update sensor values. Missions should define the sensors that are used as metrics to evaluate the effectiveness and efficiency of the mission.

The set of *mission* sensors, $S_m = \{(n, v) : ((n \in N) \wedge (v \in U))\}$, where N is a regular language defined over the alphabet {a,..,z,A,..Z,_,0,..9} and described by the regular expression $[a - zA - Z\_][a - zA - Z0 - 9\_]*$, and U is the universal set.

3.2.7 Objectives

The goal of a cell is defined as a weighted set of objective functions. The value returned by an objective function represent how far is the mission from achieving a specific simple objective. Objectives are calculated based on the sensor readings. Application designers build objective functions that return a numeric value representing the logical distance between the current state and the objective state. At runtime, the cells aim to minimize the values returned by these objective functions to get closer to achieving their mission objectives. The weight given to an objective function determines the current priority of the objective with respect to other objectives. For every cell mission, the sum of weights of all objective functions should always be 1.

The set of mission *objectives* J_m is defined as : $J_m = \{(f_i, w_i) : ((i \in 1, .., n) \wedge (f_i \in F_m) \wedge (w_i \in R^+) \wedge (\sum_{j=1}^{n} w_j = 1))$, where w_i represent the weight of the objective function f_i, R^+ is the set of positive real numbers, and F_m is the set of all objective functions. Each objective function, f_{mi} is defined as a binary relations, $f_{mi} : S_m \Rightarrow R^+$, where S_m is the *sensors* set. The relation f_{mi} is defined as $\{(s, v) : ((s \subset S_m) \wedge (v \in R^+))\}$.

3.3 Programming Framework

The BioSENSE programming framework splits the design space into four separate aspects an provides the design elements needed for each aspect as shown in figure 3.1. It exploits COA to enable the construction of NSE applications from loosely coupled elements that are then dynamically regulated and reconfigured to meet any change in the environment, or goals. The framework separates the functional design aspects from the structural aspects. Applications are deployed and managed using an application server as shown in firgure 3.2.

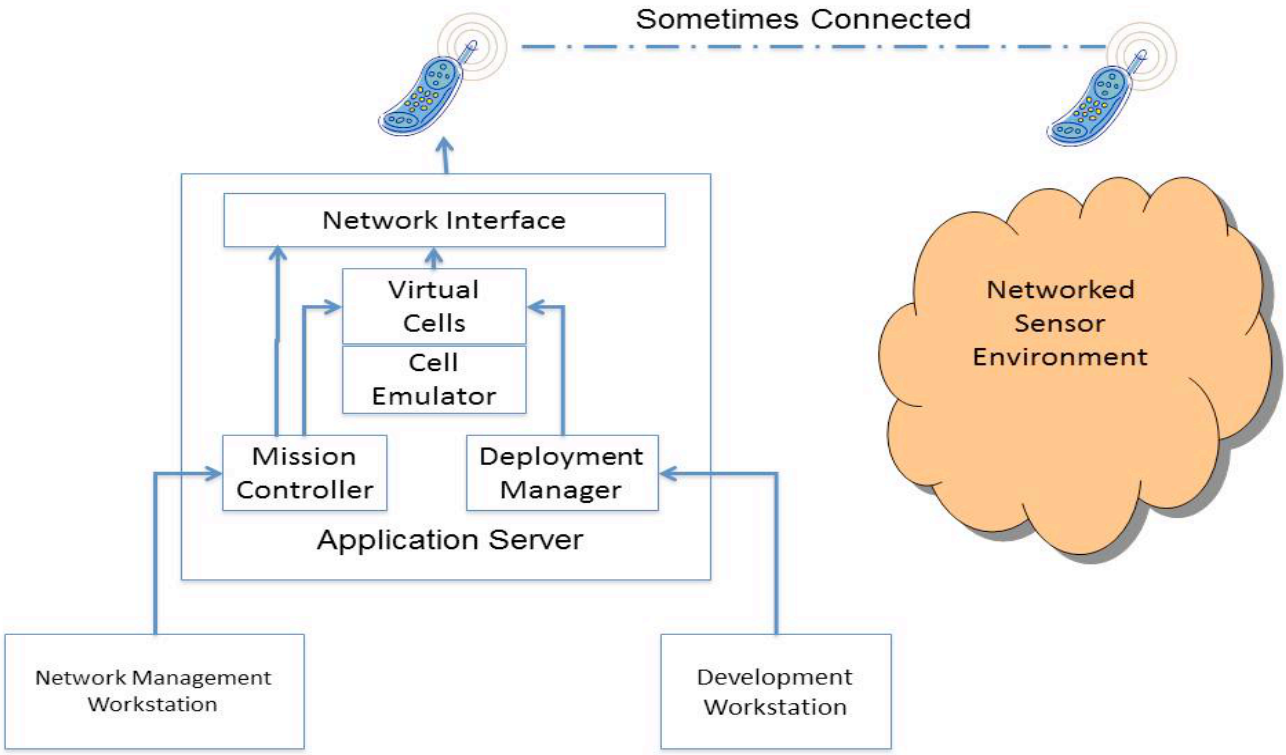

Figure 3.2: BioSENSE Application Server

3.3.1 Function Aspect

BioSENSE adopt a mission-oriented team model for functional decomposition of application responsibilities. The objective of our design is to realize NSE programs organizations that attempts to remain viable despite changing environments and mission goals. We model NSE applications as missions accomplished by co-operating roles, while enforcing strict separation between functions and processes (roles and their players). Roles define responsibilities and plans, while organisms claim the responsibilities and execute the plans. The dynamic role-organism assignment provides functional elasticity and promotes reusability of role-plans. It also promotes structural plasticity of the software components facilitating evolution, allows the application designers to focus on the application logic rather than the underlying network resources, and enables the dynamic reconfiguration of the network when needed to achieve fault tolerance, longer lifetime, and meeting specific QoS requirements. Hence, the functional and organizational integrity of software can be sustained despite the continuous changes in the underlying physical resources of the network.

Missions

A mission is a collection of roles interacting together, over a period of time, to co-operatively achieve a number of weighted objectives defined in terms of measurable phenomena. Missions provide roles a team plan. A mission M is formally defined as 4-tuple $M = (S_m, J_m, L_m)$ where : S_m is the set of *Sensors* monitored by the mission, J_m is an n-tuple of *objectives* pursued by the mission, and L_m is the set of roles that collaborate to achieve the mission objectives.

Roles

BioSENSE exploits the *role* as the concept of partitioning behaviors and emphasizing coordination and cooperation. roles to decouple the organization of role-players from the structure of cooperation processes, such that the mapping between players and roles is not pre-specified. A role is an autonomous software structure that has goals to achieve, memory to preserve (data to store), and ability to learn and interacts with other roles according to predefined contracts. The role characterizes the responsibilities and provides logic patterns to achieve certain goals and cooperate with others.

BioSENSE framework treats the role and collaboration as first class concepts. This architecture promotes software adaptivity which is important for dynamic environments like NSE. It also promotes software reusability and sharing among different applications. The application is viewed as a graph of functional roles executed by players. The binding between roles and players is managed and regulated at run-time. Such collaboration-based design suits self-managed self-optimized distributed tasks like network health monitoring, accounting, resource management, progressive software updating, etc. The collaboration process is designed from a global perspective and largely independent of concrete players so that shifting cooperative behavior does not require changing players. Players can fill multiple roles and switch between them.

The set of mission roles $L_m = \{(i, t, l, n) : ((i \subset I_r) \wedge (t \subset T_r) \wedge (l \in L_r) \wedge (n \subset N_r) \wedge fullfills(i, t))\}$ where I_r is the set of role responsibilities , T_r is the set of role contracts that define how a role interacts with other roles, L_r is the set of role plans that define the role's tactics to achieve the mission objectives, and N_r is a set of role constraints that define the required capabilities a player need to posses in order to play that role. Each role can have one or more contracts to interact with with other roles accordingly.

3.3.2 Organization Aspect

Although individual cells in COA have the necessary characteristics of autonomous role players, a single cell may not possess the capacity or the reliability required by complex and/or mission critical roles. However, multiple cells can be organized into scalable, reliable,

and malleable role players. We call these structures *multi-cellular organisms*. Each organism is a composed of dynamically connected cells.

Organism

An *organism* is an autonomous execution unit and follows the logic patterns that a role provides. Roles are dynamically assigned to organisms. An *organism* is composed of a number of cells wired together dynamically (at runtime) to form a software structure having an independent execution context. The simplest *organism* is composed on only a single cell. A more complex organism may span any number of cells that can be distributed among multiple nodes. The *organism* is the underlying physical structure for the *role* functional element. Accordingly roles can transparently span multiple physical nodes through network-wide execution contexts. This allows limited capability nodes to collectively participate in the execution of complex autonomous roles. Figure 3.3 shows an example of multi-cellular organism that exploits cell replication for enhanced reliability through redundancy.

Organisms are formally defined as an ordered triple of *cells*, *port connectivity matrix*, and *role*. An *organism* $O_i \in O$, where O is the set of all *organisms* defined as :

$O = \{(c_i, x, l_i) : ((c_i \subset ofC) \wedge (l_i \subset L_m) \wedge satisfies(c_i, l_i)))\}$ where C is the set of all cells, x is a port connectivity matrix that defines how cells are wired within an organism, and L_m is the set of all roles.

Organism Role Assignment

For a functional role to be operational, it should be assigned to BioSENSE *organisms*. Organisms are role players composed of one or more cells. The role assignment relation, $assign(l, o)$, is defined as a delegation of role l to organism o. Once the assignment is performed organism o commits to the responsibilities of role r, and the predicate $plays(o, l)$ returns *true*. The role-assignment operation is only possible if the organism capabilities satisfies the role constraints N_r.

BioSENSE maintains a dynamic role-organism map, M_r defined as, $M_r = \{(l, o) : ((l \in L_m) \wedge (o \in O) \wedge plays(o, l))\}$, where L_m is the set of mission roles, and O is the set of organisms. Application developers do not have to care about role-assignment more than just specifying the constraints.

3.3.3 Associative Communication Bus

The Associative Communication Bus (ACB) is a global application-aware generic communication facility that enables interaction among cells using a bus substrate. ACB is used

to wire cell ports within an organism. ACB uses extensible criteria-based addressing and associative routing. ACB decouples cell identification from addressing, allowing transparent failover. Associative routing is explained in details in chapter 5.

3.4 Behavior Aspect

BioSENSE provides applications a number of managed services that induce desirable behavior to enhance resilience, security, dependability, scalability, and adaptability.

BioSENSE middleware provides the following managed services:

Replication

BioSENSE allow systematic state replication of cells at different nodes to support redundancy and logical mobility of cells. To enable system managed replication, BioSENSE strictly separates data from control using a system managed state store. Application's can only change the cell state through sequence-numbered transactions. The described in the cell contract. State transactions are logged by *state transaction manager* in BS-DNA. The records of the binary transaction log can be transferred to a remote remote replica using the *replication/recovery manager*. Replicas keep track of the sequence number of last replicated transaction to resume replication after temporary disconnection. Replication is bootstrapped by serializing the state store. The BioSENSE middleware relies on replication to provide resilient organisms that can survive the death of individual nodes. It also provide a systemic way to move cells from one node to another without loss of collected experience.

Inline Code Distribution

BioSENSE manages the distribution of application code to NSE nodes. When a BioSENSE organism is created, the composing cells are allocated and specialized according to the organism definition script. The the target cells are identified using criteria predicates and the bootstrap operation *specialize(t)* is invoked. It should not be assumed that either the node that invokes the *specialize(t))* or the target nodes have the code required for that cell. The cell specialization request only provides a unique identifier for the cell contract. The specializing cell discovers the locations of different implementation variants of the cell using *code repository* middleware role. Network managers are no longer required to care about code deployment on individual nodes. They only need to inject the code to the network using code seeding virtual cells through BioSENSE application server. The *code repository* is BioSENSE middleware role that relies on the *code manager* and *distributed resource database* in the resource layer of BS-DNA. The code repository manages code versioning and recognize version dependencies among roles to ensure operational integrity during application evolution.

When a node realizes the need for a certain code variant it sends a *Code Request Query* CRQ to its neighbors. The requester node is considered the as a sink for the code serving process. Whenever a node receives a CRQ, it checks its local code repository for the existence of the requested code module.

If the module does not exists in the node's local repository, the node checks its local CRT to know if the module is reachable from that node. If the module is reachable through the nodes neighbors and the current node has enough energy to participate it transferring the module to requester, the node forwards the query to its neighbors after adding itself to the List of Visited Hops (LVH), otherwise it silently ignores the query.

If the module exists in the nodes local repository, the nodes checks if it has enough energy to transmit the module. If not enough energy, it ignores the query, otherwise it considers itself a source and waits for a short period of time (TimeWait) for other copies of the same CRQ that may arrive from different paths, then calculates the optimal reverse path using information in the LVH of each copy of the CRQ. The candidate source then replies back to the sink with Sender Path Notification (SPN). The SPN contains the optimal LVH from the senders perspective and is routed back to the sink only through the nodes in that sender optimal LVH. When a node receives an SPN, it checks if the LVH contains its node id, and if it couldn't find its own id in the LVH it simply drops the requests, otherwise it forwards the request. This guarantees the the SPN is routed through the optimal path to the sink. When the sink receives the first SPN, its wait for short period of time (TimeWait) for other SPNs that might arrive from other candidate sources, then its selects the optimal source according to information from the different SPNs. Then it sends a Start Module Transmission Request (SMTR) to the selected optimal source that contains the optimal LVH announced by the optimal source. The SCTR is routed similarly to the SPN but in the opposite direction. When the selected source receives the SCTR request, it immediately starts to stream module content to the sink through the optimal LVH.

Dynamic Structure Management

Organisms are constructed from cells using an Organ Definition Script (ODS). An ODS script is interpreted by organism *structure manager* component in BS-DNA organism layer. The structure manager provides the following capabilities:

1. Instantiate new cells on NSE resources capable of hosting them.
2. Manage the dynamic mapping of cells I/O ports within an organism in a connectivity matrix.
3. Wire cell ports together to form multi-cellular organisms.
4. Replace a cell in an organ with another replica transparent from applications.

Providing system support for structure management results in more reliable organisms due to redundancy, malleability, and logical mobility. Figure 3.3 shows an example organism with standby replica cells that can be step in-duty if the current active cell failed for any reason.

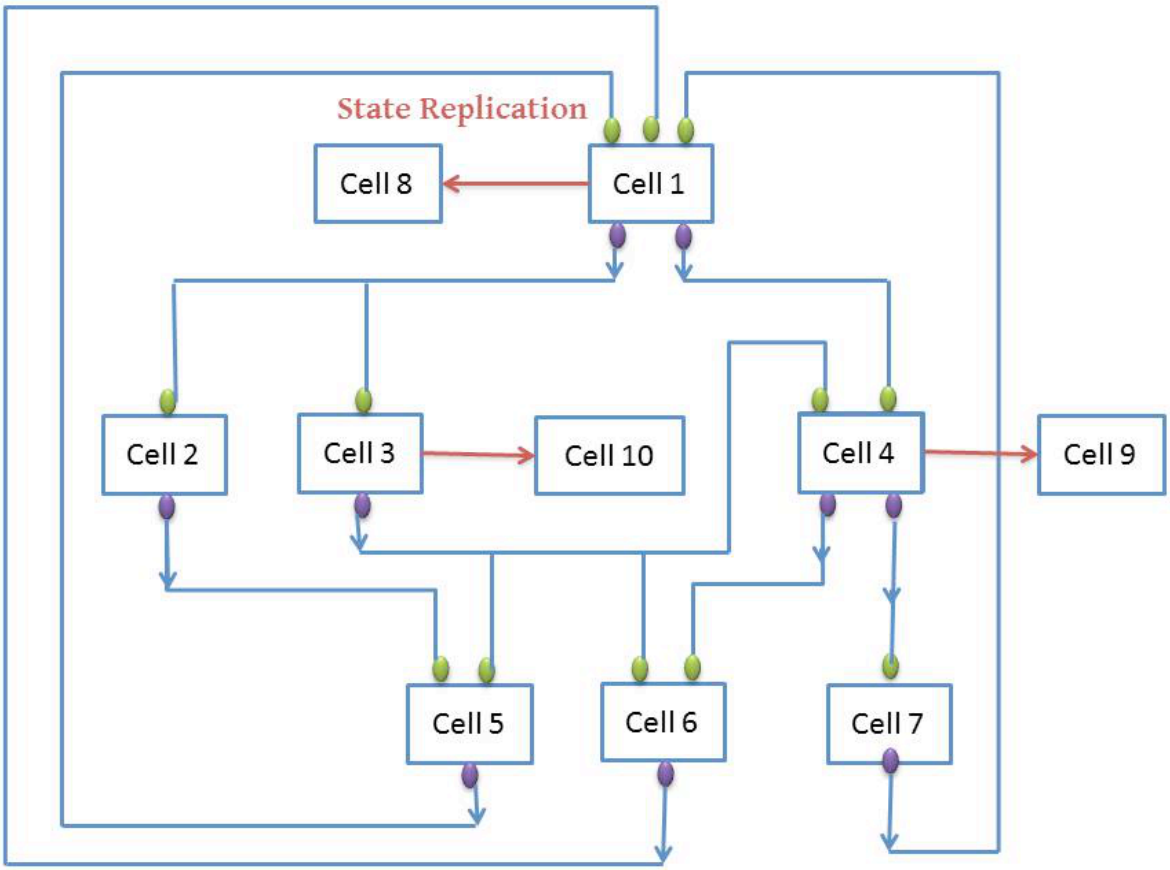

Figure 3.3: Multi-Cellular Organism

Distributed Resource Database (DRD)

DRD is A Distributed Resource Database that stores globally accessible resources. DRD adopts a document-oriented data model, where records are not grouped by their structure but by their attributes. DRD enables in-network data storage for BioSENSE. DRD stores data resources at the nearest *data-store* from the resource owner (data producer). The *data-store* is a BioSENSE middleware role that is played only on nodes that have storage capability and enough bandwidth/energy to participate in serving as DRD servers. DRD

provide a criteria-based RESTful addressing of resources based on associative routing. The criteria indexing used in AMCR routing is exploited to provide document indexing for DRD, so AMCR is used as facility to deliver DRD requests to the appropriate *data-store*. The DRD server provides operations that enables applications to read and modify these resources. DRD resources are unstructured documents that aren't bound to a strict database schema. Each document also has an identification profile of *criteria*, such as version number, document identifier, data of last update, or any application specific criteria. DRD creates a new version of a document whenever it is updated, such that it keep modification history of any document, while allowing old documents to expire using a special *expiration-date* criterion. The inline-code distribution service relies on DRD to store and locate code variants.

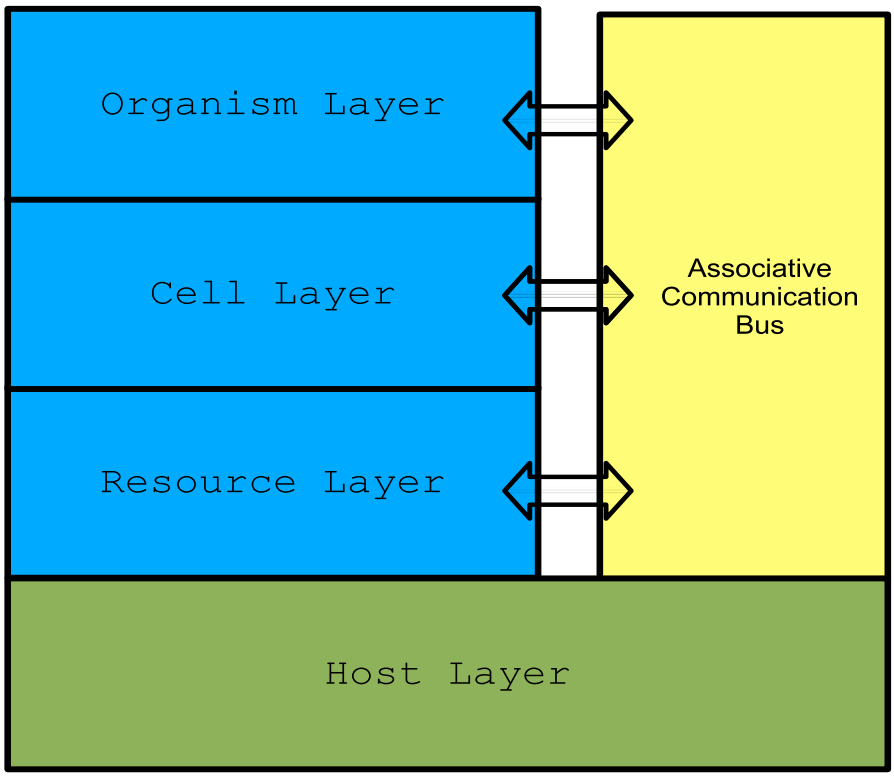

Figure 3.4: Middleware Architecture

3.5 Middleware Design

BioSENSE adopt a micro-kernel middleware architecture, that splits middleware functionality among a minimalistic common kernel installed on all NSE nodes and dynamic middle-ware roles that are selectively deployed at runtime. The middleware kernel, referred to as BS-

DNA, is only concerned with the essential functionality required for a node to participate in BioSENSE like the node's ability communicate, and acquire and execute roles. BS-DNA is the only piece of software that is required to be pre-installed on NSE nodes. All other middleware services are implemented as functional roles. Middleware roles adapt/evolve the same way as application roles. In that sense, the BioSENSE runtime cells are either implemented for all participating node, in BS-DNA, or implemented for only a dynamic subset of nodes, as middle-ware role. Each running role in BioSENSE has an owner application. For middleware roles, the reserved owner *root* is used. The BS-DNA abstracts the physical resources into stem cells, allowing node-level sharing of physical resources among both application and middleware roles as shown in figure 3.5.

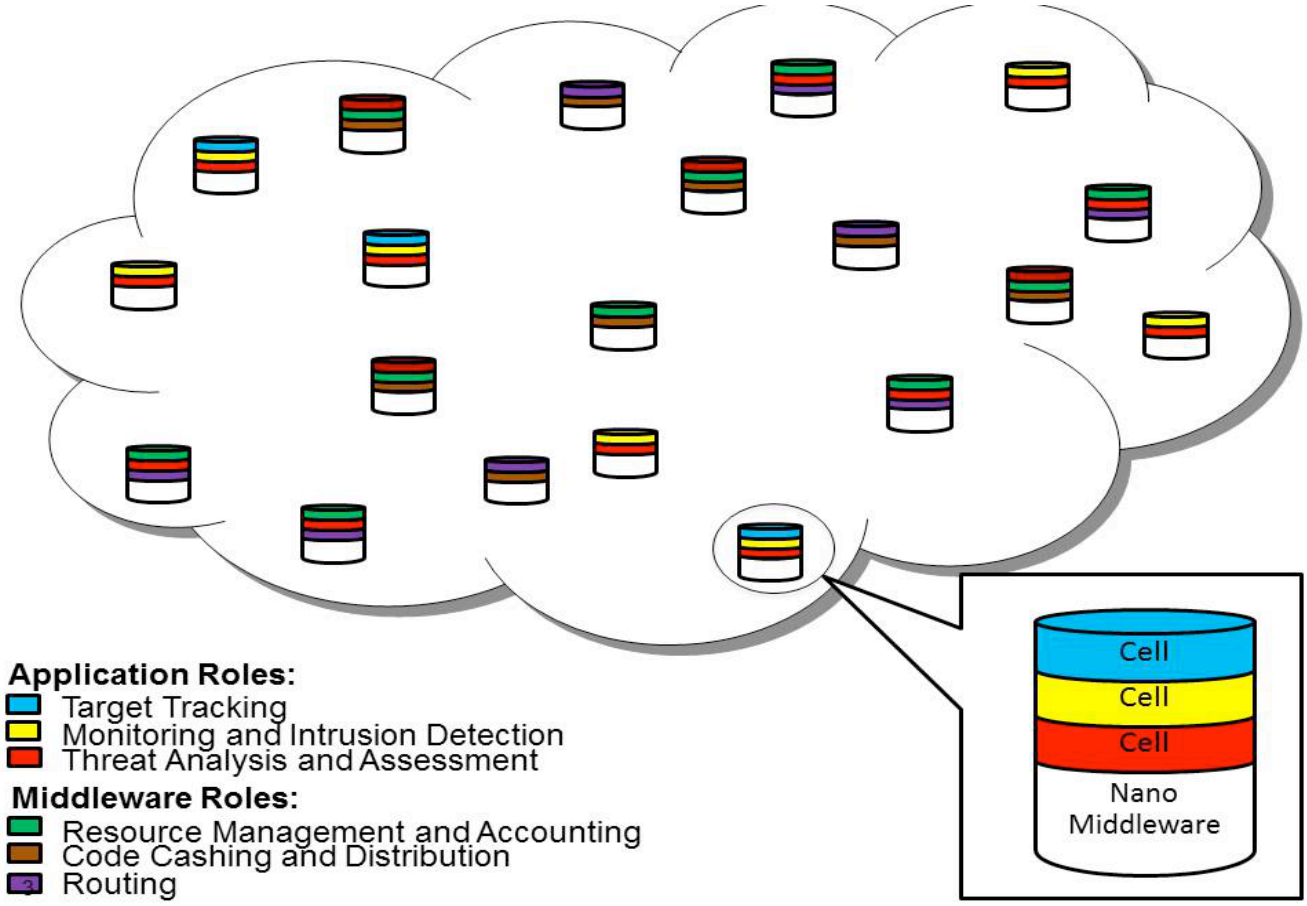

Figure 3.5: Application and Middleware Roles Share Physical Resources

Middleware Roles

The BioSENSE middleware is responsible for providing basic system roles that falls into one of two categories:

Network Oriented Middleware Roles: Roles that provide network global services not related to a specific application. Instances of such roles are typically spawned at the BioSENSE bootstrapping process. Instances of network oriented middleware roles are shared among all applications. This category includes the following roles:

Router: A role responsible for providing a flexible communication facility supporting different interaction patterns within and across applications. BioSENSE routers implement an adaptive multi-criteria routing protocol capable of unicast, multicast, anycast, and broadcast.

ResourceManager: Responsible for allocating, and freeing of distributed NSE resources. At least one instance of a resource manager role should exist in the network.

DataStore: A role responsible for reliably storing data in-network transiently or permanently.

CodeRepository: Provides code storage, discovery, and dissemination services.

Examples: Router, ResourceManager, PolicyManager, and AdmissionControler.

Application Oriented Middleware Roles: Roles that are spawned for each application upon its admission to the network. Instances of such roles are typically dedicated to a single application. This category includes the following roles:

Resource Tracker: Tracks the resources usage of a specific application.

Supervisor: Supports redundancy and fault tolerance. A supervisor monitors other nodes for failure and respawn the application roles, when the nodes die or become inaccessible for any reason.

3.5.1 BS-DNA

The BS-DNA allows the bootstrapping of middleware and application roles through the creation of role-playing organisms and automatic discovery and loading of code at runtime. Due to its highly generic nature, BS-DNA could be factory installed on nodes firmware, so that NSE nodes go from factory to field directly, then acquire their roles and implementation variants while in the field.

Figure 3.4 shows the architecture layers for the BS-DNA. Each node regardless of its capabilities must have the BS-DNA pre-installed to be able to participate in the operation of BioSENSE. The BS-DNA relies extensively on an application-aware generic associative communication plane to provide its managed services. It is composed of the following layers:

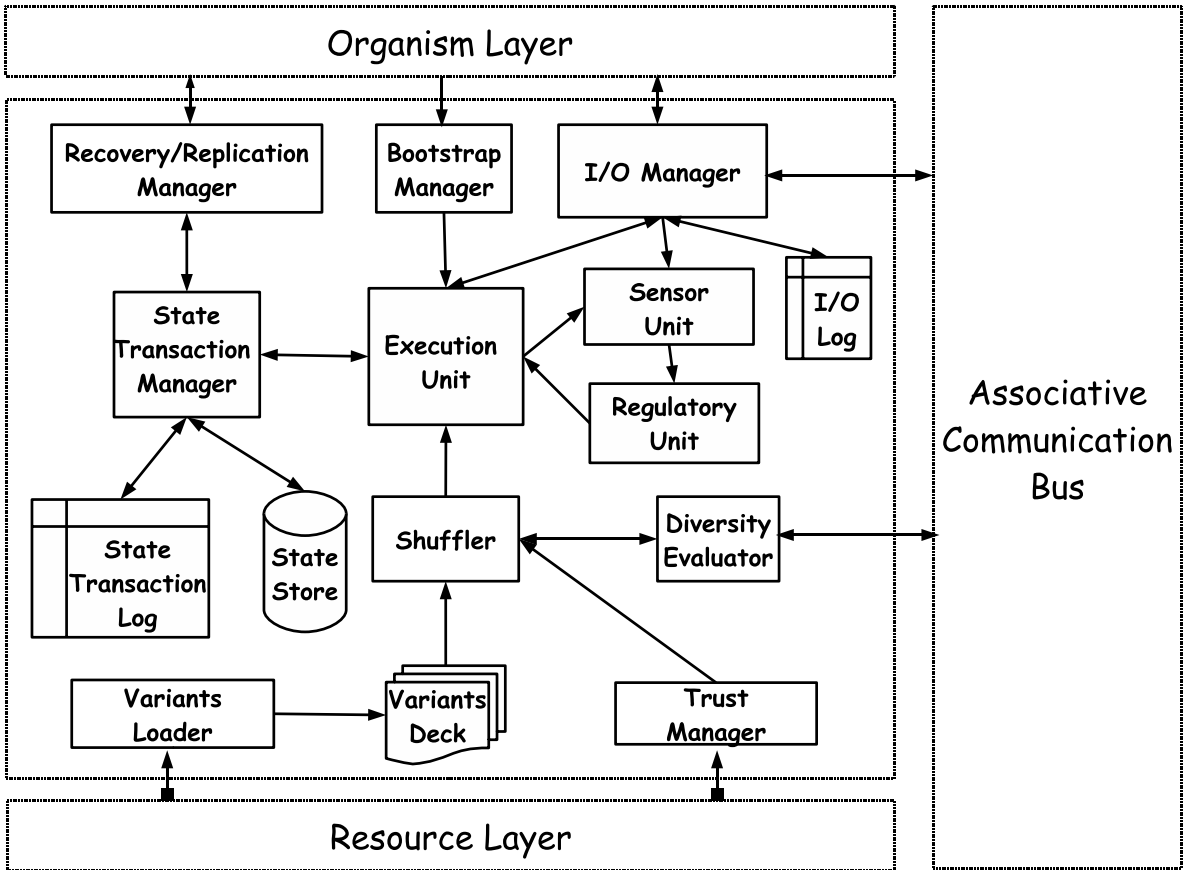

Figure 3.6: Cell Layer

Cell Layer

Provides the runtime support for active cells. The Cell layer BS-DNA is a container for hosting cells at nodes with BS-DNA installed. A node may host multiple cells, but a cell is hosted by only one node. Cells, may be replicated for fault tolerance. Cells do not provide an execution contexts of their own. They provide modular functionality for organisms. Simple organisms maybe composed of only one cell. The cell layer provides isolated execution context for the cell variants and manages the cell's internal state, input/output, and variants deck. The internal state of a cell can only be altered through a transaction manager, allowing the runtime to recover the cell to the last known good state after hitting a implementation flaw. The way cell state is explicitly specified in the cell contract to allow multiple independent variants to access a common state store. Figure 3.6 shows the design of the cell layer.

The major components constituting the cell layer are:

Execution Unit: Provides sandbox isolated execution context to run the cell's implementation variants. The execution sandbox monitors the behavior of running variants, detects failures due to crashes (eg: illegal memory references) or misbehavior (eg: excessive CPU usage). Variants are not allowed to have internal state, although they can register state information with the state store in accordance with the cell contract.

State Store: A managed database that exclusively stores the state information of the cell. The store encapsulates the cell state, maintains state consistency, and tracks state updates.

State Transaction Manager: Provides a safe and controlled way for the implementation variants to explicitly access and manipulate the information in cell's state store. Any update to the cells internal state should go through this component. It also provides a systematic way to rollbacks or revert the cell state whenever instructed by the replication/recovery manager.

I/O Manager: Cells interact through message passing over I/O ports. Wiring/Rewiring of ports is managed by the organism layer. The I/O manager assigns a sequence number for each I/O message. Recent I/O messages are logged in the I/O Log to allow automated recovery.

Shuffler: Loads different variants from the *variants deck* to the execution unit. Shuffles the variants unpredictably to maximize the temporal and spatial diversity. Uses trust evaluator to favor trustworthy variants over others.

Recovery/Replication manager: Manages the state replication of cells. Receives state transition event from the state transaction manager, and propagate the transitions to cell replicas that belongs to the organism Cell pool. Manages state recovery on cell failure.

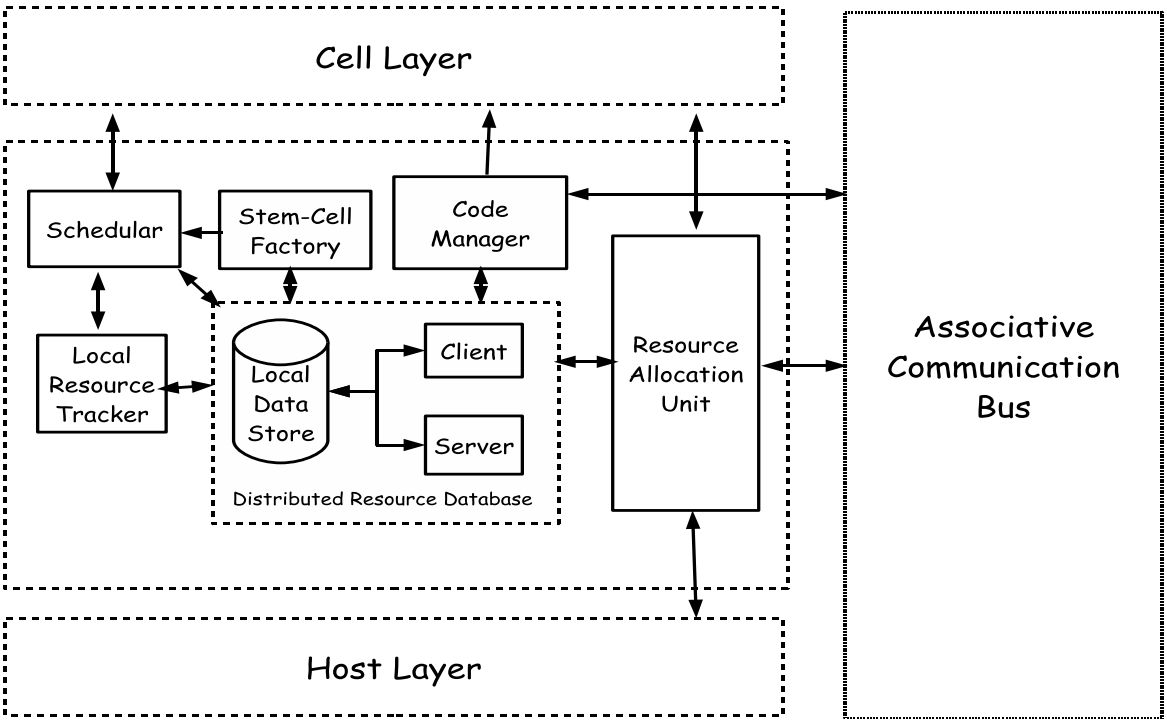

Figure 3.7: Resource Layer

Resource Layer

Manages local resources. Discover and allocate remote resources, including stem cell resources and code variant resources. Performing tracking and accounting for local resource usage, enabling utility-based provisioning of NSE applications. Each resource has an identification profile composed of a set of capability criteria. Capability criteria describe the the hardware and system software capabilities of a physical node in the network. Figure 3.7 shows the design of the resource layer.

The major components constituting the resource layer are:

Scheduler: The scheduler maintains a set of priority queues for incoming traffic, outgoing traffic and service requests invocations. It also maintains a tasks queue. The scheduler controls the dispatching of messages while enforcing limits specified in the Cell ticket of each cell. The CPU limits are eventually enforced by controlling the pace of service request invocations according to the monitored CPU overhead per service request invocation. The scheduler maintains a cpu cost value based on historical observation of the cell. An aging function is used to calculate the cpu cost value. Service requests are non-pre-emptive for lowest scheduling overhead. The scheduler was designed according to the observer/event design pattern. An event is triggered when a cell requests

to send a outgoing message, an incoming message is received, or a timer expired. On each event the scheduler is invoked and performs the following:

- Identify the cell that owns the event.
- Retrieve the limits from the Cell ticket.
- Check the *local tracker* about current resource usage state.
- Decide whether to PASS the event to be processed immediately, or DEFER the event to a later time, or REJECT the event permanantly.

Deferred events are inserted into the task queue and a timer is created delay the event processing. Delayed events occur to account for CPU abuse of a cell that may occur due to non-pre-emptive model adopted. The limits specified in the cell ticket either represent a rate limit or a quota limit. The scheduler is concerned with the communication traffic and CPU limits.

Distributed Resource Database (DRD): Stores globally accessible data. Adopts a document-oriented data model, where records arent grouped by their structure but by their attributes. Uses criteria-based RESTful addressing of resources. DRD stores implementation variants, stem cells profiles, and application data.

Code Manager: Uses DRD to store implementation variants code in-network. Locates/loads code as requested by cell layer.

Stem Cell factory: Produces stem from computational/memory resources. Manages the stem cell profiles stored in DRD.

Organism Layer

Manages the composition of cells into reliable and scalable organisms. It provides structural plasticity and malleability. Figure 3.8 shows the design of the organism layer.

The major components constituting the resource layer are:

Mission Interpreter: Receives and interprets mission commands distributed by the mission manager (part of BioSENSE application server illustrated in figure 3.2).

Role Manager: Manages the change in the organism role and updated the mission objectives and sensors of the composing cells.

ODS Interpreter: Receives and interprets ODS (Organism Definition Script). ODS specifies the organism structure. It defines cells, their contracts, and their bindings(how I/O ports of cells are wired together using bus connections).

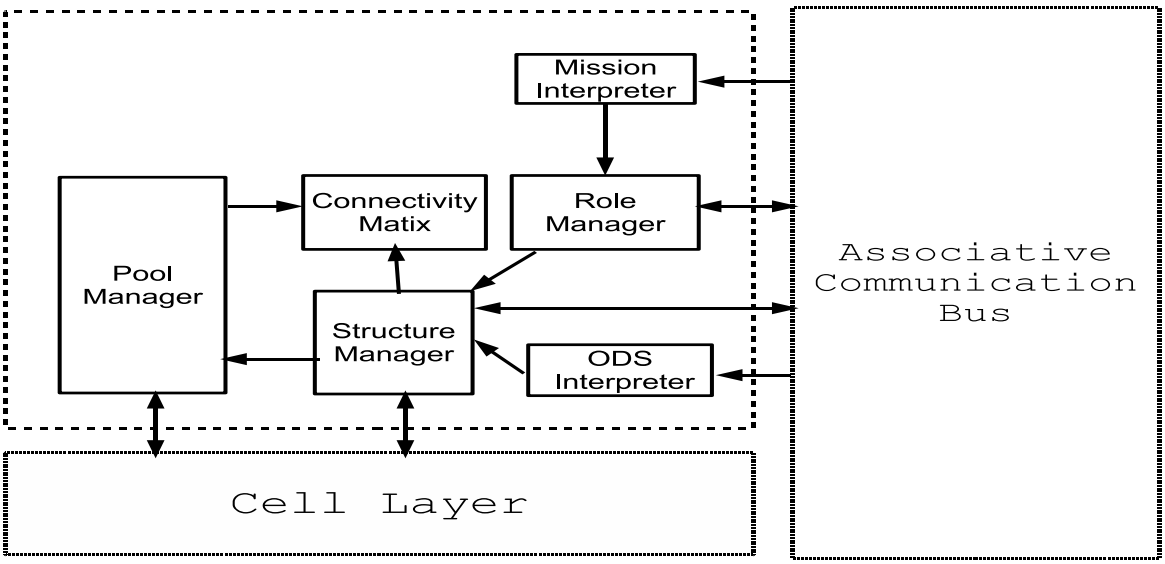

Figure 3.8: Organism Layer

Structure Manager: Manages the binding of cells in an organism, and uses connectivity matrix to model bus connectivity

Pool Manager: Manages a pool of cells assigned to an organ. The pool is dynamically updated as available resources are increased or decreased (triggered by topology changes and node deaths or new deployments).

Host Layer

The host layer provides programming abstraction that hides the details of the underlying operating system and hardware. This layer provides a common host service interface to the resource layer, masking platform heterogeneity. Figure 3.9 shows the design of the host layer.

The complete design of BS-DNA is shown in figure 3.10.

Associative Communication Bus

The middleware implements ACB as a thin layer on top of the associative routing framework. ACB maintains mappings between end points of connections and their identifying criteria then use this mapping automatically generates the destination descriptors for messages to reach intended recipients. Associative routing is explained in details in next chapter.

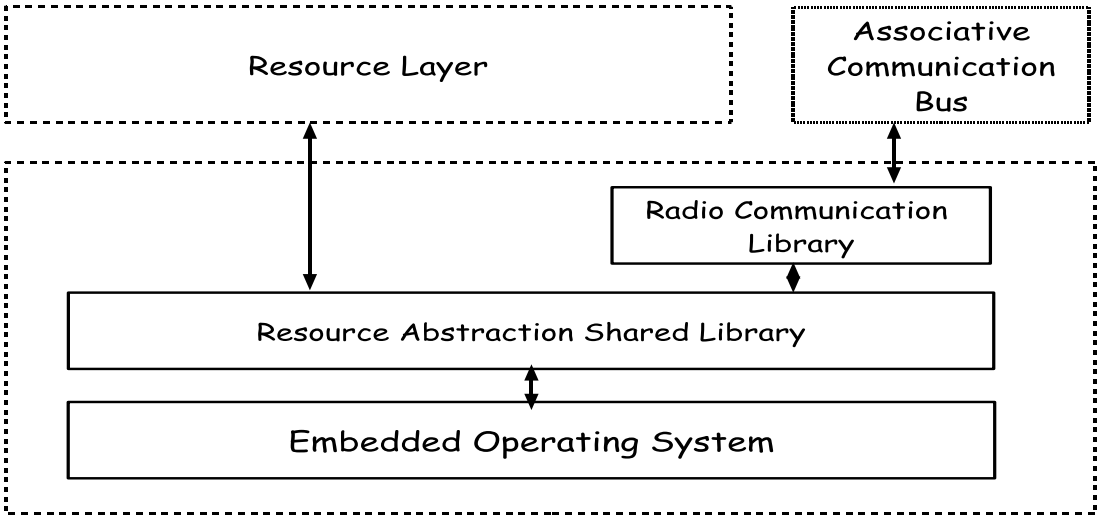

Figure 3.9: Host Layer

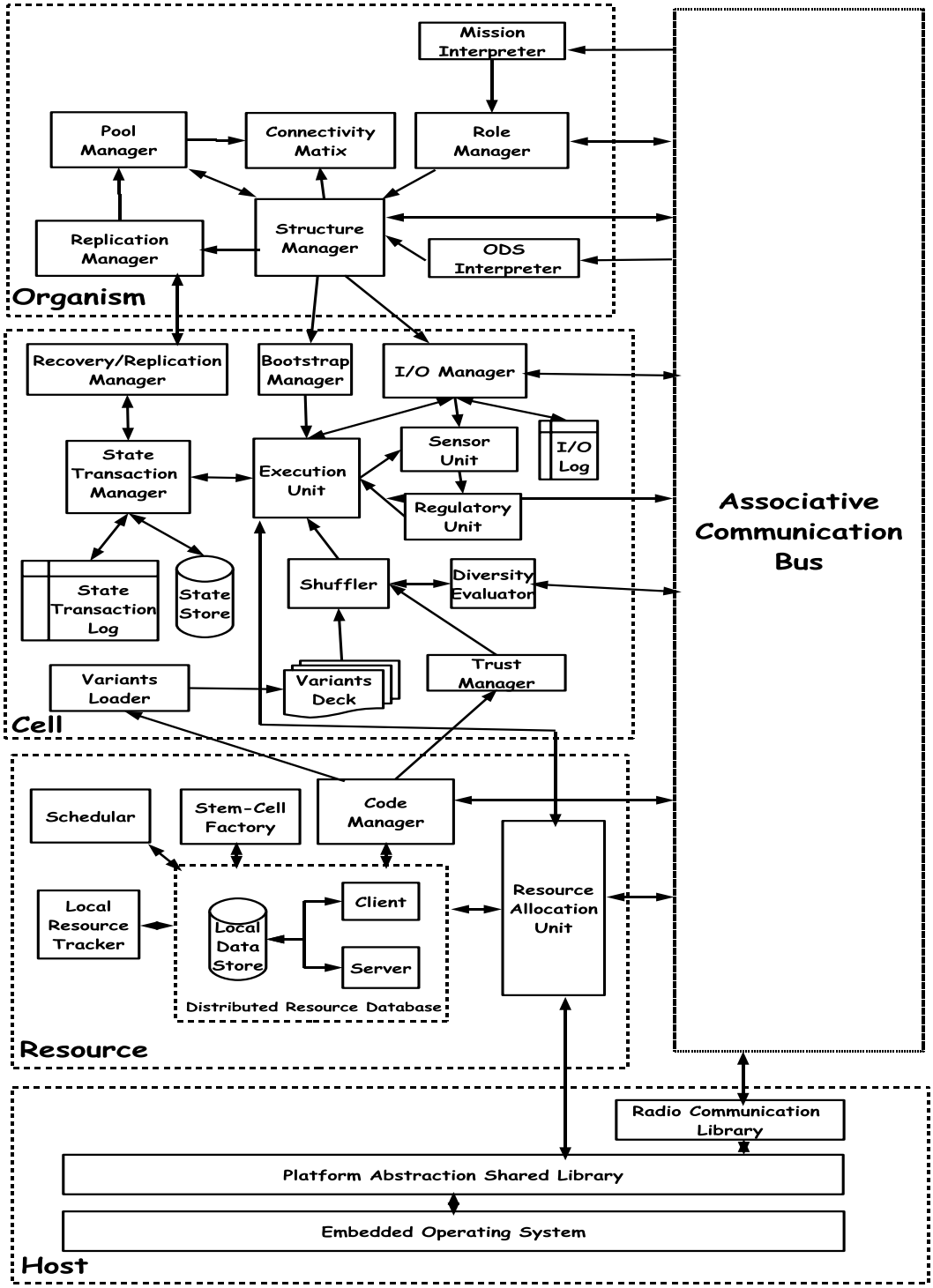

Figure 3.10: NSE Architecture

Chapter 4

Routing In BioSENSE

"Philosophy: A route of many roads leading from nowhere to nothing" .. Ambrose Bierce

4.1 Overview

Traditionally routing in computer networks has focused on finding paths along which data packets could be delivered to pre-identified destination nodes. Most existing routing protocols rely on the use of network addresses as unique node or group identifiers that are usually numeric and independent of any application semantics. The semantically-oblivious identification has forced network designers to incorporate resource/service discovery techniques at higher layers of the network stack, resulting in unnecessary overhead. While such overhead can be tolerated in high-speed wired networks, it significantly limits performance and network lifetime in wireless infrastructure-less networks with battery-powered resource-constrained devices like sensor networks. Moreover, sensor nodes are more naturally anonymous and therefore assigning unique identifiers to individual nodes limits network scalability and imposes significant overhead on resource management.

In this chapter, we present associative routing as a class of routing protocols that enables dynamic semantically-rich descriptive identification of network resources and services. As such, associative routing presents a clear departure from most current network addressing schemes, eliminating the need for a separate phase of resource/service discovery. We hypothesize that since, in essence, resource discovery operates similarly to path discovery then both can be performed in a single phase, leading to significant reduction in traffic load and communication latency without any loss of generality. We also propose a framework

for associative routing and present Adaptive Multi-Criteria Routing (AMCR) protocol as a realization of associative routing for sensor networks. AMCR exploits application-specific message semantics, represented as generic criteria, and adapts its operation according to observed traffic patterns. Analytical results demonstrate the effectiveness, efficiency, and scalability of AMCR.

4.2 Associative Routing

Associative routing replaces the semantics-free destination addresses with semantics-rich *destination descriptors* that dynamically bind to nodes at routing time. Destination descriptors provide qualitative descriptions of the destination nodes where a packet should be delivered. Associative routing does not require nodes to have unique identifiers, instead nodes identify themselves by the services they are willing to provide, resources they are willing to share, data they store, or any other attributes of relevance to applications. Network nodes maintain *identification profiles* that can by checked against *destination descriptors* of network packets by the routing protocol to determine the path along which a packet should travel. The support for associative routing only requires minimal change in the transport layer protocols to allow passing of *destination descriptors* from applications to the routing protocol.

Associative routing is a more generic class of routing protocols, where address-based routing protocols are considered special cases. An address-based protocol, like IP, can be implemented using the associative routing framework by simply exporting the numeric IP addresses in the *identification profiles*, and defining *destination descriptors* to include a single numeric field.

The idea of using dynamic addressing to solve scalability limitations in ad hoc routing protocol was presented in DART [25]. DART distinguishes between node identifiers and their network addresses, allowing nodes to change their addresses as they move around while preserving their identities. It was shown that this level of address indirection in routing later considerably enhanced the routing scalability. Associative routing takes this concept a step further by dynamically mapping attribute-based descriptors to physical nodes without the need for any explicit node identifiers. Unlike DART, there is no direct mapping between the data packet destination specified by the sender and the physical nodes the packet will be delivered to. Multiple different *destination descriptors* may lead to the same node (or same group of nodes). Also, at different times, the same *destination descriptor* may lead to different nodes (or groups) as their attributes change. For example, the battery energy level of a node could be part of its *identification profiles*, so if the energy level dropped, the node disqualifies for a descriptor that it used to qualify for in the past. Such highly dynamic nature of addressing in associative routing makes it an attractive solution to routing in large scale networks of dynamic topologies, like sensor networks. Figure 4.1 illustrates identification and addressing in associative routing.

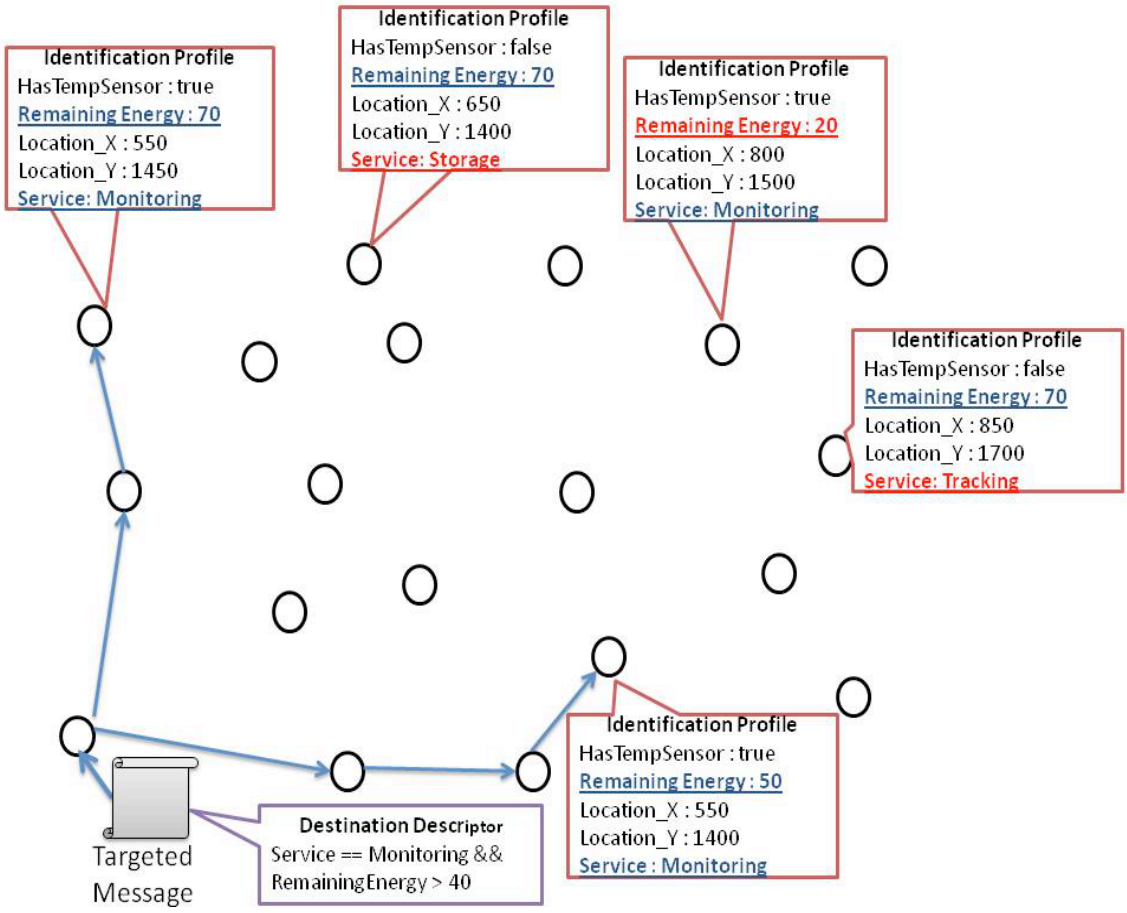

Figure 4.1: Identification and Addressing in Associative Routing

Associative routing provides the following advantages:

Efficiency: Associative routing allows designers to combine the service/resource discovery with route discovery leading to lower traffic overhead, lower communication latency, and prolonged lifetime of networks with battery powered devices.

Dynamic identification and addressing: For large scale networks with dynamic topologies and mobility, static addressing imposes serious scalability limitations. Associative addressing enhances scalability. In addition, it improves the network resilience by allowing transparent fail-over to nodes with similar attributes.

Ability to exploit application layer semantics to optimize routing: The semantics-rich nature of addressing in associative routing allows routing protocols to adapt to application semantics.

Intrinsic multicast: There is no need for nodes to explicitly join multicast groups. Instead,

the sender may specify a destination descriptor that matches any number of nodes at any time, eliminating the overhead of group formation.

Anonymity: Since senders do not need to know or explicitly specify identifiers for destination nodes, routing protocol designers can exploit that to hide node identities if anonymity is required. This also provides a convenient way to manage large numbers of nodes in wireless sensor networks, where nodes are more naturally anonymous.

Associative routing could be effectively used as the underlying routing scheme in data-centric and service-centric networks, since it provides a highly generic way to model and describe destinations. This would result in a cleaner network design in such networks that maintains layer boundaries and separates concerns, unlike current implementations that adopts all-in-one design approach.

Our framework for associative routing adopts a targeted messaging model that allows the exchange of messages between a source node and a targeted group of nodes. The routing agent defines schemas for the *destination descriptor* and the *identification profile*. The source node creates a *destination descriptor* object specifying its targeting criteria. A *Targeted Message* (TM) is a network packet carrying application data in its payload and the *destination descriptor* in its variable-length header. The application agent at the sender side instantiates a *destination descriptor* and configures it according to its targeting needs. The application agent then passed the *destination descriptor* to the transport agent as the destination address. The transport agent constructs a *targeted message* and pass it to the routing agent. The routing agent decides how the packet is forwarded according to the protocol it implements and/or the routing tables it maintains. When a message is received by a routing agent from lower layers, it extracts the *destination descriptor* from the messages and verifies it against the node's *identification profile*. If the profile matches the description in the *destination descriptor* the message is passed to the local transport agent to be delivered to the application agent, otherwise forwarding decision is made. Routing agent may still forward target messages that it matches the local identification profile if it implements multicast functionality. Figure 4.2 illustrates the basic interactions in the associative routing framework.

4.3 Adaptive Multi-Criteria Routing

In this section, we present a fully distributed routing protocol based on the associative routing framework presented earlier. AMCR does not require all nodes in the network to participate in routing or maintain routing information tables. Only a subset of designated nodes play the *routing* role. We refer to nodes that do not participate in routing as *resource* nodes in contrast to *router* nodes. All *resource* nodes are required to be within the single hop connectivity range of at least one *router* node. Each node can determine whether it is

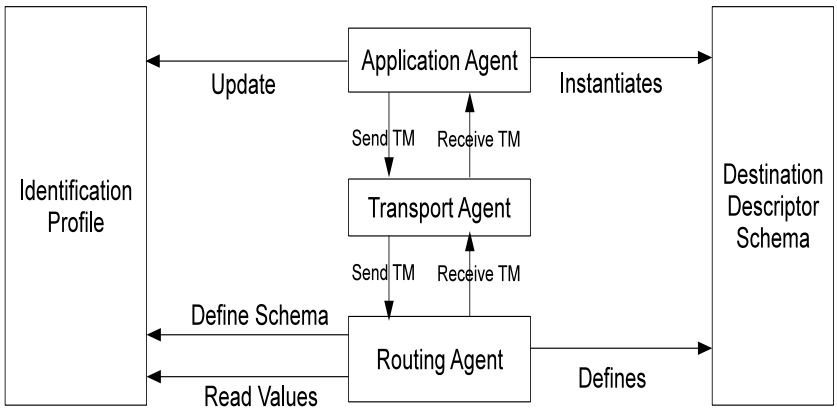

Figure 4.2: Associative Routing Framework

interested to participate in routing or not. Location-aware affinity propagation (LAP) [24] algorithm is used to elect a fully connected set of *router* nodes and establish neighborhood lists. Each *resource* node registers with only one *router* node, reachable in a single hop, that represents its gateway to the AMCR network. The average number of resource nodes per gateway is referred to as *router packing ratio* (rpr). AMCR uses criteria indexes to speed up routing of messages to frequently targeted groups. An Index Working Set (IWS) by *router* nodes containing actively used indexes. Adaptability is achieved through monitoring runtime performance metrics and automatically modify the IWS accordingly. As the application workload or the network topology changes, the IWS changes to include the most efficient indexes for the current workload. Indexes for the most frequently targeted groups are created, while indexes that no longer produce performance or efficiency gains are dropped. The *destination descriptor* used in AMCR is a boolean predicate that we refer to as *Target Predicate* (TP).

The core of the our AMCR approach is implemented in as a *Routing Agent* for the associative routing framework. Each node participating in AMCR based network has one or more *Routing Agent*s. *Routing Agent* is responsible for routing TMs to their destination groups and optionally routing responses back to the source. Figure 4.3 shows the general architecture of *Routing Agent*. This model is highly generic for WSNs as it bases its assumptions on the system level properties of WSN environment rather than on any class of WSN applications. AMCR recognizes the heterogeneity in node capabilities and maintains an identification profile for the WSN node based on its capabilities, spatial location, and other administrative criteria. We refer to this profile as *node profile*. The AMCR's node profile is an aggregation of criteria. Each criterion is key/value pair. Values are either boolean, real numbers, or text strings. A criterion may represent capability possessed by the node, context information, administrative configuration, or application data. Addresses in AMCR are predicates referencing nodes' criteria that evaluate to *true* only for the target node(s). AMCR enables applications to efficiently group resources according to application defined

criteria to support collaboration oriented tasks. Network management components can also leverage AMCR for resource management, fault tolerance, redundancy, optimization, and enforcing quality of service constraints.

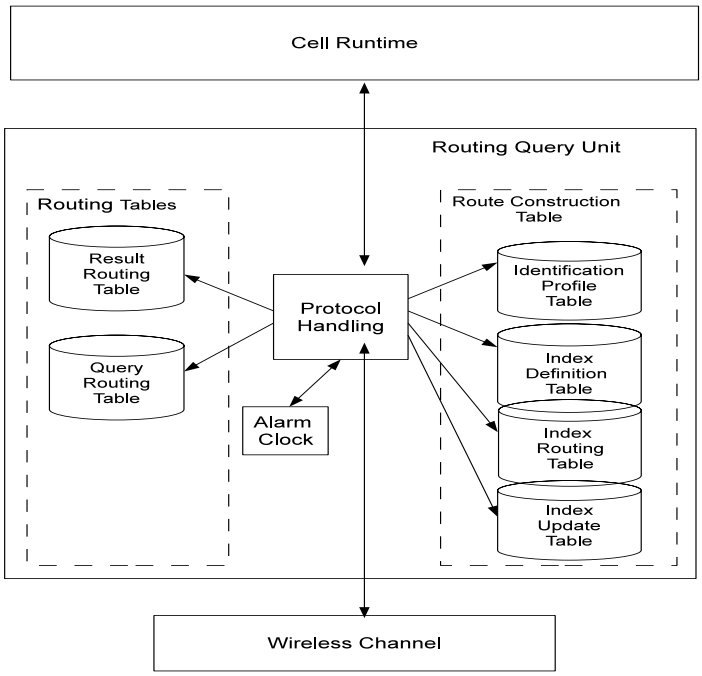

Figure 4.3: Routing Agent

4.3.1 AMCR Design

According to the associate routing framework defined earlier, protocol designers need to define an identification profile schema, destination descriptor schema, and routing agent behavior. In this section we show how AMCR defines the three aspects to qualify as a general purpose associative routing protocol for NSE.

4.3.2 Identification Profile

The AMCR protocol defines a simple criteria-based schema for the node's identification profile. Each record in the identification profile is an ordered pair *(criterion_name, criterion_value)*. The *criterion_name* is a unique identifier that follows the regular expression $[a-zA-Z\_][a-zA-Z0-9\_]*$. The *criteria_value* could be any object that implements the *comparable* interface. Table 4.1 shows an example *identification profile* for a node in BioSENSE.

Table 4.1: Example of Node Identification Profile

criterion_name	criterion_value
hasTempratureSensor	true
location_x	1184
location_y	742
IEEE_MAC_ADDR	“0014.4F01.0000.05E6”

4.3.3 Destination Descriptor

The AMCR destination descriptor is a predicate that reference node's criteria values. The header of the AMCR route construction packets contain target predicates that represents the destination address of the message that can match one or more nodes depending on the criteria values of the nodes and conditions specified in the predicate. Predicates are compiled into a serialized binary format at the source to include in the AMCR packet header, then evaluated at every hop to destination. The AMCR predicates are strings that have the following antlr [61] grammar and lexer rules:

```
predicate_expression : or_expression ( AND_OP or_expression )*;

or_expression : boolean_term ( OR_OP boolean_term )*;

boolean_term : expression COMP_OP expression;

expression : ID | final_expression;

final_expression : multExpr ( ADD_OP multExpr )*;

multExpr : final_atom (MUL_OP final_atom)*;

final_atom : STRING | INT | REAL | LPAREN final_expression RPAREN;

ID : ('a'..'z'|'A'..'Z'|'_')('a'..'z'|'A'..'Z'|'0'-'9'|'_')*;
INT : ('0'..'9')+ ;
REAL : ('0'..'9')+ '.' ('0'..'9')+;
COMP_OP : ('>' | '<');
LPAREN : '(' ;
RPAREN : ')' ;
ADD_OP : ( '+' | '-' );
MUL_OP : ( '*' | '/' );
```

```
AND_OP : '&&' ;
OR_OP : '||' ;
STRING : '\'' ( ~('\'' | '\\') )* '\'';
WS : ( ' ' | '\t' | '\r' | '\n');
```

For example, the following expression is a valid destination descriptor for AMCR:

```
// For nodes with temperature sensor within
// specific area
(hasTemperatureSensor == true &&
 location_x > 1129 && location_x < 3129 &&
 location_y > 2000 && location_y < 3000)
// Example application specific predicate
(role == 'Clusterhead' &&
  remainingenergy > 500)
```

Applications can publish application specific criteria in their cell profiles, then refer those criteria in target predicates. The *Routing Agent* receives the target predicate from application and rewrites it to add implicit QoS preserving conditions. Unicast routing is performed when the predicate is specific enough to match only one node, for example, *IeeeMacAddress = "0014.4F01.0000.05E6"*. Multicast routing is performed when the predicate matches a number of nodes that are all required to respond to the query. For example, *hasLightSensor = true*. Anycast routing is performed when the predicate matches a number of nodes but only a smaller number of the matching nodes are required to receive the message. Specifying a query limit implies anycast routing. Broadcast is performed when the literal *true* is used as the predicate.

Name	Fields
IPT	Criterion,Value,LastUpdated
IDT	IdxName,Properties,Segmented,SegmentSize,SeqNo,Time
IRT	IdxName,Criterion,Value,Time,Gw
IUT	IdxName,SeqNo,Src,Cost,Gw,Time
RRT	Src,Cost,SeqNo,Gw,Time
QRT	Src,PredID,Cost,SeqNo,ResHash,Gw,Time

Table 4.2: AMCR Routing Table Fields

4.3.4 Routing Information Tables

AMCR relies on six tables for its operation. Table 4.2 shows the fields in each table.

Symbol	Description
h	Advertisement Period
u	Index Record Expiration Time
e	Route Expiration Time
c	Path Construction Timeout

Table 4.3: Protocol Configuration Parameters

- `Identification Profile Table (IPT):` Contains the all criteria identifying a *resource*. The LastUpdated field is updated whenever the value of a criterion changes.
- `Index Definition Table (IDT):` Contains the definition of indexes. An index contains one or more criteria from IPT. For multi-criteria indexes, a record is inserted for each criteria in the index with the same *IdxName*. If *SegmentSize* is non-zero, then range of values of the indexed criteria is segmented to limit the size of the index for criteria that have a large number of unique values. The *Time* field specifies the time-stamp of the last time the index was advertised to neighboring routers. Advertisements are sequence numbered to ensure freshness of updates.
- `Index Routing Table (IRT):` Used to route Multicast Route Construction Packets (MRCP) during route construction phase. AMCR avoids uncontrolled flooding while constructing its routes by pruning dead branches early according to records in IRT. The records in IRT expire if the *Time* became older than r time units.
- `Index Update Table (IUT):` Used to temporarily store information about recently received Index Value Advertisements Packet (IVAP). The main purpose of IUT is to handle the receiving of duplicated copies of advertisement packets from the same resource, that may have arrived from different paths to select the best path to use for this resource. The *Src* field indicates the node that hosts the resource, and the *Gw* indicates the neighbor router that forwarded the index update.
- `Response Routing Table (RRT):` Used to route query results back to the query source. The table is propagated with records during route construction. The *Src* identifies the query source. The *SeqNo* field holds sequence number to guarantee fresh routes. The *Time* is used to purge routes that are no longer used.
- `Message Routing Table (MRT):` Used to route messages to their destination groups. The table is propagated with records during route construction phase.

4.3.5 Protocol Description

AMCR consists of three main activities; Index Propagation, Route Construction, and Message Routing. Table 4.3 shows the protocol parameters referenced in this section.

Index Propagation

Each node periodically checks its IDT for indexes that either contain criteria whose value was recently changed in the last h time units or whose *Time* is older than u time units. The matching criteria values are advertised by *resource nodes* to their gateways in Index Value Advertisement Packets (IVAP) that contain the property/value pairs of criteria that belongs to the indexes selected for advertisement grouped by the index name. If the index is segmented the criteria value is rounded to the nearest segment value. The IVAP contains the most recent sequence number for each index. The IVAP also contains the source node identifier and the path costs initialized to the current node identifier and *zero* respectively. When a router node receives an IVAP from a neighbor, it updates its IRT records as follows: if records already exist for that index from that node, the *Value* and the *Time* of those records are updated. If no records match the node/index combination, new records are added to the table. If the sending neighbor is a *router* node *r(j)*, it inserts a record in its IUT for each index in the IVAP associating those records, sets the neighbor value of the IUT records to *r(j)*, and updates the path cost by adding the link cost between *r(i)* and *r(j)*. Each router node constructs aggregated IVAP packets for each of its neighbor and asynchronously propagate those packets every h time units. The aggregation of advertisement eliminates duplicates by selecting updates with least *Cost* and helps minimize the number of IVAP packets transmitted limiting the criteria advertisement overhead.

Route Construction

When a node *n(s)* decides to send a directed message, its MRU sends a Multicast Route Construction Packet (MRCP) to all its neighbors containing the targeting predicate, and a sequence number. When a node *n(i)* receives an MRCP packet, *mrcp*, from a neighbor *n(j)*, it builds a predicate evaluation tree from the destination descriptor predicate expression and checks if the predicate matches its own IPT. If the predicate evaluates to *true*, the matching node sends back a unicast Route Setup Packet (RSP) to the source node through *n(j)*. The *Predicate* is copied from the MRCP to the RSP packet and the *SourceNode* is set to *n(i)*. The RSP packet also contains a *ResourceHash*, which is an integer value that assists in resource ranking. The node calculates the *ResourceHash* according to an application provided hash function that attempts to a calculate unique value for each resource. The function provided reflects the application's ranking criteria of resources. The uniqueness of the *ResourceHash* is achieved by incorporating information like resource location. The reverse path is constructed using the following algorithm:

if $n(i)$ is a *resource node* **then**
  if $predicate(n(i)) = true$ **then**
    $rsp \leftarrow$ new RSP packet
    $ResHash \leftarrow$ calculate unique ranking hash
    send rsp to $n(s)$ through $n(j)$

end if
else if $n(i)$ is a *router node* **then**
$mrcp.Cost \leftarrow mrcp.Cost + link\_cost(n(i), n(j))$
for all $n(k) \in neighbors(n(i))$ and $n(k) \neq n(j)$ **do**
if $predicate(n(k)) = true$ {Using IRT of n(i)} **then**
{update reverse path}
for all $rp \in RRT$ and $rp.Src = mrcp.Src$ **do**
if $rp.Cost \geq mrcp.Cost$ or $rp.Time \leq (now() - e)$) **then**
delete rp from RRT invalidate existing record
else
continue to next neighbor
end if
end for
$rp \leftarrow$ new record in RRT
$rp.Src \leftarrow mrcp.Src$
$rp.Cost \leftarrow Cost$
$rp.Time \leftarrow now()$
forward mrcp packet to n(k)
end if
end for
end if

When a node, *n(l)*, receives an RSP packet, rsp, from *n(m)*, it establishes a forward path as follows:

$rsp.Cost \leftarrow rsp.Cost + link\_cost(n(l), n(m))$
$record\_found \leftarrow false$
for all $qr \in QRT$ and $qr.PredID = rsp.PredID$ and $qr.ResHash = rsp.ResHash$ **do**
if $rsp.Cost < qr.Cost$ **then**
$qr.Gw \leftarrow n(m); qr.Time \leftarrow now(); qr.Cost \leftarrow rsp.Cost$
end if
$record\_found \leftarrow true$
end for
if $record\_found = false$ **then**
$qr \leftarrow$ new record in QRT
$qr.Src \leftarrow rsp.Src$; $qr.PredID \leftarrow rsp.PredID$; $qr.ResHash \leftarrow rsp.ResHash$; $qr.Cost \leftarrow rsp.Cost$
$qr.Gw \leftarrow n(m)$
$qr.Time \leftarrow now()$
end if
if $n(m) \neq rsp.DestNode$ **then**
for all $n(k) \in neighbors(n(l))$ and $n(k) \neq n(m)$ **do**
if $\exists rrt \in RRT$ where $n(k) = rrt.Gw$ and $rrt.Src = rsp.DestNode$ **then**

```
        forward rsp to n(k)
      end if
    end for
  else if predicate limit exists then
    LimitCutoffValue ← calculate cutoff value
  end if
```

The cutoff value is calculated sorting the RSP responses in descending order of *ResHash*, and using the *ResHash* value of the l_{th} response. The destination nodes wait for c time units before calculating the *LimitCutoffValue* to allow any delayed responses to arrive. Redundancy of resources and paths is achieved by selecting the top ranked routes.

Message Routing

After path is constructed the source node sends the message to its gateways to be routed to its target group. When a router node receives a message, it checks its QRT for the set of records matching the message *PredID* and have a *ResHash* greater than or equals to the *LimitCutoffValue*, then forwards the message to all the neighbor nodes specified in the *Gw* field of the matching records. When a destination node receives a message that requires a response, it processes the message by invoking the application level handler and unicasts the result back to the source. When a router node receives a result packet from a neighbor, it checks the RRT table for a record whose *Src* matches the *DestNode* of the result packet. The router node forwards the packet to the neighbor node indicated by the *Gw* field of the matching record. This is repeated till the result packet arrives to the source.

The performance evaluation of AMCR is presented in next chapter.

Chapter 5

Moving Target Defence

"When you aim for perfection, you discover it's a moving target" .. Geoffrey Fisher

5.1 Overview

A moving target software platform is capable of morphing and possessing non-deterministic behavior to confuse attackers, reduce attack surface, and induce resilience under hostile operation conditions. BioSENSE achieves Moving Target Defence (MTD) through implementation diversity and co-operative trust-based shuffling of software component implementations. The continuity of operation is improved by providing means to gracefully recover from implementation flaws at runtime.

Diversity through shuffling is highly effective to slow down automated attacks like worms. In networks with no software diversity, the compromise of a single node most likely leads to the compromise of the whole network since same attack can be repeated successfully for all nodes at all time. In BioSENSE, if a nodes is compromised, the same attack is unlikely to succeed on different nodes or at different times due to software shuffling.

BioSENSE middleware attempts to systemically impede behavior discovery of NSE software implantations and mitigates exploiting vulnerabilities resulting from implementation flaws. Consequently BioSENSE reduces the attack surface and improve the security and resilience of the system. BioSENSE middleware encapsulates and autonomously manages semantically equivalent software component variants with dissimilar implementation behavior. BioSENSE is designed to enable trust-based hot shuffling of implementation variants at software component level to maximize the spatio-temporal software diversity in NSE. It also provides a

systemic mechanism to exploit software redundancy to automatically recover from failures resulting from implementation flaws. BioSENSE achieves temporal diversity through independent shuffling of implementation variants such that the observed behavior of the system is different at different points of time independent from other nodes. On the other hand, spatial diversity requires communication among nodes to co-ordinate their implementation shuffling such that nodes close to each others tend to have different implementation at the same time instance even if they play the same role in the network.

5.2 Behavior Non-determinism

Normally, computer software exhibits deterministic behavior, even at the microlevel. This results from the fact that a computer program executing a specific function with specific system state and parameters will always executes the same sequence of machine instructions. Such determinism makes it easier for attackers to exploit implementation flaws leading to security vulnerabilities. With the increase of popularity of open-source software, a wide variety of production systems are running software whose source code is available to public. The discovery of an implementation flaw in a newly released open source software, is likely to put thousands of production systems at the high risk of being exploited by malicious attackers till the system maintainer patches the buggy software. Systems that attempts to introduce non-determinism in the software behavior are likely to be more resilient to a wide range of attacks that attempts to exploit implementation flaws. Non-determinism can either be achieved statically or dynamically.

Static Non-determinism In this type of behaviour non-determinism, each instance of a software system, or a component, is created differently from other instances in a non-deterministic way, however, the behavior does not change at run-time. Static non-determinism can be achieved at system building, and/or process creation time. Since static non-determinism does not require any runtime support, its impact of overall system performance is minimal. The tools required to support static non-determinism are compilers, linkers, and/or loaders.

Dynamic Non-determinism In this type of behaviour non-determinism, software systems, or components, are capable of changing their observed behavior at runtime. This provides higher level of attack resilience that static non-determinism. Dynamic non-determinism require runtime support from computing platforms, virtual-machines, and/or middlewares. Such runtime support leads to a higher performance overhead compared to static non-determinism.

We classify bahavior non-determinism according to the granularity at which non-determinism is introduced into three classes: 1) instruction-level, 2) component-level, and 3) system-level.

5.2.1 Instruction-level Non-determinism

Instruction-level non-determinism refers to a class of behavior non-determinism where the same high level logic is non-deterministically mapped into different linear sequences of machine instructions. This class of non-determinism can be fully achieved from a single implementation of the software in high-level programming language using non-deterministic compiler. This non-deterministic compilers are only different from normal compilers in the code-generation phase. The code-generator can convert the same abstract syntax tree into different, functionally equivalent, machine code templates. Each time the compiler is invoked it chooses the code-generator template randomly, so that the attacker who have access to the source code of the system can not completely predict the micro-level behavior of the software at runtime.

5.2.2 Component-level Non-determinism

Component-level non-determinism refers to a class of behavior non-determinism where the same software component specification is non-deterministically mapped into different component implementations. This requires programmers to provide alternative implementation of the same component in high-level language then non-deterministically select one of the implementations at a time. This class of non-determinism extends instruction-level non-determinism to allow high-level algorithm and data-structure aspects to posses non-determinism in addition to the low-level non-determinism in the machine instruction sequences. Component-level non-determinism can be realized statically using application builders that can non-deterministically select one implementation among multiple available implementation when linking application models. It can also be realized dynamically using run-time environment that are capable of non-deterministically rewiring components at run-time. Component-level non-determinism requires a single consistent component design for all implementations to adhere to.

5.2.3 System-level Non-determinism

System-level non-determinism refers to a class of behavior non-determinism where the same system specification is non-deterministically mapped into multiple standalone implementations of the system. Like, component-level non-determinism, this class requires that programmers provide alternative implementations of the same system specification. Unlike component-level non-determinism, different system implementations do not need to have the same component structure and may have major design differences. Each implementation is self-contained without any restrictions on its internal structure or code-organization. Static system-level non-determinism is achieved by simply selecting one of the alternative implementation at deployment time and does not require any platform support. On the other

hand, dynamic system-level non-determinism can be achieved using a run-time platform that can non-deterministically route requests to one of the available system implementation. For example, multiple different implementation of webserver software behind a non-deterministic load-balancer.

5.3 Moving Target Platform

BioSENSE realizes an integrated platform that systemically exploits implementation diversity, build diversity and runtime shuffling to improve the security and fault tolerance of software system by mitigating vulnerabilities resulting from implementation errors in NSE software, and introducing non-determinism in observed behavior to confuse attackers. BioSENSE also provides the mechanisms automatically recover from faults encountered at runtime improving both availability and security of NSE software. Figure 5.1 shows the principles that made robust moving target software possible. We describe each of the principles in the following subsections:

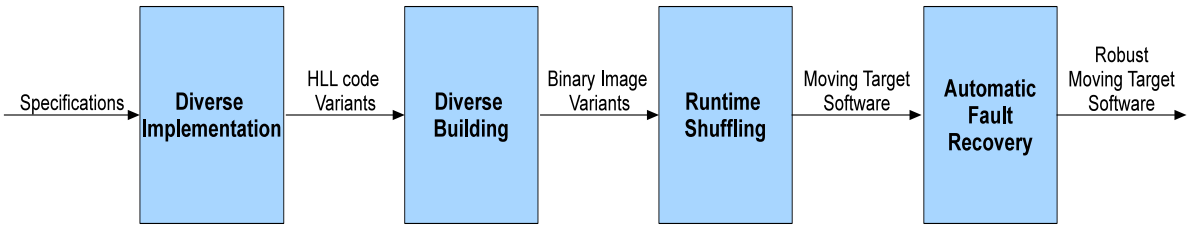

Figure 5.1: MTD Principles

5.3.1 Implementation Diversity

The development of secure, robust software systems is highly challenged by implementation flaws that could lead to serious security vulnerabilities despite careful auditing and analysis of the software specifications. The implementation flaws are usually difficult to detect and fix before the system is operational due to the prohibitive cost of auditing the implementation or testing all possible inputs and system states. Anonymous intruders could exploit implementation flaws to gain partial or total control of the system, or cause the system to crash leading to undesirable downtime. Implementation diversity is the technique of using multiple diverse implementations for the same specifications. It has long been recognized as an effective way to improve the robustness of computer systems by coping with residual software faults. In N-version programming (NVP)[5], multiple versions of a program are

independently developed by different programming teams. N-version relies on the independence in the development of program versions to achieve diversity. In NVP, multiple versions run concurrently and their results are compared by an adjudicator that usually uses majority vote to select the result assumed to be correct. BioSENSE exploits software redundancy to improve NSE resilience and introduce behavior non-determinism to confuse attackers.

There are two approaches to have multiple variants of software systems:

Independent development by different teams The concept of using multiple independently developed versions of the same specification was first introduced by n-version programming[5]. Since its introduction in 1977, n-version programming has attracted the interest of research that aims to enhance the dependability of software systems, however, the high cost of multiple implementation is a major drawback for this approach.

Machine generated Recent research attempted to produce multiple implementations with high degree of variance from formal representation of specification.

Software diversity is a multi-dimensional concept. In an attempt to define and formalize the concept of software diversity, four aspects of diversity were presented in [47].

Structural diversity the structural differences among variants.

Fault diversity the differences between the faults found among variants.

Tough spot diversity the differences in fault-proneness among the elements of variants.

Failure diversity the differences in the failure behavior among variants.

Some of the factors that affect the diversity metrics include:

- Differences in programming language
- Differences in design methodology
- Differences in libraries and frameworks used
- Differences in data structure used

For each factor, a square matrix can be used to represent diversity in a component. Table 5.1 shows an example of diversity representation of a software component with four implementations.

BioSENSE evaluates the diversity of cells when variants are deployed, providing system maintainer an estimate about the expected level of reliability the system can achieve. At

Table 5.1: Example of diversity feature matrix

	V1	V2	V3	V4
V1	0	1	0	1
V2	1	0	1	1
V3	1	0	0	0
V4	0	1	1	0

runtime, BS-DNA keeps track of the number of failures encountered and the number of successful/failed automatic recovery attempts. Software failures, detected at runtime, are automatically recoverable unless all available implementation variants of the cell fail to the same transaction. Accordingly, the more implementation variants we have for a cell the less the probability of its failure. Basically, assuming all variants have the same probability of failure p, then the probability of cell failure would be p^v where v is the number of variants. We expect the point of diminishing return to be highly dependent on the user's security/robustness requirements and what the user would consider an acceptable probability of failure for the implementation budget. We expect the number of implementation variants to increase linearly with the size of the code base. Assuming, all cells are of same size, and all cells have the same number of variants, if a system has n cells, then the code base size s can be represented as $s = cn$, where c is a constant representing the size of a cell. The total number of variants t would be $t = vn = (\frac{v}{c})s$, where v is the number of variants per cells.

Build Diversity

Build diversity is achieved when the application building tool-chain is capable of producing diverse binary images from the same high level language implementation. These tool chains maintain multiple code generation template variants for the same high level language construct. When applications are built from high level language into a platform/architecture specific binary images, the build tool-chain randomly select one of the available code generation template variants to include in the binary image produced. Build diversity enables instruction-level non-determinism, reducing the probability that an attack, designed for a specific image, succeeds.

Runtime Shuffling

We define the term behavioral state as the unique configuration of the on-duty binary-image variants of system cells resulting in a unique overall implicit behavior for the system. The higher the number of different behavioral states in the system, the higher is the potential for non-determinism. Assuming a cell-oriented design, we show that it is possible to achieve

a large behavioral state space in a cost-efficient manner using runtime shuffling. Runtime shuffling allows the morphing of the software image, and consequently the implicit behavior of the system, during the lifetime the process. The architecture of the runtime environment that supports shuffling is presented in figure 5.2.

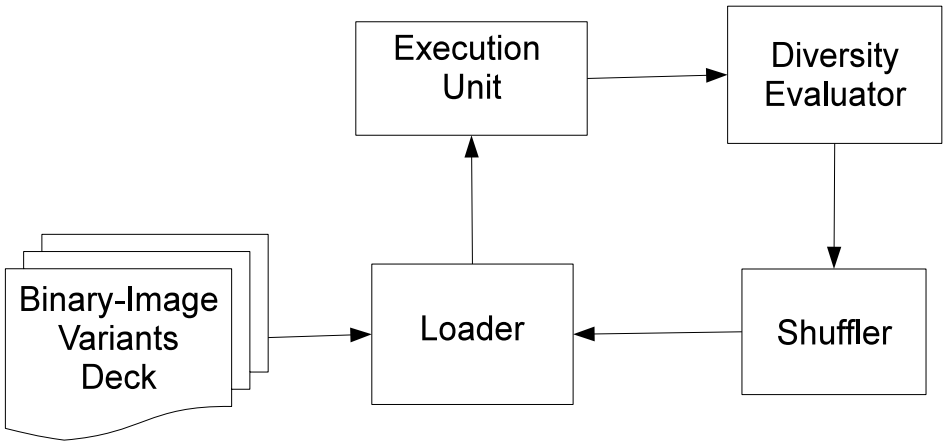

Figure 5.2: Runtime Architecture

The shuffler component generates shuffling events, and send them to the loader that replaces the on-duty binary-image variant with another variant as specified in the shuffler event. The shuffling algorithms are classified according to the shuffler's event generation pattern into the following:

Periodic shuffling Shuffling events are generated at fixed intervals, based on user-defined fixed rate.

Random shuffling Shuffling events are generated at randomized intervals following user-specified probability distribution function.

Event-driven shuffling Shuffling events are generated when specific condition is evaluated as *true*. The user specifies the condition and the shuffler monitors the condition on each state transition.

Application-initiated shuffling Shuffling events are generated when explicitly requested by the running variant. For example, application can handle unexpected errors by explicitly calling the shuffler to restart the current transaction on a different variant. The application only determines the time at which shuffling events should be generated, while the shuffler decides which variant should be used.

The following BioSENSE features are essential to enable variant shuffling:

- Separation of data and control.
- Contract-based interactions
- Contract-based internal state transitions.
- Transactional state management

Given that all different variants of the same cell are functionally equivalent as dictated by the specifications, by random independent shuffling of cell variants, we achieve a higher number of behavioral states while maintaining the same overall explicit behavior of the system. The number of behavioral states of a BioSENSE organism o is equal to $b_o = \pi_{c=1}^{n_o} |W_c|$, where n_o is the number of cells in organism o, and W_c is the working set of cell c.

Normally, a single operation may not involve all cells in an organism. Usually, only a subset of cells along a certain path is involved in the execution of any specific operation. From the perspective of executing a specific operation, this path is called the operation's execution path, and the cells residing on it are called the operation's active set. We define the perceived behavioral state for a specific system operation as the unique configuration of the on-duty variants of system cells belonging to the operation's active set. Accordingly, the perceived behavioral state of an operation is only affected if the shuffling step taken by the system involves at least one of the cells in the operation's on-duty set.

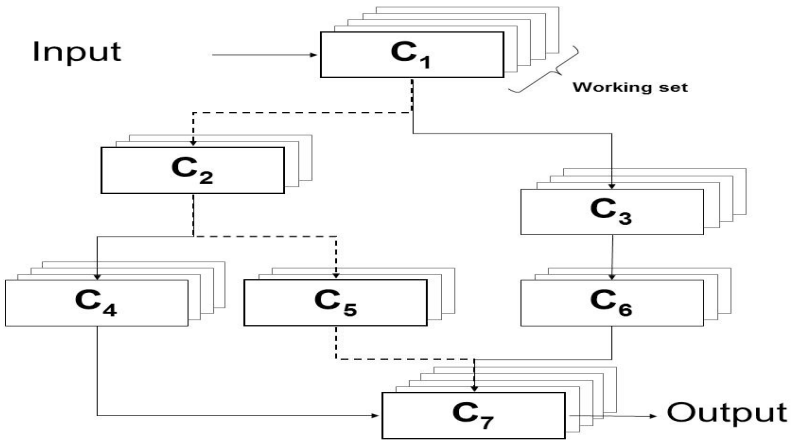

Figure 5.3: Example Organism Diversity

In the organism shown in the Figure 5.3, we assume an operation whose execution involves the on-duty cell set $\{C1, C2, C5, C7\}$, as indicated by dashed lines. Without shuffling, the variants used for the cells in the active set never change. Thus performing the same operation will always induce the system to show the same implicit behavior. However, with variant shuffling, the variants selected by the system for the active set are – with a high probability – different each time the same operation is performed. For instance, if at time

t_1, the system had the on-duty variants $\{V1, V2, V5, V7\}$ for the shown active set, then at time t_2 the on-duty variants for the active set might be $V'1, V'2, V'5, V'7$, where the two sets are equal only with a very low probability. Switching between variant sets does not affect the system functionality, nevertheless, inherent differences across the various variants of the same cell will result in a different overall implicit behavior for the system. To ensure sufficient diversity among variants, the specifications given to different developers should have different development constraints. For example, different developers maybe required to use different libraries, compilers, or even different programming languages.

5.3.2 Automatic Fault Recovery

BioSENSE provides automatic recovery from software faults that may be encountered at run-time. Each cell runs in its own isolated sandbox and owns one or more connectors, called ports, for communication with other cells, as explained earlier. The sandbox helps in detecting common software faults, like illegal memory references, deadlocks, or excessive resource usage, ..etc. When an implementation flaw is detected, BS-DNA rolls back the state of the cell to the last known good state, notifies all components that received input from that cell to roll back their states as well, replaces the working implementation with another one from the implementations deck, and replays all the input received by the cell since the last known good state.

When a cell receives data on one of its input ports, the I/O manager assign a transaction sequence number to the input and logs it to the I/O log before passing it to the execution sandbox. The received input triggers the execution of a user supplied handler that processes the input and may modify the cell's state through the transaction manager. The transaction manager logs the state updates to a transaction log, then only commit the transaction when the handler completes its processing of the input and no flaws were detected by the sandbox. Committing a transaction is considered a state transition from S_n to S_{n+1}. The transaction manager is able to revert the cell state to any previous state identified by the transaction sequence number. As part of its execution, the handler may generate output to be sent to a specific output port through the I/O manager. The I/O manager assigns a sequence number for the output that is sent to all connected cells. The I/O manager keeps track output sequence number sent during a transaction, then if the transaction fails the I/O managers sends a special revert request to the output ports specifying the output sequence number before the transaction starts instructing the connected cells to revert back the state they had before the transaction were started. This preserves the consistency of the global system state when a specific transaction needs to be restarted due to implementation flaw. Restarting a transaction requires the loader to pick up a different implementation to be loaded to the sandbox. Unless all the implementations have flaws triggered by the same input, it is likely that the cell can gracefully recover from an encountered flaw by switching to different implementations. This failure recovery technique ensures that only the minimal amount of state reverting is done for each failure and avoids unnecessary full system restart

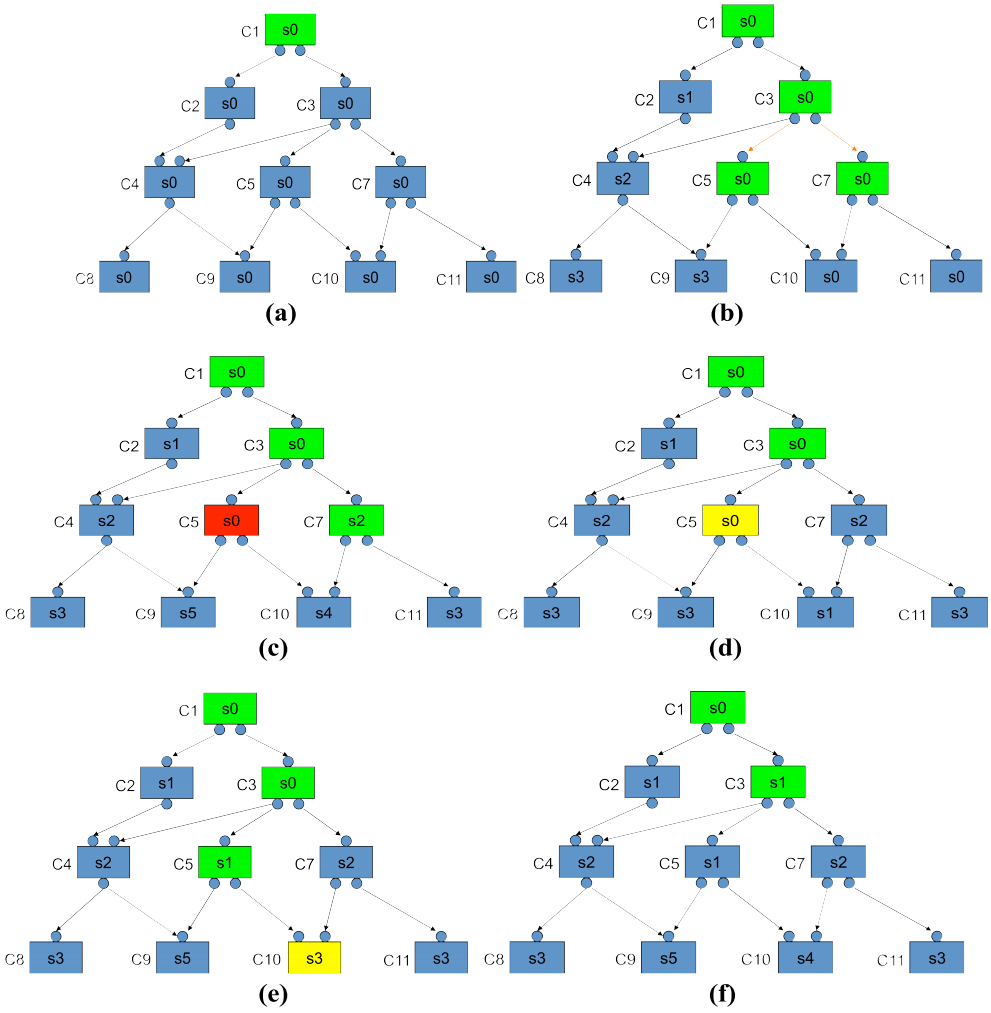

Figure 5.4: Automatic Recovery Example

in response to detecting flaws at runtime.

Example

This example aims to illustrate error recovery in BioSENSE. Consider an organism composed of 11 cells connected as shown in Figure 5.4. The cell C1 is the startup cell where the organism starts execution. At time t_0, all cells are in their initial state (s0) and the only active cell is C1. Each cell is dispatched when it receives input on one of its input ports. A state transition occurs only when the cell commits a state transaction which usually happens after the cells finishes processing its input. The dispatch of a cell may or may not lead to change in the cell's state and/or generate output that triggers the dispatch of other connected cells. We

assume that at time t_1, we have C1,C3,C5, and C7 dispatched as shown in figure 5.4(b). While cells C5 and C7 are executing they generated output that triggered the dispatch of cells C9, C10, and C11 a number of time, leading to multiple state transactions in those cells. At time t2, we reach the state shown in figure 5.4(c). At that time C9 had a state transition from s3 to s5 triggered by input from C5, and C10 had a state transition from s0 to s1 triggered by input from C7, followed by transition from s1 to s3 triggered by input from C5 and finally another transition from s3 to s4 triggered by input from C7. A failure is then detected in cell C5 as in figure 5.4(d). The recovery is performed as follows:

- The BS-DNA of C5 send a state revert request to C9 and C10 containing the sequence number of first output sent to each of the cells starting from the beginning of the transaction.
- C9 and C10 reverts their states from s5 and s4 to s3 and s1 respectively.
- C5 rolls back its state to remain at s0 and discard any uncommitted state changes
- C5 loads a different implementation and restarts the transaction using logged input.
- During the execution of the alternate implementation of C5, it generates output that triggers state transition in C9 and C10 to s5 and s3 respectively.
- C5 commits its state transaction and transition to s1, as shown in figure 5.4(e).
- Finally, C10 does forward recovery by reprocessing the input it received from C7 to transition from s3 to s4. The organism reaches the state shown in figure 5.4(f).

5.4 Discussion

The security and resilience benefits of MTD comes at additional cost ranging from minor to significant. Such cost is usually incurred in terms of performance overhead and increased development cost.

The cost associated with implementation diversity is mainly due to the increase in the development cost for implementing multiple versions of the same software system. Such cost maybe considered a major challenge facing the adoption of implementation diversity by software industry. Recent research attempted to address this challenge using synthetic code diversity where variants of protocol implementations are automatically generated from machine checked proof. The adoption of contract-based cells in BioSENSE, automates the verification and testing of components. Such techniques can be adapted to automatically synthesize implementation variants from formal specifications, to cost effectively exploit code diversity in BioSENSE.

Diverse building attempts to provide cost effective way to introduce diversity even when only a single implementation is present. The performance overhead for using diverse building of application is paid at the system building time, and should have negligible effect on the software system performance at runtime, unless runtime shuffling is employed. Performance sensitive systems that may not afford the overhead of runtime shuffling may still benefit from diverse building to produce binary-images that vary from one deployment of the system to another.

Runtime shuffling is responsible for the majority of the performance overhead associated with MTD. On each shuffling event, the runtime platform incurs significant cost of replacing the on-duty binary image variant with a different one. This results in invalidating any instruction cache. There is a trade-off between the non-determinism and performance that system designers should study to decide various shuffling parameters like shuffling algorithm, and rate. The more shuffling events, the more the performance overhead. For this reason, smart shuffling algorithms that attempt to minimize the number of shuffling events are needed.

5.5 Conclusion

In this chapter we presented the MTD approach adopted by BioSENSE to improve security and resilience of NSE software. We discussed the principles necessary to realize MTD. We introduced the concept of *behavior non-determinism* and discussed the different types non-determinism a software system can posses. We highlighted the role of diversity in achieving MTD. We classified different types of diversity including implementation diversity, build diversity, and runtime diversity. We showed how a loosely coupled architecture with dynamic contract-based bindings in addition to runtime shuffling can enable automatic fault recovery that significantly improves the robustness of software systems.

Chapter 6

BioSENSE Evaluation

"Theory guides; Experiment decides"
.. Izaak M. Kolthoff

6.1 Overview

The objective of this evaluation is to asses the effectiveness, efficiency, and performance characteristics of the BioSENSE platform under various networks conditions and scales. Our evaluation was conducted through simulations and analytical modeling. In this chapter, we start by introducing BS-SIM, an integrated simulation environment for BioSENSE, then present and discuss the evaluation results obtained from simulation and mathematical analysis.

6.2 Platform Performance Consideration

The elasticity, adaptability, and reconfigurability features of BioSENSE are key drivers of performance improvement in NSE software. Services that receive high work load may benefit from expanding its resource pool, while for services that receives low workload, the cost of maintaining a large resource pool may exceed possible savings. It also intuitive to expect that when services are processed as close to the users as possible, more savings can be gained. BioSENSE is inherently capable of providing this type of performance advantage.

Since the workload of the application and the locations of its users are dynamic in nature and may not be easily predicted, we see that the application reconfiguration due to change in the application workload has significant potential for energy savings. The elasticity provided

by BioSENSE should exploit such potential along with the other benefits described earlier. However, the type of performance advantage is hard to quantify since it is highly dependant on user and application behavior as well as the amount and nature of the change occurring.

Since BioSENSE exploits AMCR routing as a backbone for all its distributed operations, like resource discovery, resource allocation/management, data storage and retrieval, code distribution, and mission dissemination, the performance characteristics of AMCR serve as a reasonable estimator of BioSENSE performance characteristics.

6.3 BS-SIM

Simulation has been applied successfully for modeling large complex systems and understanding their behavior, especially in the networking and distributed systems areas. BS-SIM was designed and implemented with the purpose of realizing a scalable simulation environment for BioSENSE focusing on network and application layers. Modelling and simulation of large-scale, high resolution network models is very costly in terms of simulation time and hardware requirements. Also, the processing of large volume of detailed simulation data further increases the complexity of analysis. Consequently, it is desirable for the simulation of large scale networks to abstract some of the models to improve performance of the simulations by trading-off model details and fidelity. On the other hand, high-fidelity, high-resolution simulations may still be desirable for smaller scale networks. Accordingly, to satisfy different simulation needs, BS-SIM is designed to operate in two different modes: the *standalone* and *agent* modes. For maximum portability, BS-SIM is implemented in Java programming language. Figure 6.1 shows the classes and packages in BS-SIM.

6.3.1 Simulation Framework

BS-SIM simulation framework is composed of the following Java packages:

Engine

BS-SIM engine provides a parallel, thread-based, simulation runtime environment. The engine manages a pool of worker threads that provides execution context for simulation processes. Such process-based simulation environment allows operational implementation to be used in simulation environment with minimal modifications.

The engine package also provides fundamental modeling/simulation classes like, *Queue*, *Event*, and *SimulationClock*. The event manager uses the Java built-in pseudorandom number generator and adapts the uniformly distributed random sequence to the desired probability distribution when generating simulation events.

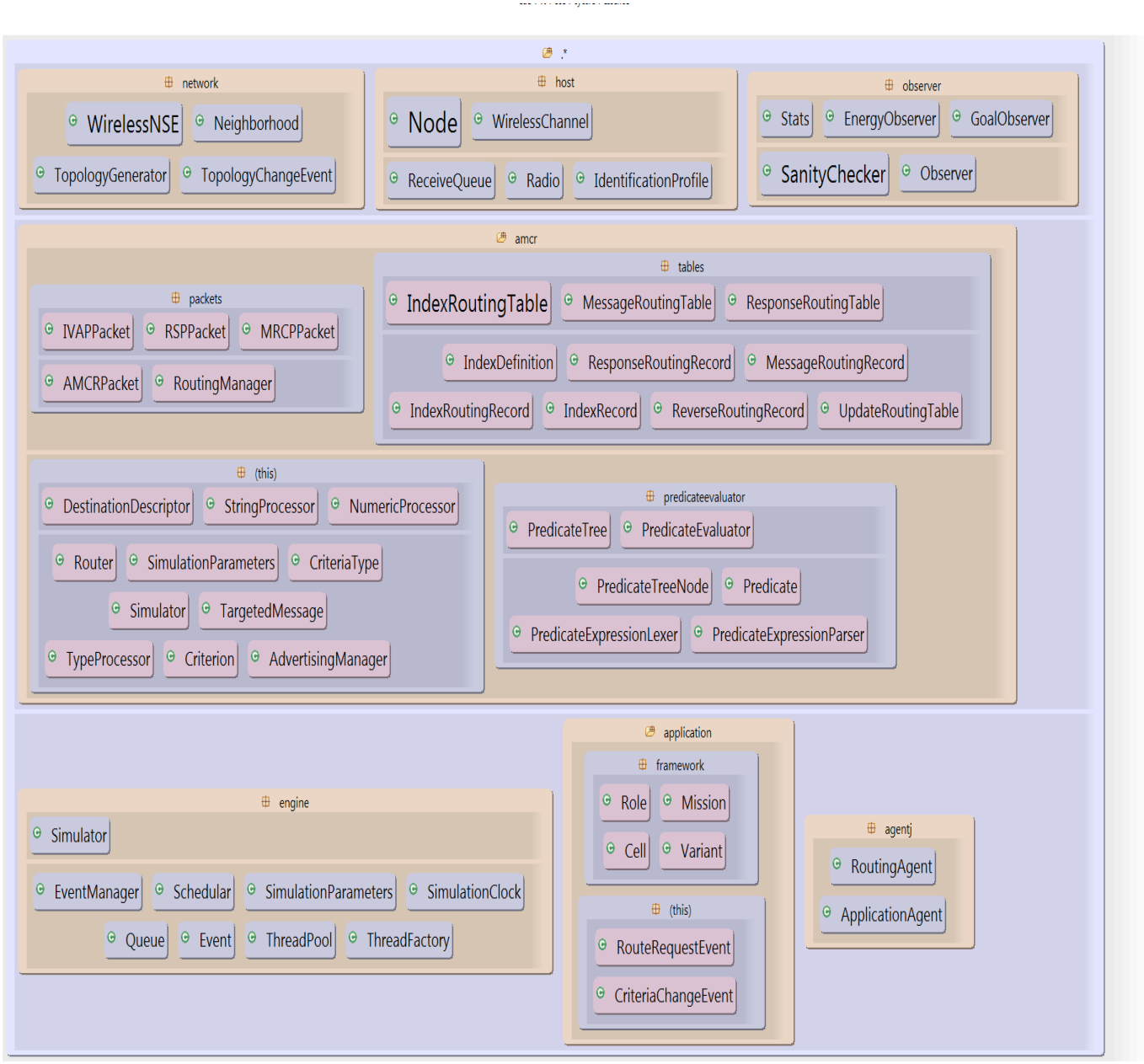

Figure 6.1: BS-SIM Simulation Framework

Observer

The observer package contains general operations classes that have a global view of the simulation process. Classes that collect runtime statistics and evaluate simulations goals belong to this package. Goal evaluation classes uses global knowledge of simulation events and network state to determine the optimal of goals and compares that to output of the simulation model.

Host

The host package contains the classes that model physical network nodes and physical layer of communication. Each host has its own independent execution context provided by the engine's thread pool.

Network

The network package contains the classes that generate topologies at simulation bootstrap time. During runtime, the topology change events are generated to reflect real life changes due to death of nodes, adding replacement nodes, and node mobility.

Application

The application package provides classes to model BioSENSE applications.

AMCR

The AMCR package provides an implementation of the routing protocol. It is contains of three sub-packages:

tables Manages the routing tables used by AMCR.

packets Defines the packet types used by AMCR.

predicateevaluator Evaluates descriptive destination descriptor predicates.

AgentJ

Contains the classes that bridge BS-SIM to AgentJ simulation environment. These classes are only used in agent mode on top of NS2 simulator [53]. The application agent class RoutingAgent

6.3.2 Standalone Mode

In standalone mode, BS-SIM works as a high performance, low-resolution, standalone process-based simulator. This mode is more suitable for simulating large scale networks. In this mode, abstract models for the physical and MAC layers are used to keep the execution cost manageable. Also, a light-weight, multi-threaded simulation engine is employed.

Advantages

The standalone mode is considerably faster and more scalable than the agent mode due to light weight design of the simulation engine and the low-fidelity modelling of the network physical properties. It is able to run simulation scenarios of 10th of thousands of nodes on a single simulation server.

Disadvantages

The use of abstract models in the physical and MAC layers decreases the fidelity of the simulation. For example, in standalone mode, the simulation results are not sensitive to different types or properties of antennas, signal propagation, or collision management. The study of the impact of such parameters is only possible in agent mode.

6.3.3 Agent Mode

In agent mode, BS-SIM is used together with NS2 and AgentJ to realize an integrated high-resolution simulation environment. BS-SIM realizes an NS2 routing agent as well as an application agent. NS2 provides high-fidelity models for the MAC and Physical layers complementing the BS-SIM models for application and network layers. AgentJ [30] is needed to allow the BS-SIM models developed in Java to integrate with NS2 models developed in C++.

Advantages

Allows high-fidelity, high-resolution simulation.

Disadvantages

Time and resource consuming. Can not scale to large number of nodes.

6.4 Routing Evaluation

In this section we show how AMCR indexes considerably improve routing efficiency, compared to purely reactive protocols, by speeding up route discovery to frequently targeted groups at the cost of pre-advertising indexed criteria values. The amount of performance improvement is dependent on the index efficiency and amount of messages that benefit from the index. The results shows that even for unbiased random requests, indexing still considerably improves the and overall routing efficiency. In this section, we evaluate the effectiveness and efficiency of AMCR using simulation and present derive an analytical formula for AMCR overhead when sparse routing overlay is used.

6.4.1 Simulation Design

We use simulation to study the performance characteristic of routing in BioSENSE. Evaluation experiments were conducted using an AMCR implementation within BS-SIM simulated environment.

6.4.2 Network Model

We model the AMCR network as a graph $G(n, m)$. The parameters n refers to the number of elements in G, and m refers to the number of edges in the graph. BS-SIM generates random graph topologies having the following constraints.

- Nodes are uniformly distributed in a square area of side width w
- The average density of nodes, d, equals $\frac{n}{a^2}$
- The network graph is unpartitioned
- Each node has e criteria in its identification profile. The values of the criteria are randomly generated within the range [0..v].
- Nodes can communicate in single hop with other nodes in the separating distance is less than or equals to s meters.

6.4.3 Traffic Model

For each request, the source node is randomly selected among all nodes, and a destination descriptor is generated with random criteria values. Nodes are considered matching the destination descriptor if their criteria values are within a window of width w of the randomly

Table 6.1: Simulation Parameters

Parameter	Symbol	Default Value	Unit
Number of nodes	n	200	node
Density of nodes	d	0.0008	$node/m^2$
Route Request Rate	q	0.05	request/sec
Criteria Update Rate	c	0.05	update/sec
Transmission Range	s	50	m
Advertisement Period	h	100	sec
Experiment Length	t	6,000	sec
Advertisement Packet Size	l	50	record/packet
Total Number of Criteria	e	4	criterion
Number of Targeted Criteria	f	2	criterion
Number of Indexed Criteria	r	1-4	criterion
‘ Average Size of Destination Group	g	5	node
Criteria Value Range	v	1000	-
Destination Descriptor Window Size	w	$v(\frac{g}{n})^{\frac{1}{f}}$	-
Router Packing Ratio	p	1	-
Number of router nodes	u	$\frac{n}{p}$	node
Deployment Area Side length	a	$\sqrt{\frac{n}{d}}$	m

selected descriptor value. The window width, w, is calculated by the simulator such that the average number of nodes matching the descriptor remains equals to g. $w = v[\frac{g}{n}]^{\frac{1}{f}}$. The inter-arrival time of route requests is exponentially distributed with rate parameter q. Criteria change events randomly occur during the experiment. The inter-arrival time of change events is exponentially distributed with rate parameter c. Figure 6.1 shows the parameters used in our simulation study.

6.4.4 Experiment Design

In our simulations, we used BS-SIM, in standalone mode, over Java 1.6 virtual machine running on top of 64 bit Linux 2.6.18 kernel. The simulation hardware has 16 CPU cores (quad CPU, quad core) operating at 2GHz clock speed and 16GB of RAM. We conduct four sets of experiments to evaluate effectiveness, efficiency, scalability, and change overhead. Each set of experiments is composed of a number of experiments, each with a unique set of simulation parameters. In our evaluation, we are interested in studying the impact of criteria indexing on effectiveness and efficiency of AMCR, so we repeat each experiment for five different criteria indexing degrees ranging from "no indexing" to full indexing, resulting in five unique simulation scenarios for each experiment. We repeat each simulation scenario ten times and record the mean value, standard deviation, and confidence interval at 95% calculated using normal distribution. Whenever the calculated value of confidence

interval exceeds 5% of the mean value, additional runs are performed to till the confidence interval reaches acceptable range.

6.4.5 Results

We evaluate the effectiveness and efficiency of AMCR under different criteria update rates and number of indexes criteria. The main purpose of AMCR is to construct a routing path, on-demand, between a sender and a destination group of nodes that match specific set of criteria. The evaluation of AMCR effectiveness is required to verify that it actually provides such functionality. We calculate the perfect destination groups using an oracle process that has global knowledge of all generated destination descriptors and node profiles. We then compare the perfect group to the real group and count false negatives. The efficiency of AMCR is evaluated in terms of the energy consumed to construct the routes. We use the total number of packet transmissions as an estimator for the the energy consumed assuming that reception energy is negligible compared to transmission energy. The total number of transmissions includes all advertisement overhead as well as the route construction control packets. For each experiment, we compare 5 data sets for different levels of indexing ranging from "no indexing" to "full indexing".

Experiment Set 1: Effectiveness

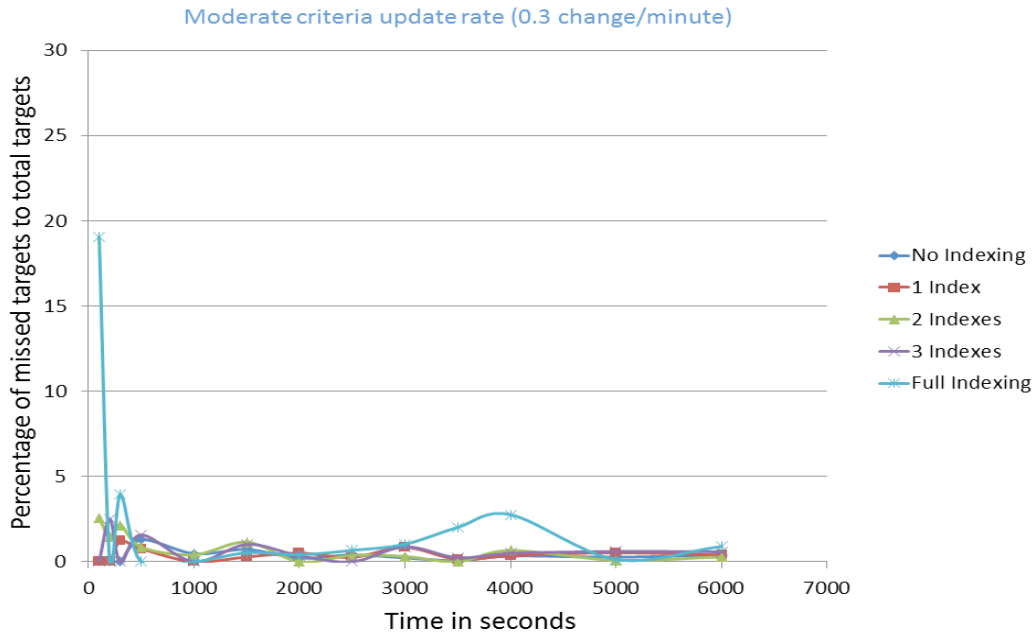

Figure 6.2: Percentage of False Negatives Over Time (Moderate Criteria Update Rate)

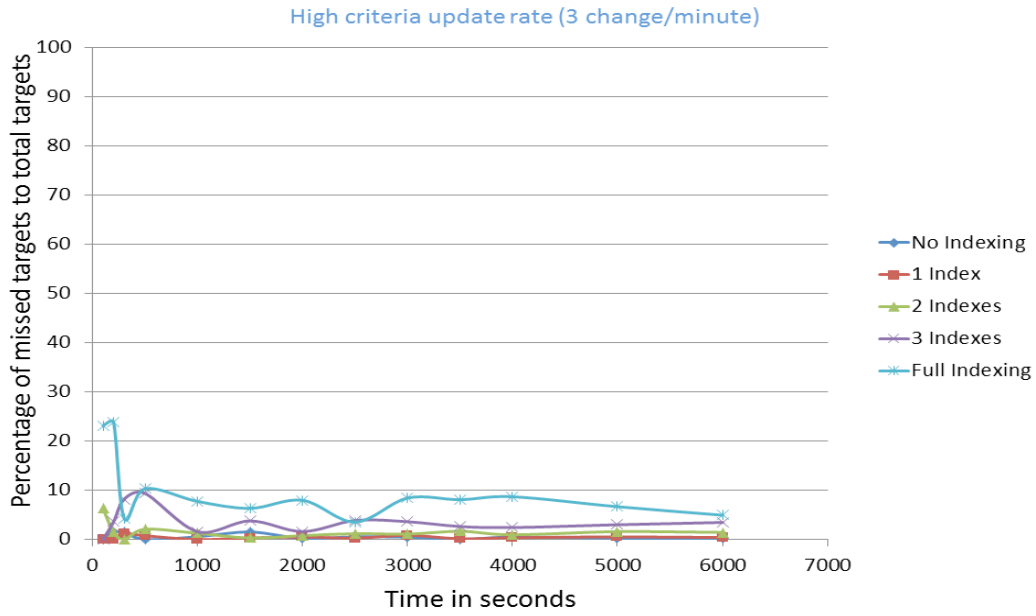

Figure 6.3: Percentage of False Negatives Over Time (High Criteria Update Rate)

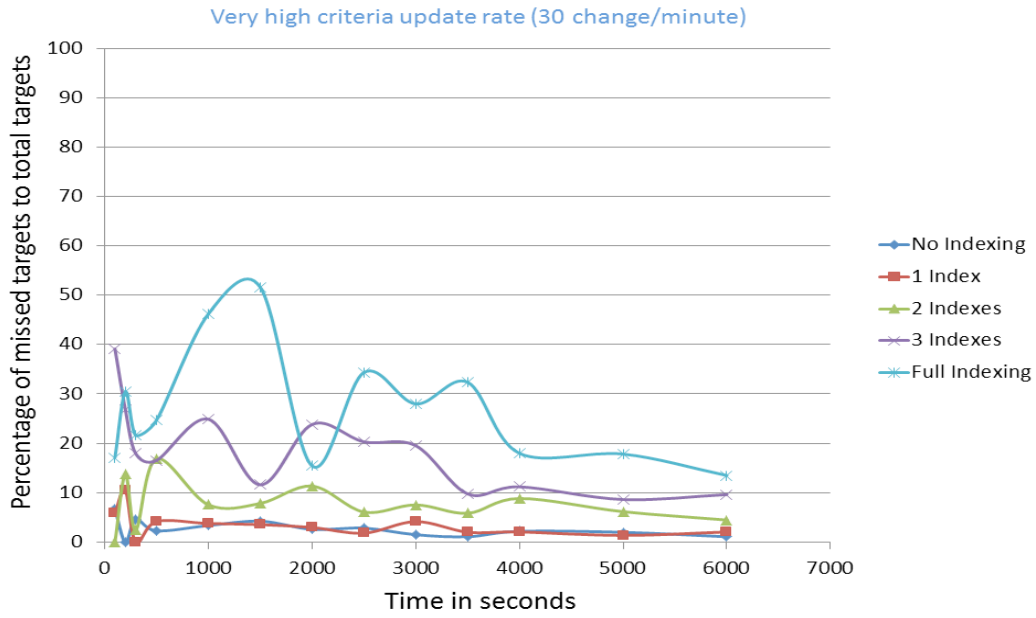

Figure 6.4: Percentage of False Negatives Over Time (Very High Criteria Update Rate)

The first set of experiments measure the effectiveness of AMCR through the number of false negatives under moderate, high, and very high criteria update rates. The results of this set of experiments are plotted in figures 6.2, 6.3, and 6.4. False negatives occur primarily when a node updates the value of an indexed criteria at a point of time, but advertising the new value is not yet complete. Accordingly, adding more indexes is expected to increase the number of false negatives. After a brief setup period, the false negatives start to converge for moderate and high criteria update rates. As the change rate increases, the difference between the data sets becomes more significant.The very high criteria update rates, random full results in unacceptable effectiveness. For criteria with very high update rates, random indexing should be avoided.

Experiment Set 2: Efficiency

The second set of experiments measure the energy consumed by recording the number of packet transmissions per unit time. The results of this set of experiments are plotted in figures 6.5, 6.6, and 6.7. The results show that after a brief setup period, full indexing managed to save 58% of the energy consumption. At moderate criteria update rate, all levels of indexing provided savings. As the criteria update rate increases the energy cost of indexing increases.

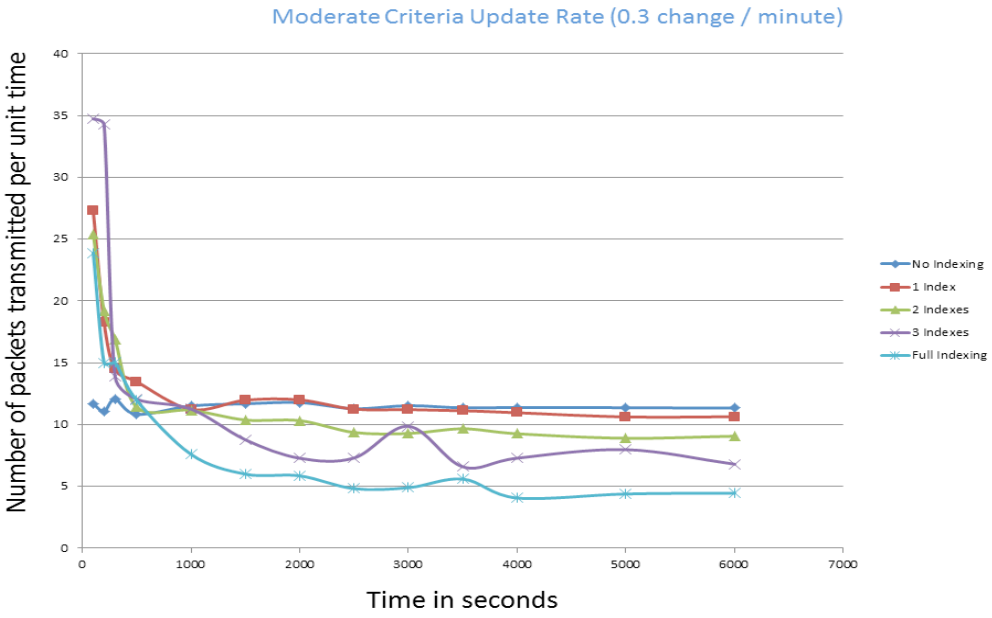

Figure 6.5: Energy Consumption Over Time (Moderate Criteria Update Rate

Experiment Set 3: Scalability

The third set of experiments evaluates the network and workload scalability of AMCR. Figures 6.8, and 6.9 show the impact of the network size and workload on the energy consumption. As the

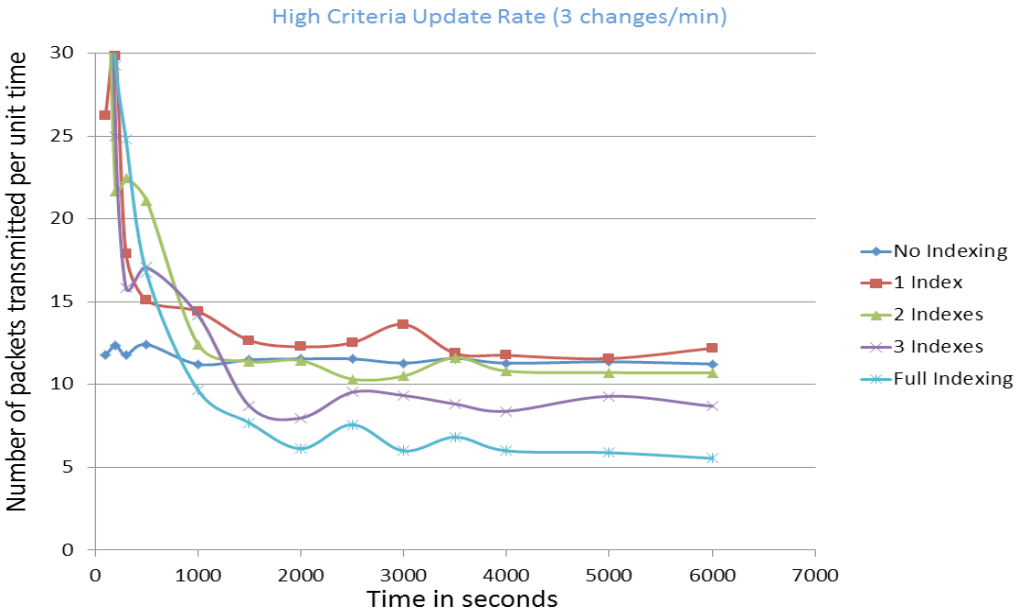

Figure 6.6: Energy Consumption Over Time (High Criteria Update Rate

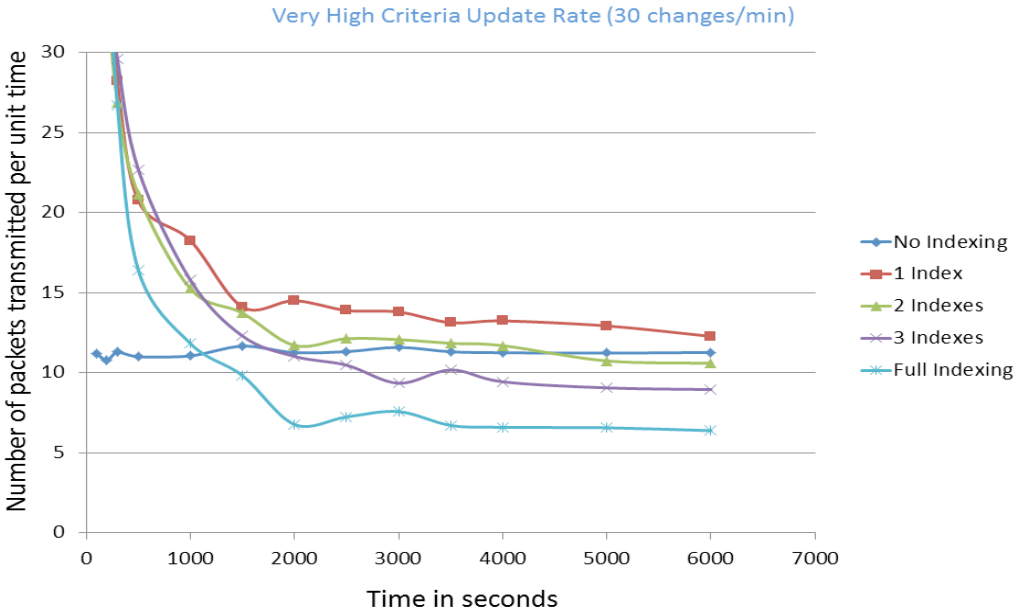

Figure 6.7: Energy Consumption Over Time (Very High Criteria Update Rate

number of nodes increase the energy consumed is expected to increase. As the level of indexing increases, the increase in the energy consumed becomes slower. The results show that as the network size or workload increases, the energy saving from indexing becomes more significant. At very low workload indexing can result in increased energy consumption.

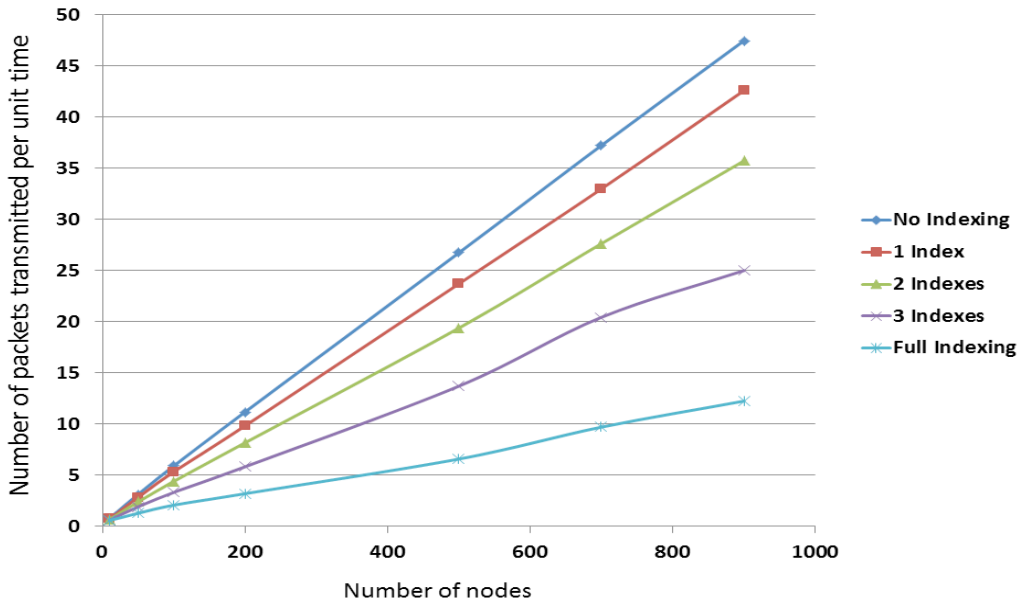

Figure 6.8: Energy Consumption Against Number of Nodes

Experiment Set 4: Update Overhead

We finally show the impact of the criteria update rate and the average size of destination groups on the energy consumed in figures 6.106.11. As criteria update rate increases, the indexing overhead increases till it reaches a point where any more increase in the update rate will result in indexing of more overhead than savings.

6.4.6 Routing Overlay

In the previous section we presented a simulation study for AMCR effectiveness and efficiency in a flat network, where every node in the network participate in routing. In this section, we analytically study the impact of limiting AMCR functionality to a routing overlay that.

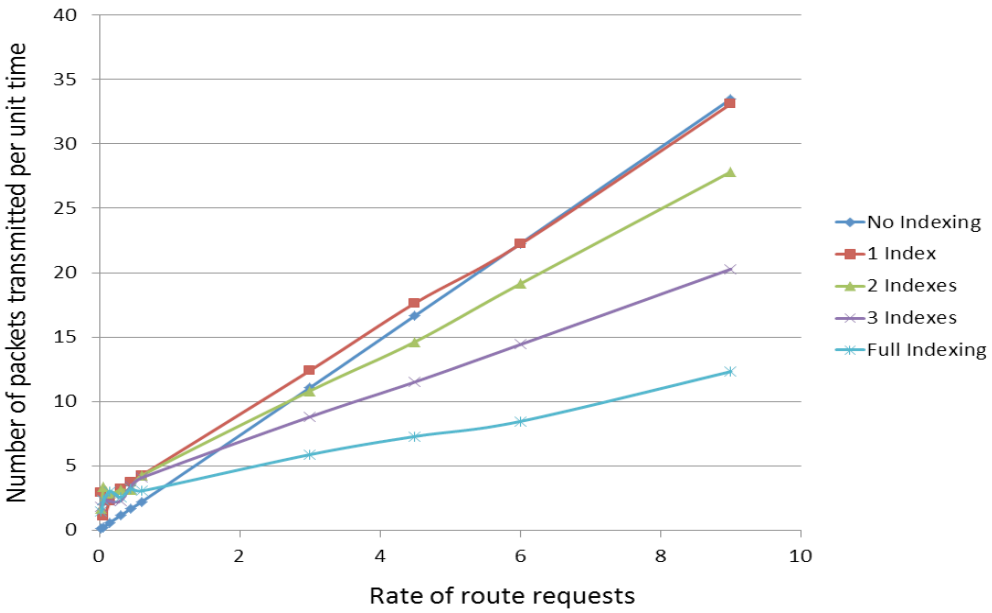

Figure 6.9: Energy Consumption Against Route Request Rate

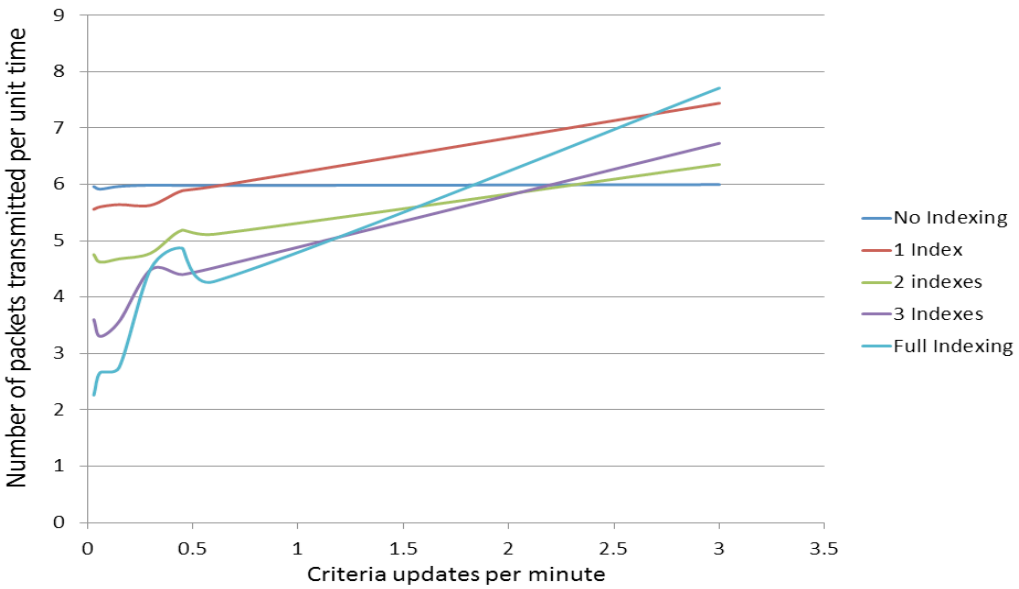

Figure 6.10: Energy Consumption Against Criteria Update Rate

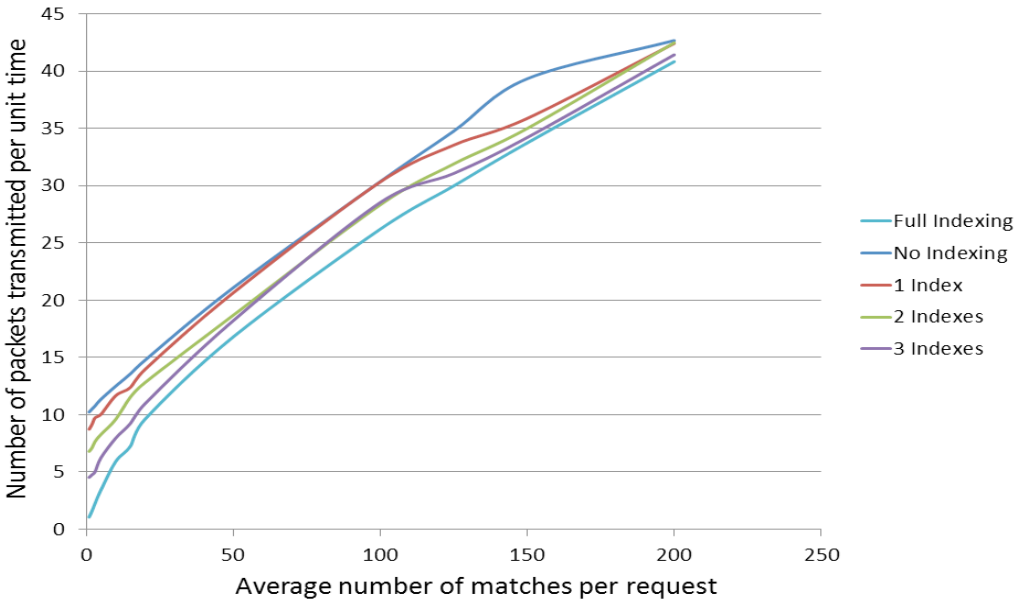

Figure 6.11: Energy Consumption Against Destination Group Size

6.4.7 Network Model

We model the AMCR network as a graph $G(G_r(u, m_r), p)$, where $G_r(u, m_r)$ refers to *routing overlay graph*, which is a random graph of *router nodes*. The set of vertices of the routing graph, R, contains all *router nodes*. The set of vertices of the *network graph*, R, is a superset of N containing also vertices for *resource nodes*. The parameters u refers to the number of elements in R, m_r refers to the number of edges in the graph, and p refers to the router *packing ratio*, which represents that average number of resource nodes per each router node. The total number of nodes in the network, $n = u(p+1)$. The degree of the *routing graph*, d_r, represents the average number of router neighbors. For the *routing graph* to be connected with high probability, the average degree of nodes grows with $O(\log(u))$ [43], hence we assume that the $d_r = k \log u$, where k is a constant.

We define the index propagation overhead O as the expected value for the number of IVAP packets forwarded by a node in unit time. We show that $O = (\frac{1}{h} - \frac{1}{h(p+1)})\lceil \frac{r - re^{-hc}}{l} \rceil + \frac{k}{h(p+1)} \log \frac{n}{p+1}$, where n is the number of nodes in the network, r is the number of indexed criteria, c is the rate parameter of exponential distribution generating the index update events, and l is the maximum number of index update records that fit in a single advertisement packet, h is the period of the index advertising cycle, and p is the *router packing ratio*.

We assume that updates to indexed criteria are random events that occur independently with a expected value for the time units between updates to the criterion $\frac{1}{c}$. For a given indexed criterion, the probability that an update will occur during the period of an advertisement cycle $= \int_0^p ce^{-tc}\, dt = 1 - e^{-hc}$. The average number of inde1xed criteria updates in a node per advertising

Table 6.2: Analytical Study Parameters

Parameter	Symbol	Default Value	Unit
Number of nodes	n	200	node
Number of Indexed Criteria	r	1-4	criterion
Packing Ratio	p	10	
Advertisement Packet Size	l	50	record/packet
Advertisement Period	h	100	sec
Criteria Update Rate	c	0.05	update/sec

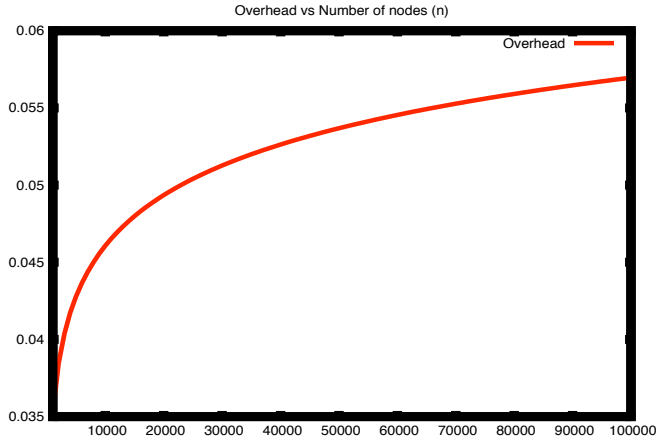

Figure 6.12: Overhead vs Number of nodes

cycle $= i = r - re^{hc}$. The average number of packets generated by a single resource node per advertising cycle is $\lceil \frac{i}{l} \rceil = \lceil \frac{r-re^{-hc}}{l} \rceil$. The average number of aggregated packets generated by a router node per advertising cycle $= d_r = k \log u$. Total number of packets transmitted in the network per advertising cycle $= (n-u)\lceil \frac{r-re^{-hc}}{l} \rceil + ku \log u$ Therefore, $O = \frac{f(n-u)}{n} \lceil \frac{r-re^{hc}}{l} \rceil + \frac{kfu}{n} \log u = (\frac{1}{h} - \frac{1}{h(p+1)}) \lceil \frac{r-re^{-hc}}{l} \rceil + \frac{k}{h(p+1)} \log \frac{n}{p+1}$

Table 6.2 shows the parameters used in our study and their default values in the simulated experiments.

6.4.8 Result

We used the analytical overhead equation presented above to draw the relation between various parameter and index propagation overhead when routing is performed by a sparse overlay. Figure 6.13 shows that the overhead quickly drops to as routing *packing ratio* increases to practically reasonable values. The router packing ratio is the average number of resource nodes per gateway (router node). The scalability of index propagation is clearly shown in figure 6.12 where overhead

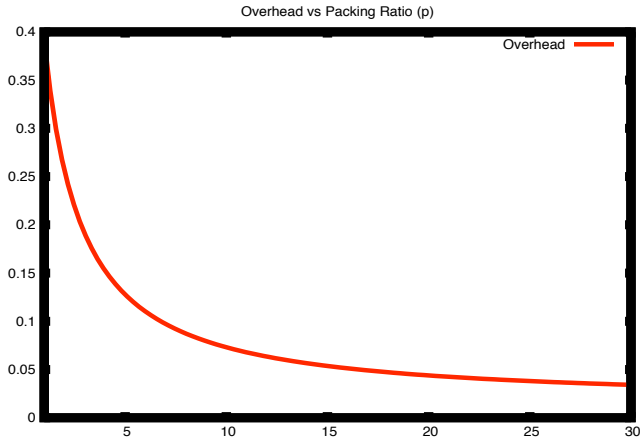

Figure 6.13: Overhead vs Packing ratio

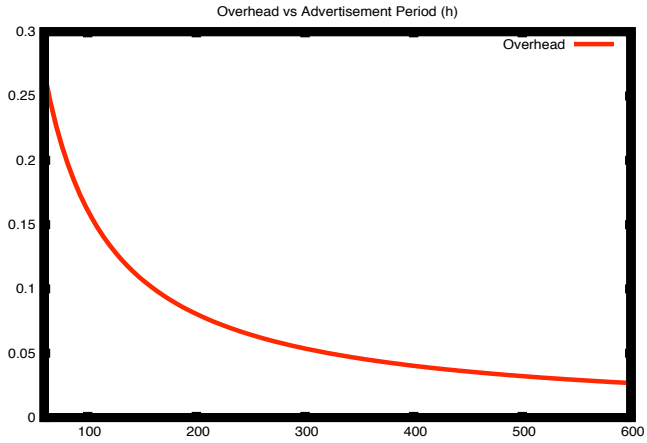

Figure 6.14: Overhead vs Advertisement Period

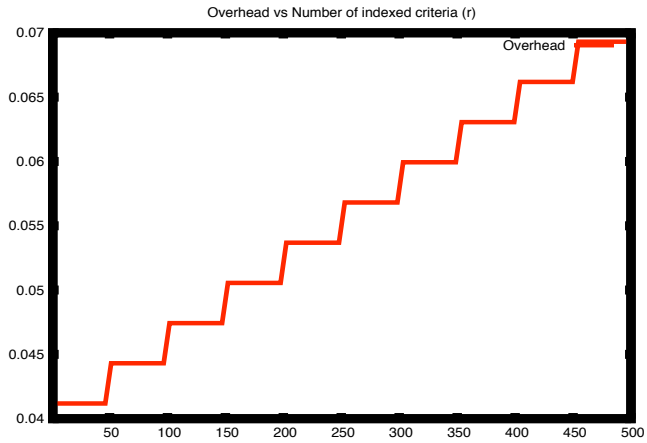

Figure 6.15: Overhead vs No. of indexed criteria

is only slightly increased from 1000 nodes to 100,000 nodes at packing ratio of 10. Figure 6.14 shows that overhead quickly drops as the advertisement period is increased beyond few minutes. Applications can trade-off index propagation efficiency for faster response to change in indexed criteria, however, for most typical NSE applications a delay of few minutes or more to propagate criteria changes is acceptable. The index propagation overhead also increases in steps with the increase in the number of indexed criteria as shown in figure 6.15. The step effect results from IVAP packetization. The above results show that routing overhead can be greatly reduced when only a subset of nodes participate in routing, given that all nodes are within a single hop communication distance from at least one router.

Chapter 7

Conclusion and Future Work

"I came to the conclusion that I am not a fiction writer" .. Tim LaHaye

7.1 Contributions

In this dissertation we presented, BioSENSE, a secure elastic application platform for large-scale shared/federated networked sensor environments. BioSENSE is designed to adapt its structure and behavior according to scale, mission, and context. The following contributions to the field of Computer Engineering were made as part of this dissertation:

- Biologically-inspired construction of elastic, adaptable, resilient NSEs transcending node lifetime.
- Application-aware generic communications framework with extensible criteria-based addressing and associative routing.
- Management of component-based spatio-temporal diversity subject to diversity-trust-performance tradeoff.

BioSENSE enables on-demand, online, adaptable allocation of federated resources shared among multiple long-lived applications with dynamic missions and quality of service, realizing the "platform-as-a-service" model for provisioning NSE applications, and promising to lower the market entry barriers that impedes the adoption of NSE at a large scale. BioSENSE supports on-line, over-the-air code distribution allowing generic nodes to go from factory to field directly, then acquire their configuration and application code from within the network.

BioSENSE is inspired by biological systems that exhibit non-deterministic behavior, adaptable role-playing, and elastic structural organization. These properties of biological systems are needed

to solve the security, resilience, and scalability challenges facing large scale, publicly accessible networked sensor environment. BioSENSE systemically impede behavior discovery of NSE software implantations by inducing non-determinism in software behavior at runtime, resulting in attacker confusion and minimizes the impact of compromising individual nodes through system managed spatio-temporal software diversity. Consequently, BioSENSE reduces the attack surface and improve the security and resilience of the system. It also provides a systemic mechanism to exploit software redundancy to automatically recover from failures resulting from implementation flaws.

We proposed a novel associative routing that exploits generic node identification profiles and destination descriptors to enable both resource/service and path discovery in a single phase. The advantages of associative routing include: 1) dynamic identification and addressing, 2) ability to exploit application layer semantics to optimize routing, 3) intrinsic multicast, and 4) preserving node anonymity. We proposed a framework for associative routing protocols and presented AMCR that was designed and implemented using the proposed framework. AMCR is a generic routing protocol for NSE that exploits associative routing. AMCR supports multi-application optimization while being purely generic, promising the construction of future NSE that are of large scale, long lived, and multi-purpose. Analytical evaluation shows that AMCR is highly scalable and has minimal advertisement overhead.

Figure 7.1 summarizes the contributions, features, and advantages of BioSENSE.

	Efficiency	Adaptability	Resilience	Security
Bio-Inspired Secure Elastic System Design	▪ Resource Sharing ▪ Specialization of resources	▪ Elasticity ▪ Role-oriented architecture ▪ Transparent run-time code distribution ▪ Re-specialization of resources ▪ Cooperative diversity	▪ Elasticity ▪ State-Replication ▪ Automatic Fault recovery ▪ System-managed state transitions ▪ Trust-based Shuffling	▪ Sandbox Execution ▪ Cooperative diversity
Application-Aware Generic Associative Communication Plane	▪ Merging resource/service discovery and route discovery ▪ Using criteria indexes to avoid uncontrolled flooding	▪ Criteria indexing according to observed workload	▪ Path redundancy ▪ Resource redundancy	▪ Node anonymity

Figure 7.1: BioSENSE Features vs Research Objectives

7.2 Future Work

Future work will focus on the following directions:

1. Pervasive monitoring and analytics : Study methods and techniques to enable pervasive monitoring and analytics as an intrinsic BioSENSE feature, providing a systematic way to collect and analyze information for NSE applications to make informed decisions.

2. Quality of Service (QoS) : Devise models for the instrumentation and control of various QoS parameters in BioSENSE, and develop corresponding control mechanisms based on dynamic contract negotiation and allows various types of resource reservation.

3. Collaborative diversity: BioSENSE provides the necessary system-level mechanisms to exploit software diversity using variant shuffling, however, algorithms and protocols for coordinated spatio-temporal diversity are needed to maximize the application resilience against attacks. While temporal diversity is fully achieved using variant shuffling where nodes independently vary their active software implementation over time independent from any other node, the spatial diversity requires communication among nodes to co-ordinate their implementation shuffling such that nodes close to each others tend to have different implementation at the same time instance even if they play the same role in the network. Effective control algorithms, models and metrics for quality-of-diversity should be developed.

4. Reliable transport for associative routing: While the AMCR protocol presented in this dissertation provides an adaptive network layer for NSE that can exploit both path redundancy and end-point redundancy, an adaptive transport layer is still needed on-top of associative routing to provide applications reliability and quality of service control. Different transport layer protocols could be designed for different communication schemes that can share the same pool of NSE resources due to the clear network layer boundary defined by associative routing. The interaction between different transport layer protocols is a subject of future studies.

5. Test bed: Complete the implementation of BioSENSE test bed and conduct rigorous evaluation of different classes of real life applications under various scales and workload patterns. BioSENSE test bed integrates the Java implementation of BioSENSE, NS2 network simulator, and AgentJ Java-to-NS2 bridge. Future versions of the test bed will exploit the JavaRMI technology to allow parallel simulation of processing intensive scenarios. The test bed will allow us to explore massive-scale hybrid environments with mobile and stationary nodes.

6. Smart Infrastructures: Although BioSENSE is design for infrastructure-less ad-hoc networks, it is potentially a flexible and manageable platform for various smart infrastructures such as transportation, power-grid (smart grid), emergency response, etc. The applicability ans feasibility of adopting AMCR for such smart environment should be studied and compared to alternative approaches.

7. IP compatibility layer for AMCR: Although the AMCR protocol, proposed in this dissertation, provides greater flexibility in identification, addressing, and communication patterns than the IP protocol, the implementation of IP compatibility layer is useful allow objective

comparative study between the associative paradigm and the identifier-based paradigm. The co-existence of associative routing service and IP routing on NSE allows easy and cheap path to adopt associative networking incrementally in a hybrid environment.

8. Electronic application marketplace and application configuration management: Design and implement component-based electronic marketplace for BioSENSE applications allowing application evolution through managed updates of cell variants assisted by a configuration/version control system. A component versioning scheme should be developed as well as models for safe updates. An update manager should be designed to execute these models to keep applications up-to-date.

Bibliography

[1] Web Services Architecture. Technical report, World Wide Web Consortium, February 2004.

[2] T. Abdelzaher, B. Blum, Q. Cao, Y. Chen, D. Evans, J. George, S. George, L. Gu, T. He, S. Krishnamurthy, L. Luo, S. Son, J. Stankovic, R. Stoleru, and A. Wood. Envirotrack: towards an environmental computing paradigm for distributed sensor networks. In *Distributed Computing Systems, 2004. Proceedings. 24th International Conference on*, pages 582 – 589, 2004.

[3] Paul E. Ammann and John C. Knight. Data diversity: An approach to software fault tolerance. *IEEE Trans. Comput.*, 37:418–425, April 1988.

[4] Isaac Amundson, Manish Kushwaha, Xenofon Koutsoukos, Sandeep Neema, and Janos Sztipanovits. "oasis: A serviceoriented middleware for pervasive ambientaware sensor networks". Technical report, Institute for Software Integrated Systems, Vanderbilt University, 2006.

[5] A. Avizienis. The n-version approach to fault-tolerant software. *Software Engineering, IEEE Transactions on*, SE-11(12):1491 – 1501, dec 1985.

[6] Dharini Balasubramaniam, Alan Dearle, and Ron Morrison. A composition-based approach to the construction and dynamic reconfiguration of wireless sensor network applications. In Cesare Pautasso and Éric Tanter, editors, *Software Composition*, volume 4954 of *Lecture Notes in Computer Science*, pages 206–214. Springer, 2008.

[7] David Barkai. *Peer-to-Peer Computing: Technologies for Sharing and Collaborating on the Net*. Intel Press, 2001.

[8] Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. *Swarm intelligence: from natural to artificial systems*. Oxford University Press, Inc., New York, NY, USA, 1999.

[9] Peter Braun and Wilhelm Rossak. *Mobile Agents: Basic Concepts, Mobility Models, and the Tracy Toolkit*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[10] Timothy A. Budd. *An introduction to object-oriented programming*. Addison-Wesley, 1991.

[11] Bush, Stephen, Kalyanaraman, and Shivkumar. Management of active and programmable networks. *Journal of Network and Systems Management*, 14(1):1–5, March 2006.

[12] Stephen F. Bush and Amit B. Kulkarni. *Active Networks and Active Network Management: A Proactive Management Framework*. Kluwer Academic/Plenum Publishers, 2001.

[13] M. Caleffi and L. Paura. M-dart: multi-path dynamic address routing. *Wireless Communications and Mobile Computing*, pages n/a–n/a, 2010.

[14] Andrew T. Campbell, Herman G. De Meer, Michael E. Kounavis, Kazuho Miki, John B. Vicente, and Daniel Villela. A survey of programmable networks. *SIGCOMM Comput. Commun. Rev.*, 29:7–23, April 1999.

[15] Andrew T. Campbell, Herman G. De Meer, Michael E. Kounavis, Kazuho Miki, John B. Vicente, and Daniel Villela. A survey of programmable networks. *SIGCOMM Comput. Commun. Rev.*, 29(2):7–23, 1999.

[16] Miguel Castro, Peter Druschel, Ayalvadi Ganesh, Antony Rowstron, and Dan S. Wallach. Secure routing for structured peer-to-peer overlay networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):299–314, 2002.

[17] Hojung Cha, Sukwon Choi, Inuk Jung, Hyoseung Kim, Hyojeong Shin, Jaehyun Yoo, and Chanmin Yoon. The retos operating system: kernel, tools and applications. In *Proceedings of the 6th international conference on Information processing in sensor networks*, IPSN '07, pages 559–560, New York, NY, USA, 2007. ACM.

[18] Xingchen Chu, Tom Kobialka, Bohdan Durnota, and Rajkumar Buyya. Open sensor web architecture: Core services. In *Proceedings of the 4th International Conference on Intel ligent Sensing and Information Processing (ICISIP 2006*. IEEE Press, Piscataway, New Jersey, USA, December 2006.

[19] Paul Clements, Rick Kazman, and Mark Klein. *Evaluating Software Architectures: Methods and Case Studies*. Addison-Wesley, 2001.

[20] Alan Colman and Jun Han. Using role-based coordination to achieve software adaptability. *Sci. Comput. Program.*, 64(2):223–245, 2007.

[21] Thierry Coupaye and Jean-Bernard Stefani. Fractal component-based software engineering. In *ECOOP Workshops*, pages 117–129, 2006.

[22] Alan Demers, Johannes Gehrke, Rajmohan Rajaraman, Niki Trigoni, and Yong Yao. The cougar project: a work-in-progress report. *SIGMOD Rec.*, 32(4):53–59, 2003.

[23] Amnon H. Eden. A theory of object-oriented design. *Information Systems Frontiers*, 4:379–391, December 2002.

[24] Mahmoud ElGammal and Mohamed Eltoweissy. Location-aware affinity propagation clustering in wireless sensor networks. *Wireless and Mobile Computing, Networking and Communication, IEEE International Conference on*, 0:471–475, 2009.

[25] J. Eriksson, M. Faloutsos, and S. V. Krishnamurthy. Dart: Dynamic address routing for scalable ad hoc and mesh networks. *Networking, IEEE/ACM Transactions on*, 15(1):119–132, 2007.

[26] David F. Ferraiolo, Ravi Sandhu, Serban Gavrila, D. Richard Kuhn, and Ramaswamy Chandramouli. Proposed nist standard for role-based access control. *ACM Trans. Inf. Syst. Secur.*, 4(3):224–274, 2001.

[27] Roy T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[28] Chien-Liang Fok, Gruia-Catalin Roman, and Chenyang Lu. Mobile agent middleware for sensor networks: an application case study. In *IPSN '05: Proceedings of the 4th international symposium on Information processing in sensor networks*, page 51, Piscataway, NJ, USA, 2005. IEEE Press.

[29] Heinzelman and H. Balakrishnan. Adaptive protocols for information dissemination in wireless sensor networks. In *Proceedings of the fifth annual ACM/IEEE international conference on Mobile computing and networking*, pages 174–185, Seattle, WA USA, 1999.

[30] U. Herberg and I. Taylor. Development framework for supporting java ns2 routing protocols. In *Future Information Technology (FutureTech), 2010 5th International Conference on*, pages 1 –5, May 2010.

[31] Christophe Huygens and Wouter Joosen. Federated and shared use of sensor networks through security middleware. *Information Technology: New Generations, Third International Conference on*, 0:1005–1011, 2009.

[32] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva. Directed diffusion for wireless sensor networking. *Networking, IEEE/ACM Transactions on*, 11(1):2 – 16, February 2003.

[33] Manabu Isomura, Christian Decker, and Michael Beigl. Generic communication structure to integrate widely distributed wireless sensor nodes by p2p technology.

[34] Inc. John Wiley & Sons, editor. *Handbook of wireless networks and mobile computing*. John Wiley & Sons, Inc., New York, NY, USA, 2002.

[35] Safoora Shakil Khan and Muhammad Jaffar-ur Rehman. A survey on early separation of concerns. In *Proceedings of the 12th Asia-Pacific Software Engineering Conference*, pages 776–782, Washington, DC, USA, 2005. IEEE Computer Society.

[36] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-oriented programming. In Mehmet Akşit and Satoshi Matsuoka, editors, *ECOOP'97 Object-Oriented Programming*, volume 1241 of *Lecture Notes in Computer Science*, chapter 10, pages 220–242. Springer-Verlag, Berlin/Heidelberg, 1997.

[37] Sangkyung Kim, Wonjong Noh, and Sunshin An. Multi-path ad hoc routing considering path redundancy. *Computers and Communications, IEEE Symposium on*, 0:45, 2003.

[38] J. King, R. Bose, S. Pickles, A. Hetal, S. Vanderploeg, and J. Russo. Atlas a service-oriented sensor platform. In *Proceedings of SenseApp*, 2006.

[39] Manish Kochhal, Loren Schwiebert, and Sandeep Gupta. Role-based hierarchical self organization for wireless sd-hoc sensor networks. In *WSNA'03*, pages 98–108. ACM, 2003.

[40] Joanna Kulik, Wendi R. Heinzelman, and Hari Balakrishnan. Negotiation-based protocols for disseminating information in wireless sensor networks. *Wireless Networks*, 8:169–185, 1999.

[41] Manish Kushwaha, Isaac Amundson, Xenofon Koutsoukos, Sandeep Neema, , and Janos Sztipanovits. "oasis: A programming framework for serviceoriented sensor networks". In *Proceedings of the Second International Conferences of Communication System Software and Middleware (Comsware 2007), Bengalore, India*, January 2007.

[42] Edward A. Lee. Cyber physical systems: Design challenges. Technical Report UCB/EECS-2008-8, EECS Department, University of California, Berkeley, Jan 2008.

[43] Katharina A. Lehmann1 and Michael Kaufmann1. *Peer-to-Peer Systems and Applications*, volume 3485 of *Lecture Notes in Computer Science*, chapter Random Graphs, Small-Worlds and Scale-Free Networks, pages 57–76. Springer Berlin/Heidelberg, 2005.

[44] Hock B. Lim, Yong M. Teo, Protik Mukherjee, Vinh T. Lam, Weng F. Wong, and Simon See. Sensor grid: Integration of wireless sensor networks and the grid.

[45] Ting Liu and Margaret Martonosi. Impala: a middleware system for managing autonomic, parallel sensor systems. In *PPoPP '03: Proceedings of the ninth ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 107–118, New York, NY, USA, 2003. ACM Press.

[46] Keong Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim. A survey and comparison of peer-to-peer overlay network schemes. *Communications Surveys & Tutorials, IEEE*, pages 72–93, 2005.

[47] Michael R. Lyu, Jia-Hong Chen, and Algirdas Avizienis. Software diversity metrics and measurements. In *IN PROC. THE SIXTEEN ANNUAL INT. COMPUTER SOFTWARE AND APPLICATIONS CONF*, pages 69–78, 1992.

[48] C. Matthew MacKenzie, Ken Laskey, Francis McCabe, Peter F. Brown, and Rebekah Metz. Oasis reference model for service oriented architecture.

[49] Samuel R. Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. Tinydb: an acquisitional query processing system for sensor networks. *ACM Trans. Database Syst.*, 30(1):122–173, 2005.

[50] Pedro José Marrón, Thiemo Voigt, Peter I. Corke, and Luca Mottola, editors. *Real-World Wireless Sensor Networks - 4th International Workshop, REALWSN 2010, Colombo, Sri Lanka, December 16-17, 2010. Proceedings*, volume 6511 of *Lecture Notes in Computer Science*. Springer, 2010.

[51] S. Marwaha, J. Indulska, and M. Portmann. Challenges and recent advances in qos provisioning, signaling, routing and mac protocols for manets. In *Telecommunication Networks and Applications Conference, 2008. ATNAC 2008. Australasian*, pages 97 –102, dec 2008.

[52] Rainer Matischek. *A TinyOS-Based Ad Hoc Wireless Sensor Network: Introduction, Versatile Application Design, Implementation*. VDM Verlag, Saarbrücken, Germany, Germany, 2008.

[53] S. Mccanne, S. Floyd, and K. Fall. ns2 (network simulator 2). http://www-nrg.ee.lbl.gov/ns/.

[54] H Mcheick H Milli. Understanding separation of concerns. *In Workshop on Early Aspects - Aspect Oriented Software Development (AOSD04)*, 2004.

[55] Luca Mottola, Gian P. Picco, and Adil A. Sheikh. Figaro: Fine-grained software reconfiguration for wireless sensor networks. In Roberto Verdone, editor, *EWSN*, volume 4913 of *Lecture Notes in Computer Science*, pages 286–304. Springer, 2008.

[56] Ioanis Nikolaidis, Michel Barbeau, and Evangelos Kranakis, editors. *Ad-Hoc, Mobile, and Wireless Networks: Third International Conference, ADHOC-NOW 2004, Vancouver, Canada, July 22-24, 2004. Proceedings*, volume 3158 of *Lecture Notes in Computer Science*. Springer, 2004.

[57] Adam J. O'Donnell and Harish Sethu. On achieving software diversity for improved network security using distributed coloring algorithms. In *Proceedings of the 11th ACM conference on Computer and communications security*, CCS '04, pages 121–131, New York, NY, USA, 2004. ACM.

[58] Stephan Olariu, Mohamed Eltoweissy, and Mohamed Younis. Answer: autonomous wireless sensor network. In *Q2SWinet '05: Proceedings of the 1st ACM international workshop on Quality of service & security in wireless and mobile networks*, pages 88–95, New York, NY, USA, 2005. ACM Press.

[59] OMG. *The CORBA Component Model*, 2002. OMG doc. formal/2002-06-65.

[60] Hyojin Park, Jinhong Yang, Juyoung Park, Shin Gak Kang, and Jun Kyun Choi. A survey on peer-to-peer overlay network schemes. In *Advanced Communication Technology, 2008. ICACT 2008. 10th International Conference on*, volume 2, pages 986 –988, feb 2008.

[61] Terence J. Parr and Russell W. Quong. Antlr: A predicated-ll(k) parser generator. *Software Practice and Experience*, 25:789–810, 1994.

[62] C.E. Perkins and E.M. Royer. Ad-hoc on-demand distance vector routing. In *Mobile Computing Systems and Applications, 1999. Proceedings. WMCSA '99. Second IEEE Workshop on*, pages 90 –100, February 1999.

[63] M. Petrovic, Vinod Muthusamy, and H. A. Jacobsen, editors. *Content-based routing in mobile ad hoc networks*, 2005.

[64] Roberto Pietro, Luigi Mancini, and Alessandro Mei. Energy efficient node-to-node authentication and communication confidentiality in wireless sensor networks. *Wireless Networks*, 12:709–721, 2006. 10.1007/s11276-006-6530-5.

[65] Rajiv Ranjan, Aaron Harwood, and Rajkumar Buyya. Grid federation: An economy based, scalable distributed resource management system for large-scale resource coupling.

[66] A. Rezgui and M. Eltoweissy. Service-driven query routing in sensor networks. In *Proceedings of the 1st IEEE International Workshop on Practical Issues in Building Sensor Network Applications, in conjunction with LCN 2006, Australia*, 2006.

[67] A. Rezgui and M. Eltoweissy.
textmu racer: A reliable adaptive service-driven efficient routing protocol suite for sensor-actuator networks. *IEEE Transactions on Parallel and Distributed Systems*, 20(5), 2009.

[68] Paul Robertson and Brian Williams. Automatic recovery from software failuoodtinydbagillaagillaagillaagillaretosre. *Commun. ACM*, 49:41–47, 2006.

[69] Mohammad Samarah, , Mohammad Samarah, and Philip Chan. Distributed communication for highly mobile agents.

[70] Tobias Senner, Reinhardt Karnapke, Andreas Lagemann, and J? Nolte. A combined routing layer for wireless sensor networks and mobile ad-hoc networks. *Sensor Technologies and Applications, International Conference on*, 0:147–153, 2008.

[71] John A. Sharp. Data oriented program design. *SIGPLAN Not.*, 15:44–57, September 1980.

[72] Munindar P. Singh. Toward interaction-oriented programming. Technical report, 1996.

[73] Vivek K. Singh and Ramesh Jain. Situation based control for cyber-physical environments. In *Proceedings of the 28th IEEE conference on Military communications*, MILCOM'09, pages 2198–2204, Piscataway, NJ, USA, 2009. IEEE Press.

[74] Jan Steffan, Ludger Fiege, Mariano Cilia, and Alejandro Buchmann. Towards multi-purpose wireless sensor networks. In *Proceedings of the 2005 Systems Communications*, pages 336–341, Washington, DC, USA, 2005. IEEE Computer Society.

[75] SUN MICROSYSTEMS. *Java Remote Method Invocation Specification, JDK 1.2*. Sun Microsystems, Mountain View, Calif., October 1998.

[76] Tetsuo Tamai, Naoyasu Ubayashi, and Ryoichi Ichiyama. An adaptive object model with dynamic role binding. In *ICSE '05: Proceedings of the 27th international conference on Software engineering*, pages 166–175, New York, NY, USA, 2005. ACM.

[77] Luminita Vasiu and Qusay H. Mahmoud. Mobile agents in wireless devices. *Computer*, 37(2):104–105, 2004.

[78] Jennifer Yick, Biswanath Mukherjee, and Dipak Ghosal. Wireless sensor network survey. *Comput. Netw.*, 52:2292–2330, August 2008.