Modeling, Analysis, and Self-Management of Electronic Textiles

Phillip Stanley-Marbell, *Student Member*, *IEEE*, Diana Marculescu, *Member*, *IEEE*, Radu Marculescu, *Member*, *IEEE*, and Pradeep K. Khosla, *Fellow*, *IEEE*

Abstract—Scaling in CMOS device technology has made it possible to cheaply embed intelligence in a myriad of devices. In particular, it has become feasible to fabricate flexible materials (e.g., woven fabrics) with large numbers of computing and communication elements embedded into them. Such *computational fabrics*, *electronic textiles*, or *e-textiles* have applications ranging from smart materials for aerospace applications to wearable computing. This paper addresses the modeling of computation, communication and failure in e-textiles and investigates the performance of two techniques, *code migration* and *remote execution*, for adapting applications executing over the hardware substrate, to failures in both devices and interconnection links. The investigation is carried out using a cycle-accurate simulation environment developed to model computation, power consumption, and node/link failures for large numbers of computing elements in configurable network topologies. A detailed analysis of the two techniques for adapting applications to the error prone substrate is presented, as well as a study of the effects of parameters, such as failure rates, communication speeds, and topologies, on the efficacy of the techniques and the performance of the system as a whole. It is shown that code migration and remote execution provide feasible methods for adapting applications to take advantage of redundancy in the presence of failures and involve trade offs in communication versus memory requirements in processing elements.

Index Terms—Electronic textiles, sensor networks, wearable computing, fault tolerance, application remapping, code migration, remote execution.

1 Introduction

The scaling of device technologies has made possible significant increases in the embedding of computing devices in our surroundings. Embedded microcontrollers have for many years surpassed microprocessors in the number of devices manufactured. The new trend, however, is the networking of these devices and their ubiquity not only in traditional embedded applications such as control systems, but in items of everyday use, such as clothing, and in living environments.

A trend deserving particular attention is that in which large numbers of simple, cheap processing elements are embedded in environments. These environments may cover large spatial extents, as is typically the case in *networks of sensors*, or may be deployed in more localized constructions, as in the case of *electronic textiles*. These differing spatial distributions also result in different properties of the networks constituted, such as the necessity to use wireless communication in the case of sensor networks and the feasibility of utilizing cheaper wired communications in the case of electronic textiles.

Electronic textiles, or e-textiles, are a new emerging interdisciplinary field of research, bringing together specialists in information technology, microsystems, materials, and textiles. The focus of this new area is on developing the enabling technologies and fabrication techniques for the economical manufacture of large-area, flexible, conformable information systems that are expected to have unique applications for both the consumer electronics and aerospace/military industries. They are naturally of particular interest in wearable computing, where they provide lightweight, flexible computing resources that that are easily integrated or shaped into clothing.

Due to their unique requirements, e-textiles pose new challenges to hardware designers and system developers, cutting across the systems, device, and technology levels of abstraction:

- The need for a new model of computation intended to support widely distributed applications, with highly unreliable behavior, but with stringent constraints on the longevity of the system.
- Reconfigurability and adaptability with low computational overhead. E-textiles must rely on simple computing elements embedded into a fabric or directly into *active yarns*. As operating conditions change (environmental, battery lifetime, etc.), the system has to adapt and reconfigure on-the-fly to achieve better functionality.
- Device and technology challenges imposed by embedding simple computational elements into fabrics, by building yarns with computational capabilities, or by the need for unconventional power sources and their manufacturing in filament form.

In contrast to traditional wearable computers, which are often a single monolithic computer or a small computer system that can be worn, e-textiles will be cheap, general purpose computing substrates in the form of a woven fabric that can be used to build useful computing and sensing systems "by the yard" [1].

• *The authors are with the Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA 15213. E-mail: {pstanley, dianam, radum, pkk}@ece.cmu.edu.*

Manuscript received 1 July 2002; revised 29 Nov. 2002; accepted 28 Jan. 2003. For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 118290.

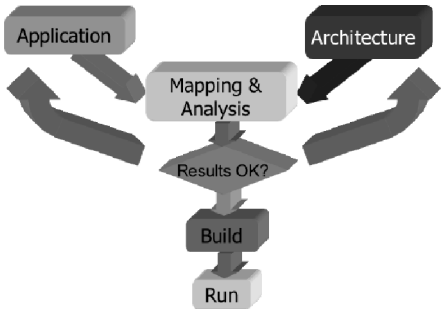


Fig. 1. Classical static design cycle—no remapping occurs after the initial design is built.

Fig. 2. Dynamic continuously adapting *Living Designs*—through continuous online monitoring, the mapping of applications to the hardware substrate may evolve over time.

At the architecture layer, we expect to have a large number of resource-constrained computing elements which will be defect prone (i.e., manufacture-time defects will be common due to difficulties and costs of testing). In addition, the devices will witness significant fault rates during their lifetime due to the nature of the environments in which devices will be deployed and due to the psychological treatment of these devices by humans since they will no longer be regarded as computing devices per se. These individual elements are interconnected in a network which is likewise subject to defects and faults, for similar reasons as the individual computing elements. The computing elements connected in the interconnection network must be able to provide some useful functionality, with stringent constraints on *quality of results* or *operational longevity*.

Techniques to program such networks are required that permit useful applications to be constructed over the defect and fault-prone substrate. There is a need for a new model of computation to support distributed application execution with highly unreliable behavior at the device-level, but with stringent constraints on longevity at the system level. Such a model should be able to support local computation and inexpensive communication among computational elements. In the classical design cycle (Fig. 1), the application is mapped onto a given platform architecture, under specified constraints (performance, area, power consumption). When these constraints are met, the prototype is tested, manufactured, and used for running the application. In the case of e-textiles (Fig. 2), the substrate is comprised of large numbers of interconnected computing elements, with no prescribed functionality. To achieve high yields (that is, low defect rate), as well as high fault-tolerance later in the lifetime cycle, regularity is important. The application is modified (*partitioned*) so as to expose as much local computation as possible. At power-up, the application is mapped so as to optimize different metrics of interest (such as quality of results, power consumption, operational longevity, fault tolerance) and later remapped whenever operating conditions change.

Although e-textiles ostensibly present distributed computing challenges similar to those currently being pursued in adaptive control networks and pervasive computing, the specific characteristics and demands of the e-textile hardware platform add new dimensions to those challenges. E-textiles impose specific challenges as opposed to other applications in the general area of networked systems:

1.1 What E-Textiles Are *Not*

- *Classic data networks*. While the underlying structure of an e-textile application implies the existence of many processing elements, connected in a Textile-Area Network (TAN), they have limited processing and storage capabilities, as well as very limited power consumption budgets. Hence, classic techniques and inherent methodologies for coping with mapping an application, communication among nodes, and dealing with network failures are not appropriate. In addition, having very limited processing capabilities, e-textiles are not the equivalent of "desktops/laptops on a fabric,", significantly restricting the range of applications that can be mapped on them.
- *Networked embedded systems* (including wireless sensor networks). Wireless sensor networks share with e-textiles the constraints of limited power budgets and, to some extent, the limited processing and storage capabilities. However, communication among processing elements in e-textiles is wired and, thus, much less expensive than communication in wireless sensor networks. In addition, as opposed to ad hoc networks, the topological location of different processing elements is fixed throughout the lifetime of the application (although the mapping of the application on processing elements may change). Last, e-textiles must have low manufacturing costs and, thus, the defect rate of the processing nodes (and physical links) will be much higher and very different than in the case of wireless networks. More limited processing and storage capabilities, in conjunction with higher failure rates, imply that the existing body of research for sensor networks is not directly applicable to TANs.
- *Defect-free reliable mobile systems*. E-textiles must have low manufacturing costs and, thus, the defect rate of the processing nodes (and physical links) will presumably be much higher than in the case of traditional mobile wireless networks.

1.2 What E-Textiles Are

E-textiles are "living designs" consisting of highly unreliable, low-cost, simple components and interconnect. These characteristics are shared by other nonsilicon computing systems (such as those based on nanoelectronics). In many

ways, e-textile systems will be the conformable, fabric equivalent of rigid body, Micro-Electro-Mechanical Systems (MEMS). However, unlike MEMS, e-textiles require coping with tears in the fabric, whether it is due to regular wear of the fabric or due to unexpected damage. Some of the salient features of e-textiles are:

- limited processing, storage and energy per computational node,
- potential failures for both nodes and communication links,
- highly *spatially and temporally correlated* node and/or link failures due to topological placement or due to external events,
- need for scalability and flexibility of resource management,
- a marked difference between traditional computing systems and e-textile-based systems is the possibility of *incorporating computational and signal processing capabilities directly into communication links*. These "active yarns" may perform signal conditioning and data processing in situ and may contain devices such as A/D converters or simple computing elements (i.e., adders, multipliers, pseudorandom number generators).

This paper presents techniques for addressing many of the aforementioned challenges at different layers of abstraction. At the lowest layer, we present techniques for achieving useful computation from individual unreliable computing elements and for adapting to runtime failures. One level above, we show how the individual elements can monitor the runtime failure rates and adapt to these failures in what we call *on-the-fly fault tolerance management*, by analogy to *power management*. At an even higher level, we introduce the concept of *colloidal computing* as a model for structuring systems comprised of unreliable computing elements and interconnects.

The next section presents the relation of various aspects of our work to already existing efforts. The computing model we propose is presented in Section 3. Section 4 discusses issues related to application partitioning and presents a driver application that we shall use in the analysis throughout the remainder of the paper. Section 5 describes the communication architecture employed in our experimental evaluation and presents two techniques for adapting applications running on the individual computing elements, in the presence of failures in the elements themselves and in the networks that interconnect them. Section 6 outlines the simulation framework that was employed in the subsequent investigations and provides a detailed experimental evaluation of the ideas presented in the paper. The ideas and results presented are summarized in Section 7, along with projections for future directions of this work.

2 Related Work

There have been a handful of attempts to design and build prototype computational textiles. In [2], the authors demonstrate attaching off-the-shelf electrical components, such as microcontrollers, surface mount LEDs, piezoelectric transducers, etc., to traditional clothing material, transforming the cloth into a breadboard of sorts. In fabrics which contain conductive strands, these may be used to provide power to the devices as well as to facilitate communication between devices. In [3], the authors extend the work presented in [2], detailing methods by which items such as user interfaces (keypads) and even chip packages may be constructed directly by a textile process.

The routing of electrical power and communications through a wearable fabric was addressed in [4]. In [4], the authors provide a detailed account of physical and electrical components for routing electricity through suspenders made of a fabric with embedded conductive strands. The authors also detail the physical construction of a battery holder to be attached to this power grid, as well as a data-link layer protocol for interconnecting devices on a Controller Area Network (CAN) bus, also implemented with the strands embedded in the fabric.

A complete apparel with embedded computing elements is described in [5]. The authors describe a jacket designed to be worn in the harsh arctic environment, which augments the capabilities of the wearer with a global positioning system (GPS), sensors (accelerometers, conductivity electrode, heart rate monitors, digital thermometers), and heating. All the components obtain power from a central power source and the user interacts with them through a single user interface.

The "wearable motherboard" project [6] is a substrate that permits the attachment of computation and sensing devices in much the same manner as a conventional PC motherboard. Its proposed use is to monitor vital signs of its wearer and perform processing. Proposed applications include monitoring vital statistics of soldiers in combat.

Adaptive techniques such as code migration and remote execution have previously been employed in server and mobile computing systems. The use of remote invocation to reduce the power consumption in mobile systems has previously been investigated in [7], [8]. The goal in both of these efforts was to reduce power consumption by offloading tasks from an energy constrained system to a server without constraints in energy consumption. The trade off involved determining when it was worthwhile to ship data to the remote sever. Both approaches involved the use of remote invocation and not code migration. The systems investigated were mobile computers and PDAs operating with fast reliable wireless networking, an environment very different from low power, unreliable, network sensors with possibly unreliable communication investigated in this paper.

Process migration is the act of transferring a process between two machines (the *source* and *destination* nodes), during its execution [9]. Process migration has traditionally been employed in distributed systems of servers and workstations, primarily for load distribution and fault tolerance. Unlike the traditional implementations of process migration, the code migration implementation studied in this paper is significantly lighter weight, taking into consideration the special properties of the target application class.

The use of code migration has been successfully applied in the field of mobile agents [10], [11], [12]. Mobile agents are an evolution of the general ideas of process migration. They can be thought of as autonomous entities that determine their traversal though the network, moving their code as well as state as they do so. The examples employed in this study may be loosely considered to fall into the category of mobile agents. Our focus here is not on the

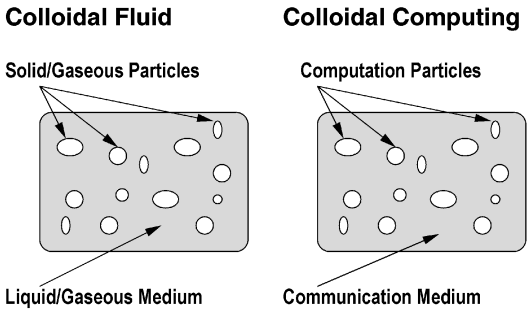


Fig. 3. Colloidal computing model—analogy to colloidal chemistry.

techniques for constructing mobile agents, but rather on the basic principles of employing code migration in the presence of redundancy and energy constraints.

It has previously been proposed [13] to employ redundantly deployed nodes in a sensor network to increase the operational lifetime of the network. There have been proposals such as [14] that exploit redundancy to support routing of data in wireless ad hoc networks and thereby save energy. In contrast to these previous efforts, this paper investigates the feasibility of employing code migration in the presence of redundancy to increase the operational lifetime of a system, specifically looking at the cost of migration in terms of energy consumption and attendant techniques to make this feasible, in the presence of high failure rates.

3 Colloidal Computing

The *Model of Colloidal Computing* (MC^2) [15] supports local computation and inexpensive communication among computational elements: Simple computation particles are "dispersed" in a communication medium which is inexpensive, (perhaps) unreliable, yet sufficiently fast (Fig. 3). This concept has been borrowed from physical chemistry [16][1] since some of its properties and characteristics resemble the features that an e-textile-based computational model would require. In the case of unstable colloidal suspensions, colloidal particles tend to *coalesce* or *aggregate* together due to the Van der Waals and electrostatic forces among them. Coalescing reduces surface area, whereas aggregation keeps all particles together without merging. Similarly, the resources of a classic system are coalesced together in a compact form, as opposed to the case of e-textile-based computation where useful work can be spread among many, small, (perhaps) unreliable computational elements that are dynamically aggregated depending on the needs (Fig. 4). *Dynamic or adaptive aggregation* is explicitly performed whenever operating conditions change (e.g., failure rate of a device is too high or battery level is too low).

The MC^2 model [15] was previously proposed to model both the application software *and* architecture platform. Most of the applications under consideration consist of a number of computational kernels with high spatial locality, but a low degree of communication among them. Such kernels (typically associated with media or signal processing applications) can thus be mapped on separate computational "particles" that communicate infrequently for exchanging results.

1. *Colloid* [käl'oid] = a substance consisting of very tiny particles (between 1nm and 1000nm), suspended in a continuous medium, such as liquid, a solid, or a gaseous substance.

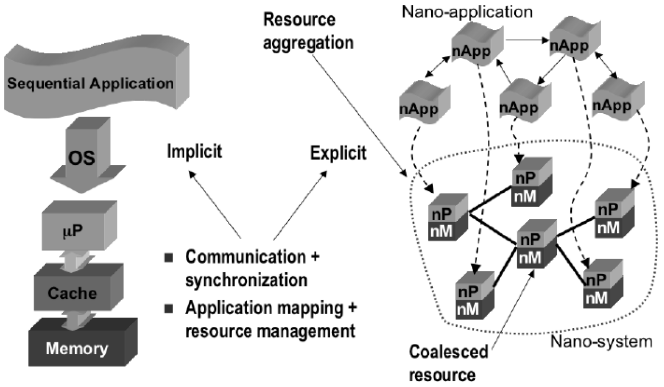


Fig. 4. Coalesced versus aggregated resources: partitioning applications to expose concurrency.

The underlying architecture of the e-textile matches the proposed MC^2 model. Both the architecture configuration and the mapping of the application on the computational elements is done dynamically, at power-up and whenever the system becomes "unstable." As an analogy, in the case of unstable colloids, a minimal energy configuration is achieved via coalescing (e.g., oil in water) or aggregation (e.g., polymer colloids). In e-textiles, a "stable" configuration is one that achieves the required functionality, within prescribed performance, power consumption, and probability of failure limits. We propose employing aggregation (Fig. 4) or *dynamic connection* of computational "particles" based on their state (i.e., functional or not, idle or active) and their probability of failure so as to achieve a required quality of service (QoS) (characterized by performance, power, and fault tolerance). The mapping and reconfiguration process of the application onto the underlying architecture is achieved via *explicit* mechanisms, as opposed to classic computing systems where mapping and resource management is done via *implicit* mechanisms.

Reorganization and remapping requires thin *middleware* or *firmware* clients, sufficiently simple to achieve the required goals without prohibitive overhead. In addition, fault and battery modeling and management become intrinsic components for achieving requisite levels of quality of results or *operational longevity*. We describe in the sequel some of the issues that are critical to e-textile application lifetime, namely, application modeling and partitioning, architecture, and communication modeling, as well as fault modeling and management.

4 Application Partitioning

A fundamental issue which should be addressed in order to efficiently map complex applications onto e-textiles is that of concurrency. Given that the e-textiles will contain many computational particles distributed on large surfaces, it is of crucial importance to expose the concurrency available in applications since this will dictate the ultimate limit of parallelism that may be obtained when they are partitioned to execute on e-textiles. The methods by which such concurrency may be extracted are a very interesting research avenue in their own right. In this work, a driver application that is trivially partitioned was employed.

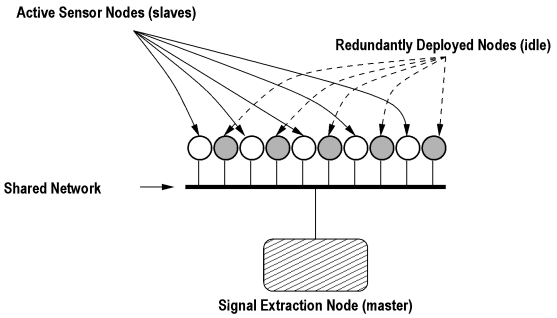


Fig. 5. Beamforming in a wired sensor network.

4.1 Driver Application: Beamforming

For the analysis of the mechanisms for adaptation that will be described in later sections, the driver application is that of *beamforming*. This application is intended to highlight the effect of different parameters on the benefits that can be derived from the architectures and adaptation techniques to be discussed.

4.1.1 Beamforming in Wired Sensor Network

Beamforming consists of two primary components—*source location* and *signal extraction*. It is desirable to detect the location of a signal source and "focus" on this source. The signals from spatially distributed sensors are sent to a central processor, which processes them to determine the location of the signal source and reconstruct a desired signal. Each received sample is filtered and this filtering could indeed be performed at the sensor. Fig. 5 illustrates the organization for a wired network of sensors used to perform beamforming, deployed, for example, on an e-textile.

The beamforming application is easily partitioned to run over an e-textile. The filtering operation on each collected sample can be considered to be independent of other samples, thus it could be performed individually at each sensor node (*slave node*). The final signal extraction need only be performed at one node (*master node*). This division of tasks scales well with an increasing number of sensors since the complexity of processing at each sensor node remains the same and the only increase in complexity is in the number of filtered samples collected at the master.

Our example system operates in periods, during each of which all the slaves collect samples, filter them, and send the filtered samples to the master. The duration of the sampling period will differ for different applications of beamforming. In the case of beamforming for speech applications, an overall sampling rate of 8KHz is sufficient. For geophysical phenomenon, a sampling rate of 1KHz is sufficient. For applications such as tracking motion of animals, a sampling rate of 10Hz is sufficient. In the analysis used throughout the rest of the paper, a sampling rate of 10Hz corresponding to a 100 millisecond sampling period is used. Using a larger sampling rate would shift all the results by a constant factor, but would not change the general trends observed and insights provided.

The communication messages between the master and slave nodes consist of 4 byte data packets containing the digitized sample reading. When the battery level on any of the slave nodes falls below a specified threshold, the slave application attempts to use its remaining energy resources to migrate to one of the redundant nodes. If migration is successful, the slave application resumes execution on the redundant node and adjusts its behavior for the fact that it is now executing on a different sensor, which is detected when it restarts. The migrated application code and data for the slave application is small (only 14K). The application on the processing elements with attached sensors implemented a 32-tap FIR filter and consists of 14,384 bytes of application code and 648 bytes of application state. The application mapped on the sample aggregation (master) node performs a summation over the samples and is never migrated in the experiments that follow. The sequence of messages that are exchanged between the master and slaves during normal operation and between the slaves and redundant nodes during migration is illustrated in Fig. 7.

5 Communication Architecture and Fault Management

Achieving reliable computation in the presence of failures has been an active area of research dating back to the early years of computing [17], [18]. Unlike large networked systems in which failure usually occurs only in communication links or in computational nodes and communication links with low correlation, in the case of e-textiles, nodes and links coexist in close physical proximity and thus witness a high correlation of failures.

An important issue is fault modeling, according to their type (electrical versus mechanical, intermittent, or permanent). Intermittent failures, such as those due to electrical failures, tend to follow a uniform failure probability distribution in which the failure probability remains constant over time. Mechanical failures, on the other hand, can be modeled with an exponential failure probability distribution, where each failure increases the probability of subsequent failures. A special class of permanent faults are those due to battery depletion. They have the advantage of being predictable, given a sufficiently accurate battery model.

It is assumed that the application is initially mapped at system power-up for given quality of results (QoR), power, and fault tolerance constraints. As operating conditions change (e.g., permanent failures due to fabric wear and tear or intermittent failures due to battery depletion), the entire application (or portions of it) will have to be remapped and/or communication links rerouted (Fig. 6). Such reconfiguration mechanisms assume that redundancy exists for both nodes and links. Many aproaches exist for providing such redundancy in nodes and links. In a fixed infrastructure, the logical implementation of redundancy is to replicate resources, with one resource taking over on the failure of the other. Upon such failures, applications must be *remapped*, for example, by *code migration* or *remote execution*. Code migration is generally a difficult problem as it could, in the worst case, require the movement of the entire state of an executing application. However, migration of running applications can be greatly simplified by restricting the amount of application state that must be preserved.

In what follows, the potential benefits of using code migration and remote execution for alleviating the effects of finite battery lifetime on operational longevity constrained e-textiles in the presence of intermittent electrical failures will be discussed.

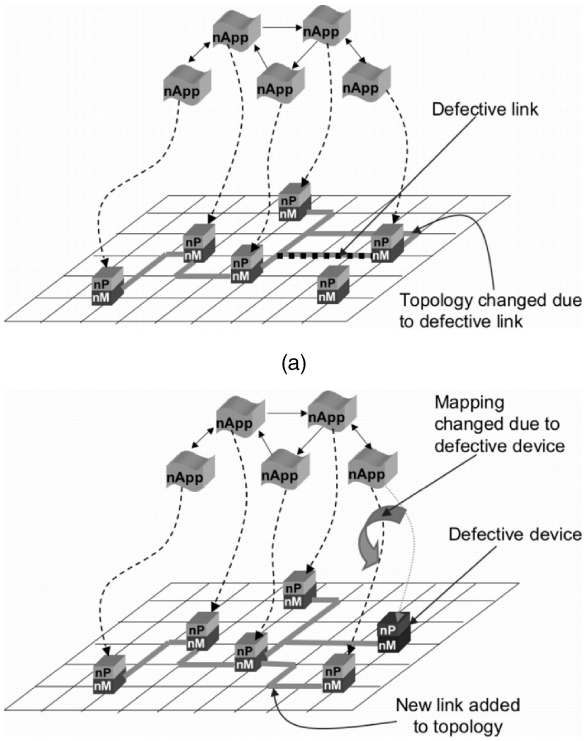


Fig. 6. Dynamic reconfiguration in the presence of failures. In the event of a defective link (a), the network topology mapping is adapted to circumvent the faulty link, likewise in the case of a defective node, mapping and topology change to use a redundant node and a different link (b).

5.1 Application Remapping by Lightweight Code Migration

Many difficulties and trade offs exist in implementing code migration and the solution proposed here is a compromise between implementation complexity and flexibility.

In the ideal case, migrating an executing application will mean the application itself can be unaware of this move and can continue execution on the destination host without any change in behavior. This transparent migration requires that all state of the executing application (code, data, machine state, and state pertinent to any underlying system software) must be migrated faithfully. Such migration is, however, costly in terms of data that must be transmitted to the target host and in terms of implementation complexity.

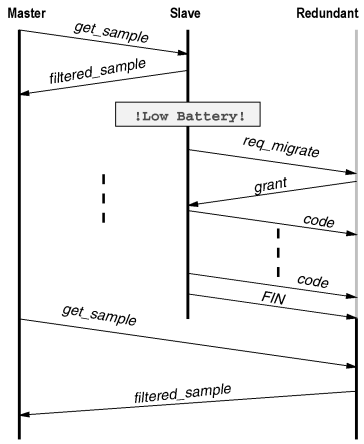


Fig. 7. Messages exchanged in beamforming application.

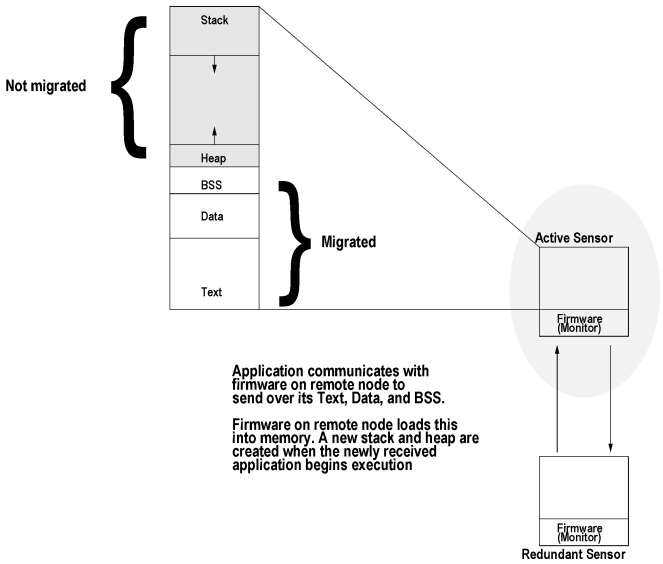


Fig. 8. Migration of application code and state.

A compromise is to implement applications in a manner in which they can be *asynchronously restarted*, while maintaining persistence for any important state. This then makes it possible to migrate only the state necessary for correct execution upon a restart, in addition to the migration of application code. Fig. 8 illustrates such a solution. The textile processing node consists of a processor and memory. The memory space is partitioned between that used for the application and memory dedicated to the devices firmware. For example, in many embedded systems, firmware may run in SRAM while applications run in DRAM or both may be placed in different regions of a single memory. The memory region in which the application runs is occupied by the different parts of the running application—its code (`text`), initialized data (`data`), uninitialized data (`bss`), stack (grows downward from the top of the memory region), and heap (grows upward from `bss`), as illustrated by the blow-up in Fig. 8.

Global initialized (uninitialized) variables reside in the `data`(`bss`) segment. By placing information that must be persistent across restarts in the `data` and `bss` segments of the application, it is possible to maintain state across migration while only transferring the `text`, `data`, and `bss` segments. Applications must, however, be designed to check the values of these persistent variables on each start of execution and must generally be implemented to be able to continue correct execution in the presence of such restarts. For example, the ID of the original node on which the application was launched is kept in the data segment during migration, enabling the application to resume its old identity on the new host. This constraint to application construction is reasonable and was used to successfully implement code migration in the examples described in this paper.

Each node has preloaded into it firmware referred to as the *monitor*. This firmware is responsible for receiving applications and loading them into memory. Each application can transmit its `text`, `data`, and `bss` (uninitialized data) segments and, when these are loaded by a remote monitor, they effect the migration. Upon loading an application into

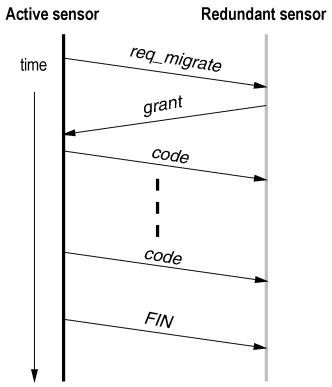


Fig. 9. Message exchange during migration of application.

memory, control of the hardware is transferred to the loaded code. On completion of execution, control returns to the monitor.

The sequence of messages that are exchanged between a node attempting to migrate and one that receives it is illustrated in Fig. 9. A node that wants to migrate first sends a migration request (`req_migrate`) out to a remote node. In practice, this initial migration request might often be broadcast. A node that is capable of receiving a migrating application responds to the migration request with a `grant` message. The migrating application then sends over its `text`, `data`, and `bss` segments, which the remote node loads into memory. Finally, the migrating application sends an `FIN` message to the remote node indicating that it is done sending its image over. The monitor application on the remote node then transfers control of execution to the newly loaded code in memory and migration is complete.

5.2 Application Remapping by Remote Execution

The technique we refer to here as *remote execution* is also often referred to as remote invocation. It entails replicating the code for each individual application that will ever be run in the network, at each node. The storage of the applications at nodes must be persistent across time, in the presence of failures at the nodes and failures in the links that interconnect them. The applications must therefore be stored in some form of nonvolatile memory.

Consider two different scenarios that may arise when employing remote execution to take advantage of redundant resources in a network. Both cases involve two nodes, one whose energy resources are close to exhaustion and the other with a full supply of energy. In the first scenario, the nodes are identical, and interchangeable, with neither having any distinguishing resources (except remaining energy storage) and both having identical copies of the application code. In such a situation, the energy supply of one node could be harnessed by the other simply by migrating only the *state* of the application to the device with more energy and resuming execution on that node.

A different scenario, however, arises if the two nodes under discussion do not have identical resources. Not having identical resources could mean one of many different things:

- The nodes have different code. This would occur if one node was deployed at a different time than the other, with the application of interest being available only during the deployment of the first node (or vice versa). It could also mean different versions of the same application, one not being fully substitutable for the other (for example, the later version could contain critical bug fixes). It could also be due to one of the nodes simply not being capable of storing more than one application in nonvolatile storage.
- The nodes have identical code, but different resources, such as sensors. It can be expected that devices in the network might have different attached sensors to provide a rich range of information sources for the system as a whole. In such a situation, one node is not substitutable for the other unless the data to be processed, such as a set of readings from a sensor, will be transmitted to the replacement device during migration.

In the first case of different code at each node above, remote execution is not feasible since the underlying requirement in remote execution is that all application code be duplicated at all elements in the network. In the second case, remote execution is only useful if the ability to process *just one* set of data readings (the set migrated with the application state during remote invocation) is critical to system functionality. This would be the case, for example, if the system correctness criteria require that a set of sample readings from sensors should never be discarded.

Remote invocation is therefore best suited for situations in which all nodes are identical, in terms of both hardware and software deployed. It has the benefits that, when applicable, it can minimize the amount of data that must be transmitted since the migrated state is always smaller than the combination of state and application code, which is the case for code migration. It, however, has the disadvantages that the flexibility of the system is reduced (requires any node pair to have identical code stored), the cost of the system is increased (nonvolatile storage such as Flash can be as much as twice the cost of DRAM), and the power consumption is likewise increased, due to the added storage device.

6 Experimental Evaluation

As a newly emerging field, e-textiles impose new challenges not only on modeling and analysis, but also on simulation tools used for validation. It is desirable to employ a simulation environment built around a processor architectural instruction set simulator, able to provide detailed power consumption and performance results for the execution of real partitioned applications on processing elements (i.e., for example, compiled binaries). The simulation testbed must include detailed *battery models*, accurate *communication link modeling*, as well as *failure modeling*. Since processors, active, or passive links may each fail independently, or may have correlated failures, multiple *failure models* have to be supported. The simulation environment should enable modeling of *failure probabilities*, *mean duration of failures*, and *failure probability distributions* for each of the *nodes* and *communication links*. Furthermore, failures between nodes and links may be correlated and an environment should enable the specification of *correlation coefficients* between the failure probabilities of any given network segment and any processing element.

A simulation framework that meets the above criteria was developed with the express purpose of performing

cycle-accurate investigations of e-textile-based systems, both in terms of performance/functionality and in terms of power dissipation and battery lifetime effects. This simulation framework was employed to investigate the impact of node/link failure rates and correlations among failures on performance, as well as the feasibility of using code migration versus remote execution for increasing operational longevity.

6.1 Simulation Framework

The simulation framework used in this study enables the cycle-accurate modeling of computation (instruction execution), communication, failure (in both the interconnection network and processing elements), batteries, and different network topologies. It was designed to enable the functional and power consumption/battery discharge simulation of moderately sized (approximately 100 nodes) networks of embedded processors. We routinely employ this simulation framework to simulate networks with of the order of 50 processors with an equal number of batteries and 50 network links. The processing elements may be connected together in arbitrary topologies (e.g., multiple processing elements to a single shared link, point-to-point links, etc.) and they may likewise be connected many-to-one with batteries, subject to maximum current draw restrictions imposed by the battery models.

At the core of the simulation framework is an energy estimating architectural simulator for the Hitachi SuperH embedded processor architecture [19]. It models a network of embedded processors, each consisting of a Hitachi SH3 microcontroller, memory, and communication interface. The modeled processor in each system may be configured to run at different voltages and, hence, operating frequency. The time-scale of all of these components is synchronized, making possible cycle accurate estimation of both computation and communication performance. The communication links between the modeled embedded systems are cycle-accurate with respect to the simulation of each processor in the network. The mapping between simulation cycles and simulated elapsed time is dictated by the configured operating frequency. For example, if nodes run at 60MHz, each clock tick corresponded to 16.6667ns. Each processing element may be attached to a battery of configurable capacity, with battery models based on the discrete-time models described in [20].

The simulation environment permits the instantiation of a large number of nodes (the limit is currently 512 and the absolute limit is determined by the amount of memory in the simulation host machine), as well as interconnection links. Each node may be configured with one or more network interfaces, each of which may be attached to one of the instantiated communication links, in an arbitrary topology. The links may be configured in terms of their transmission delays (i.e., bit-rate), frame size, failure probability, and whether they permit single or multiple simultaneous accesses.

The power estimation performed for each individual processor is based on an instruction level power model. The model used within the simulator has been verified to be within 6.5 percent of measurements from the modeled hardware [19]. The power estimation for the communication links assigns fixed costs for transmission and receiving, respectively. In the experiments, costs of 100mW for both transmission and receiving were used for the wired network.

In the experiments presented in the following sections, the nodes were attached to batteries which were sized for each set of experiments so as to limit simulation time. The lifetimes of the simulated batteries were thus of the order of a few seconds since simulating the execution of a system for, say, one day would take over an order of magnitude more simulation time for each simulated configuration and would not necessarily provide additional insight.

The beamforming application was implemented in C and compiled with GCC [21] to generate code for the target architecture modeled in the simulation environment. The application was constructed as a typical event-driven embedded application, which runs directly over a microcontroller in the absence of a separate operating system layer. It consists of two primary parts—the application code and interrupt handlers. Execution of the application code is triggered by the occurrence of events (arrival of a data packet, elapse of a period of time). This is a typical implementation in small embedded software and has even been used as the model for an operating system for networked sensors [22].

6.2 Effects of System Parameters on Application Remapping

The key focus in this simulation study and analysis is the effect of system parameters such as node and link failure probabilities, link speed, and link frame size on the utility of remote execution and code migration to prolong system lifetime. Issues such as topology discovery, routing in large sensor networks, and node discovery have received attention elsewhere, for example, in [23], [24], [25], [26], and are beyond the scope of this paper.

6.2.1 Effect of Node and Network Configuration on Baseline Performance

Before delving into details on the effect of various node and network architectures on the cost and utility of remapping, it is instructive to note the effect of these parameters on the performance of a baseline system without remapping.

To this end, the effect of link transmission delay, link frame size, node failure rate, and link failure rate were investigated for the beamforming application, which was described previously in Section 4.1. The configuration used consisted of one master and 50 slaves, interconnected on a single shared network segment. The default configuration employed a link frame size of 64 bytes and speed of 1.6Mb/s unless otherwise varied. Fig. 10 shows plots of the average number of samples received by the master per sampling period (shown as *Average Quality of Result, QoR* in the figure), as each of the parameters was varied across a range of 10 configurations. For each configuration, the data point plotted is the average number of samples received before the battery level falls below the usability threshold. The data points were obtained by simulating the entire system using the previously described simulation infrastructure, configured with the specified interconnection link speeds, failure rates, etc., with the compiled beamforming master and slave binaries simulated in the different nodes in the network.

The performance of the beamforming application (average QoR) is seen to be insensitive to failure probabilities in

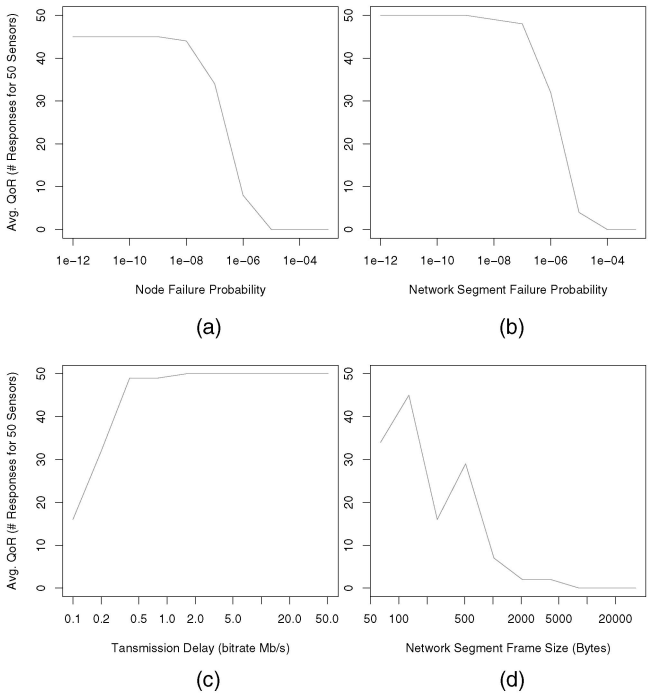


Fig. 10. Effect of transmission delay, link frame size, node and link failure rates on baseline performance. The plots show the throughput of the beamforming application per *round* in terms of the number of samples received by the master node.

nodes and links below 1E-7 per simulation quantum, as shown in Fig. 10a and Fig. 10b. This is due to the fact that, at the sample request rate employed, 100Hz, there is sufficient slack between requests from the master node and, thus, failure rates in the nodes and links can be hidden and do not manifest as an overall degradation in system performance. Above failure probabilities of 1E-7, performance drops off rapidly, with the system being nonfunctional at failure probabilities greater than 1E-4. The system is more sensitive to the link speed and frame size. Faster links are preferable, though larger frame sizes hurt performance as they require longer computational time for transmission and receipt, requiring the copying of larger amounts of data into and out of the modeled hardware transmission and receive buffers, respectively.

6.2.2 Naively Adding Redundancy to Beamforming Application

In order to facilitate application remapping in the presence of failures, such as low battery levels, there must be redundantly deployed (idle) devices to which such remapping may occur. The simple addition of redundant nodes in the presence of applications that can take advantage of redundancy is not sufficient for effective exploitation of redundancy. There are many factors that influence the efficacy of remapping, though they might not otherwise be considered when just looking at application performance.

Fig. 11 shows the sample arrival profile for the beamforming application, in the presence of varying degrees of redundancy, for two different configurations. In the first case, for the 1.6Mb/s network, even though this network speed is sufficient for retrieving samples, as shown in Fig. 10a and also seen in the first 60 sample periods of

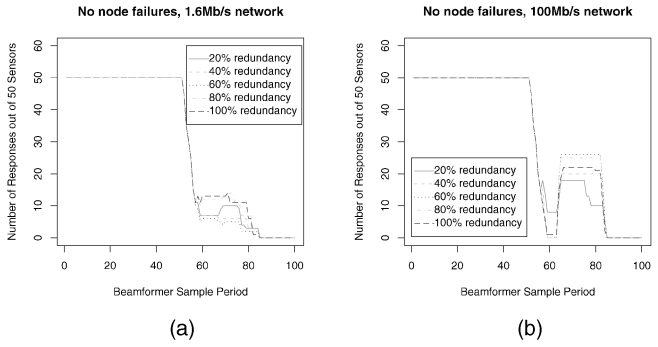


Fig. 11. Effect of adding redundancy to baseline system.

simulation, once nodes begin to attempt to migrate, it severely handicaps the benefits of remapping. Increasing the network link speed to 100Mb/s, as shown in Fig. 11b, provides a substantial increase in the benefits of redundancy (though not improving the performance of the application otherwise).

Observing the trends in recovery of the network versus added redundancy, it can be seen in Fig. 11 (more clearly in Fig. 11b) that increasing the amount of redundancy beyond a point in this case actually leads to a *decrease* in the number of successful remappings. This is due to the fact that, in the beamforming application, increasing the number of redundant nodes increases the opportunity for dying nodes to find a node to migrate to. However, since the network is shared among all active and redundant nodes, this leads to an effective decrease in the available link capacity, reducing the number of successful migrations.

6.2.3 Remote Execution and Code Migration in Fault-Free Network

The effect of adding increasing amounts of redundancy on the total energy consumption of a fabric was investigated in order to compare the overall energy consumption for given levels of redundancy when employing remote execution compared to code migration. A fabric consisting of a baseline of 20 computing elements was employed. The beamforming application was implemented for this substrate, with low battery levels (battery level below 50 percent) at an element triggering an attempt to transfer the application. For remote execution, each processing element in the fabric was modeled as having an additional 4MB Flash, with power consumption based on the Am29PDS322D Flash memory device from AMD [27].

The experiments were performed by limiting the energy consumption of the system as a whole to 5.94 Joules, corresponding to the theoretical capacity of a 0.5mAh battery with a terminal voltage of 3.3V. The performance of the fabric with different amounts of redundancy was judged by observing the system performance across time, being quantified by the number of samples received by the signal aggregation node in the beamforming application.

Fig. 12 shows the number of samples received per sampling period with time, as the percentage of redundancy is increased from 10 percent to 40 percent, and this redundancy was exploited using remote execution. The actual number of samples received per period and the trend in this number of samples is not of much interest by itself,

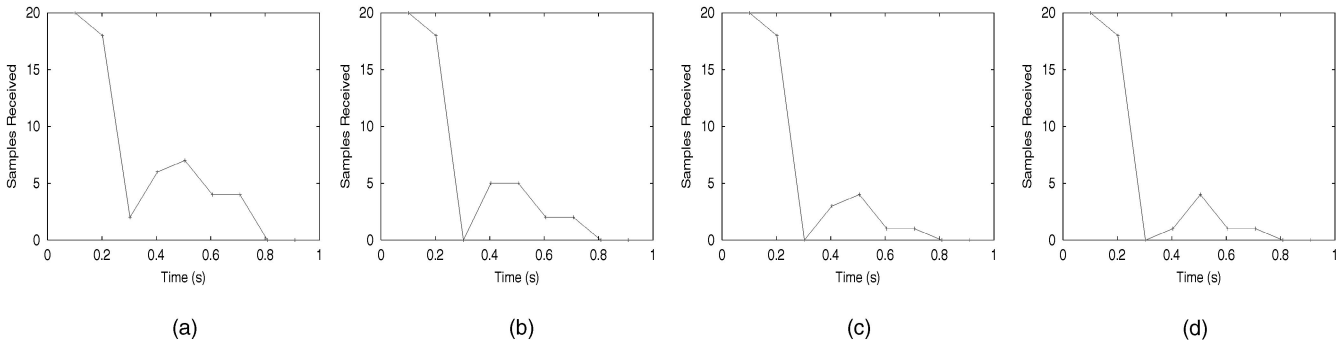


Fig. 12. Employing remote execution in a fault-free network with 10-40 percent redundancy (a network without redundancy would otherwise fail at 0.3s). The plots depict the number of samples received by the master accross time.

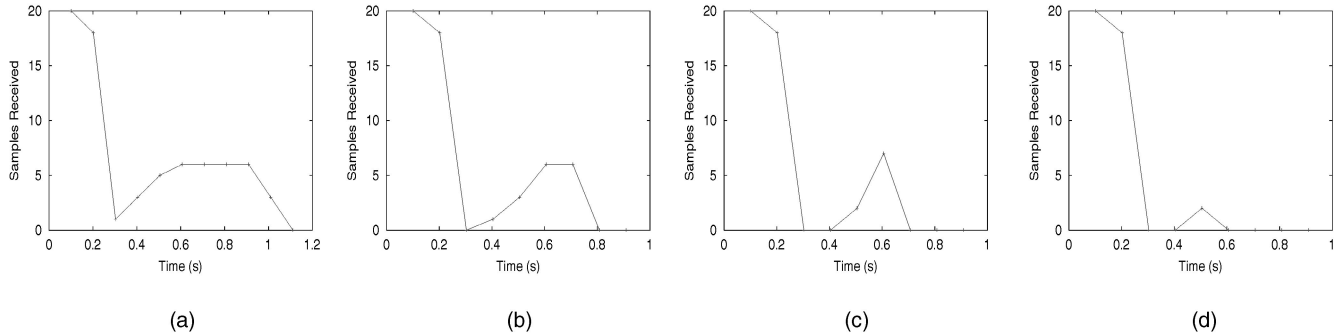


Fig. 13. Employing code migration in a fault-free network with 10-40 percent redundancy. The plots depict the number of samples received by the master across time.

but rather only when compared to the same trend for code migration, shown in Fig. 13.

For small amounts of redundancy (up to 20 percent redundancy), code migration offers a larger sustained sample rate than remote execution. The trade off point in these experiments occurred at 30 percent redundancy, where code migration is able to achieve a larger peak sample rate after low battery levels begin to occur, but, however, is able to extend the system lifetime for a shorter period (up to 0.7 seconds, i.e., the seventh sample period) versus 0.8 seconds for remote execution.

This observed behavior is a result of the added power consumption of the modeled nonvolatile storage for remote execution. For a small number of redundant nodes, this additional power drain constitutes a larger fraction of the overall (computation and communication) power consumption as communication overhead increases rapidly with an increased number of nodes, due to increased contention for the shared medium. Thus, even though the additional power consumed by the Flash per fetched instruction is about one-fifth of the power dissipated in the communication interface for the equivalent amount of time, the processing elements expend a larger fraction of power due to the Flash devices, in the case of remote execution versus code migration, in situations with less than 30 percent redundancy.

For larger amounts of redundancy, above 30 percent, the cost of performing more network activity in code migration exceeds that of the continual power drain from the Flash in remote execution and, thus, code migration performs worse, witnessing a reduction in system lifetime and performance.

It is, however, important to note that code migration might actually provide greater effective redundancy than remote execution in practice. This is because, since code migration does not require elements to have identical replicated copies of code, it is more likely that, for a given fabric, there will be more processing elements that will benefit from code migration since the chances of finding just *any* surrogate processing element will be higher than finding a *specific* one with the necessary nonvolatile storage and identical replicated code.

6.2.4 Remote Execution and Code Migration in Faulty Network

As motivated in Section 1, one reason for building redundancy into e-textiles is to be able to continue functioning in the presence of failures. In the previous section, we investigated the effect of adding redundancy in the presence of *predictable* failures, in the form of low battery levels. A second type of failure that will be common in e-textiles is that of unpredictable intermittent failures, such as those due to electrical faults.

Fig. 14 shows the trends in the number of samples received across time for the case of remote execution, with link failure rates from 1E-8 to 1E-6, as well as the corresponding cases for code migration.

The results in the case of intermittent failures for the lowest failure rate (1E-8 per 17ns) are to be expected from the previous experiments with the fault-free network—for the system with 10 percent redundancy investigated, code migration outperforms remote execution at the low failure rate. As the failure rates are increased, however, the performance of code migration degrades more rapidly than

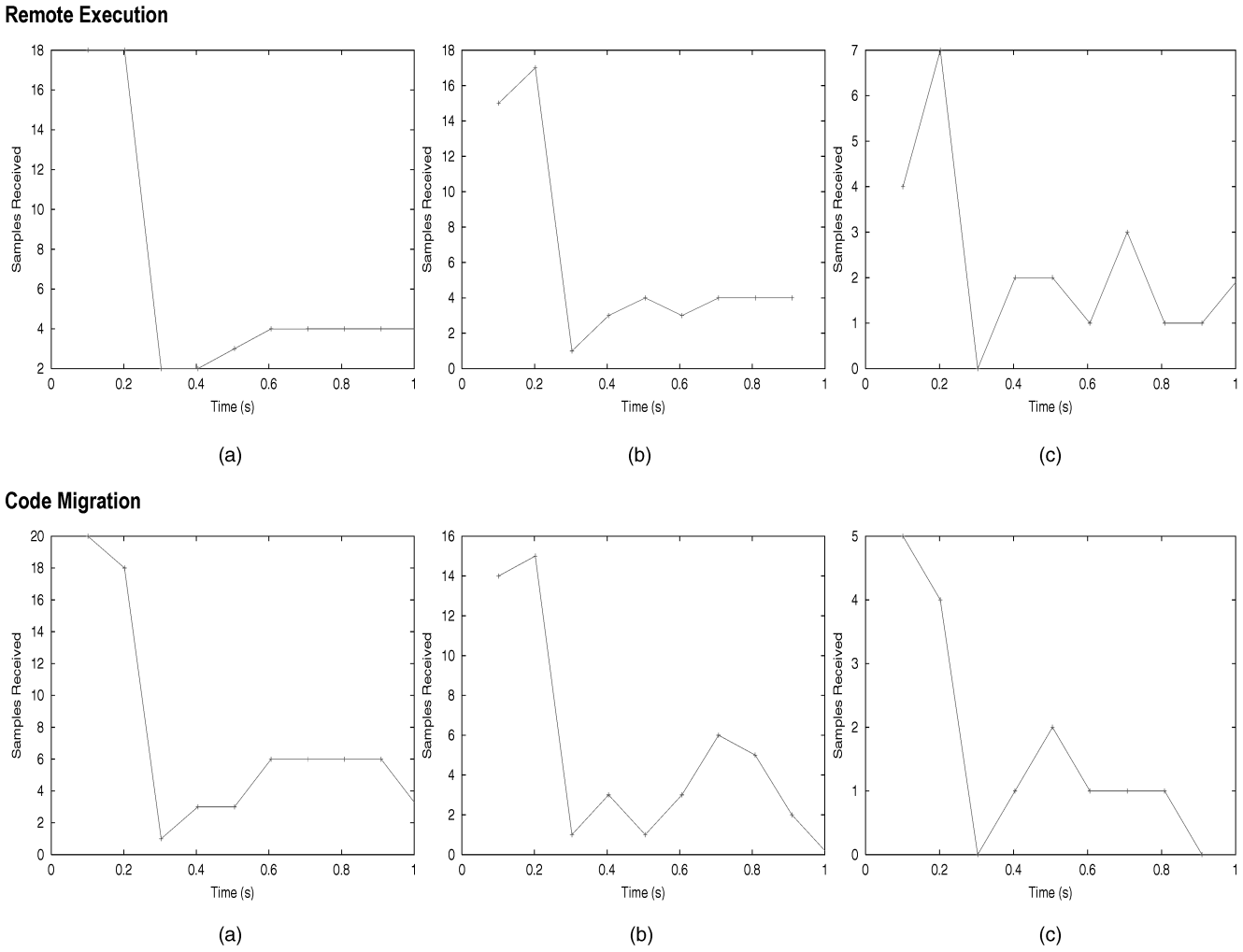


Fig. 14. Effect of variation in the node failure rate in remote execution and code migration, in a faulty network with 10 percent redundancy and interconnect failure rate of 1E-8 per 17ns. (a) Failure rate 1E8, (b) failure rate 1E7, (c) failure rate 1E6.

that of remote execution and remote execution turns out to be superior to code migration for all but the lowest failure rates.

In the application investigated, code migration involves a significantly larger amount of communication (14K bytes versus 648 bytes) and, as such, with intermittent failures in the interconnection links between nodes, the cost of migration is quickly amplified with increasing failure rates as more transmissions fail and must therefore be retried.

6.2.5 Effect of Network Topology on Migration

In a wired network with fixed link throughput capability, one means of increasing the effective link throughput is to partition the network into separate segments. Typically, this would mean adding the requirement of having routers or switches to forward traffic between segments. However, it is possible, in some cases, to partition a network in an application-specific manner.

A factor that leads to unnecessary depletion of energy resources in the redundant nodes is that, in the single bus network, they must awake from sleep mode whenever the master broadcasts a request to all nodes to collect samples. The second factor leading to unnecessary power dissipation is a side effect of fixed throughput on links—the nodes are attached to batteries in groups and, thus, they all experience low battery levels during the same time frame.[2] There can therefore be a significant number of collisions at the link layer as nodes compete for the communication channel during migration. These repeated retries lead to unnecessary power dissipation, both in the dying nodes and in the redundant ones.

One possible partitioning strategy of the network for the beamforming application is shown in Fig. 15. Each node has two network interfaces, one attached to the network segment used for sending back samples to the master and the other attached to a segment dedicated for use during migration. This partitioning takes advantage of the fact that there are two distinct types of communication that occur in the network—normal exchange of samples between the master and slaves (regular, small amounts of traffic) and the migration of applications from active nodes to redundant ones (bursty, large amounts of data).

It is possible to further partition the network as shown in Fig. 16. The link used for communicating between the master and slaves is left as is, but the network for

2. This battery connection topology is assumed since it reduces the cost of wiring power to the individual nodes and is thus likely to be the adopted approach in smart fabrics.

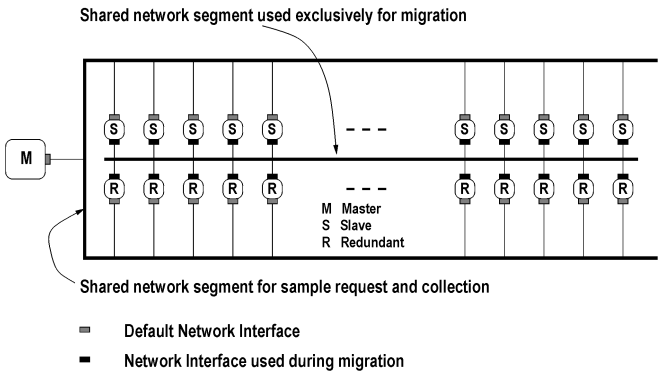


Fig. 15. Baseline topology. A single shared link is employed for the regular exchange of samples and a separate shared link is employed for migration.

communicating between slaves and redundant nodes is subdivided. Since there is no need to communicate between these new partitions (each slave will try to reach a redundant node on its local partition if possible, otherwise, it will give up), there is no need for any forwarding of data between these links. The obvious extreme case of this extension is that which employs direct links between active nodes and redundantly deployed ones.

Fig. 17a and Fig. 17b illustrate the sample arrival profile for an experiment with the network configured as in Fig. 15, with one master, 20 slaves, and 50 percent and 100 percent redundancy, respectively. The gains from partitioning can be seen by comparing to Fig. 17c and Fig. 17d, which illustrate the same application running over the repartitioned network. In the latter case, the network used for migration was partitioned so that each segment was used by five nodes. Partitioning the network permits a larger number of nodes to successfully migrate, as shown by the longer and larger tails on the sample arrival profiles in Fig. 17c and Fig. 17d.

6.2.6 Effect of Node Failures on Migration

The effect of intermittent failures in nodes was investigated to assess the extent to which these failures affect the effectiveness of using migration and the performance of the system. Intermittent failures in the nodes with five failure probabilities per 16ns, of 1E-9 to 1E-5 were simulated. The failure durations were uniformly distributed between 0ms and 16ms. During a failure, a node ceases to execute instructions of the application running over it. Its network interface is likewise decommissioned and any network frames destined for it are lost.

Fig. 18 illustrates the cost of migration for the example application. The beamforming slave is able to migrate even the presence of node failure probabilities as high as 1E-6 per 16ms.

The beamforming application can continue to function at node failure probabilities as high as 1E-5, even though the performance (average sample inter arrival time) is degraded by an order of magnitude.

6.2.7 Effect of Link Failures on Migration

The effect of intermittent failures in the communication link on the efficacy of code migration and application performance was investigated. Five different failure probabilities

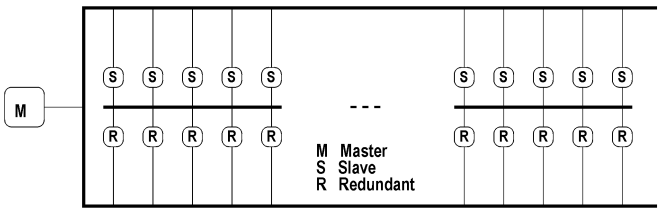


Fig. 16. Segmented topology. Single shared link is employed for the regular exchange of samples and multiple separate links are employed for migration, with five nodes per migration link.

per 16ns of 1E-9, to 1E-5 were investigated, with an average failure duration of 16ms, shown in Fig. 19.

During a link failure, nodes attached to the link continue to execute application code, but will be unable to communicate on the link. Thus, nodes that would usually transmit a frame and return to sleep mode would keep trying to retransmit (sleeping between attempts) until the link recovers from failure. This leads to increased power dissipation in the nodes.

As in the case of node failures, link failures have a larger impact on the effectiveness of migration than on performance. The effect of link failures on migration in the beamforming slave is largely due to there being a large number of nodes that contend for the link. With intermittent failures, more of these nodes will need to retry their transmit requests, effectively reducing the performance of the system.

6.2.8 Effect of Correlated Failures on Migration

The effect of correlated failures was investigated, with correlation coefficients between the links and the nodes of

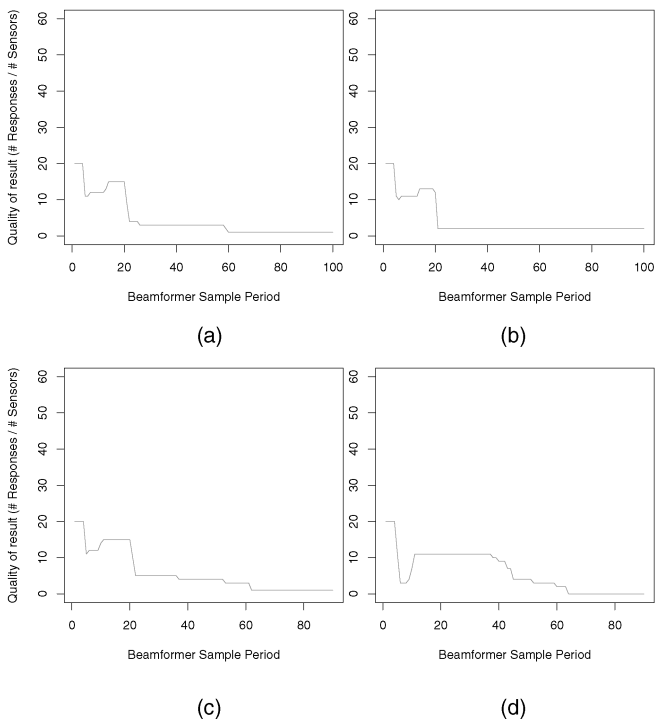


Fig. 17. Effect of segmenting network on efficacy of migration—(a) shared topology, 50 percent redundancy, (b) shared topology, 100 percent redundancy, (c) segmented topology, 50 percent redundancy, and (d) segmented topology, 100 percent redundancy.

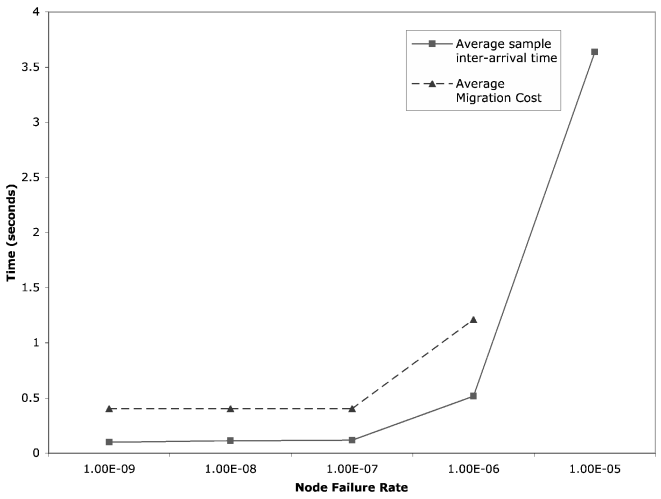


Fig. 18. Effect of intermittent node failures on cost of migration and sample receipt rate for beamforming application.

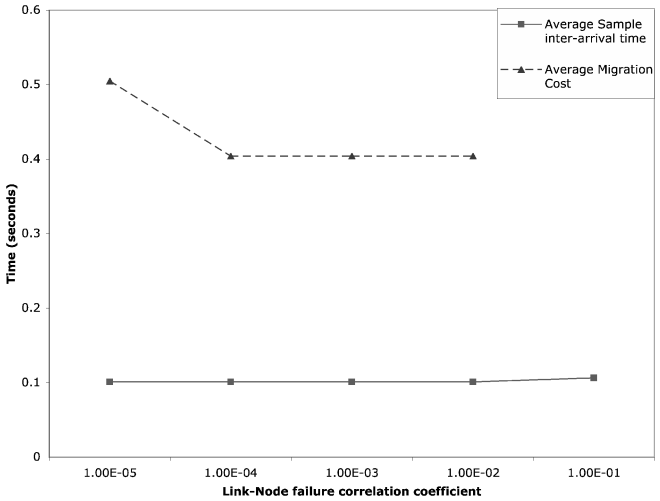


Fig. 20. Effect of correlated link-node failures on cost of migration and sample receipt rate for beamforming application.

1E-5 to 1E-1 and a fixed link failure probability of 1E-8 per 16ns. The result of this investigation is shown in Fig. 20.

In general, the beamforming application was unaffected by increasing correlation between failures in the links and failures in the network. This is partly due to the fact that, with correlated failures, both the node and link fail together with increasing probability as the correlation coefficient is increased. When both the nodes and links fail simultaneously, there is less power wasted as the node is not trying to transmit on a failed link and other nodes will likewise not be trying to reach a failed node.

6.2.9 Effect of Link Speed on Migration

The effect of the speed of the interconnection links on performance and efficacy of migration was investigated.

The beamforming application was able to migrate at link speeds as low as 0.2 Mb/s. The performance of the beamforming application was unaffected by increasing link speed due to the small samples that are the norm in the beamforming application, as implemented in the experimental evaluation. There was no change in performance of the beamforming application in increasing the link speed from 0.2 to 3.2 Mb/s. This behavior is shown in Fig. 21.

Faster network links do not, however, always lead to improved migration cost in the beamforming application. At link speeds of 3.2Mb/s, migration begins to fail in the beamforming application. This is due to the fact that the monitor program on the redundant node responsible for loading the migrating application into memory could not keep up with such high data rates. This is not just an artifact of the application, but rather a manifestation of an important fact—*the processing elements have limited processing capabilities, thus, increasing the speed of the interconnect that links them together will not always provide benefit if the processing elements cannot keep up*.

6.2.10 Effect of Link Frame Size on Migration

It was observed that migration cost in the beamforming application was relatively insensitive to increases in frame size. Fig. 22 illustrates the trends in sample interarrival times and costs of migration for link frame sizes ranging

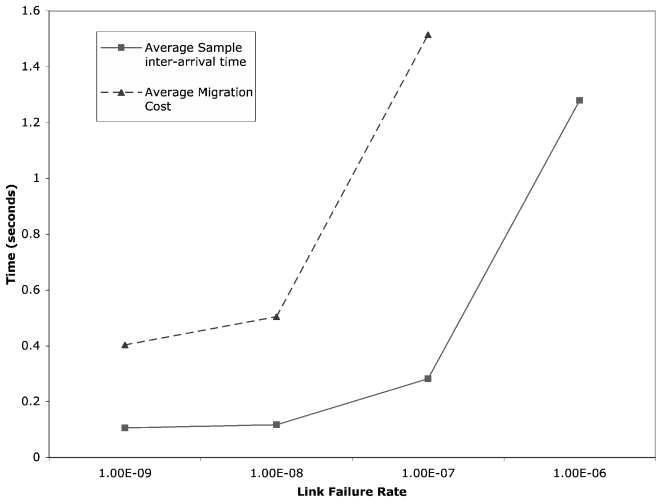


Fig. 19. Effect of intermittent link failures on cost of migration and sample receipt rate for beamforming application.

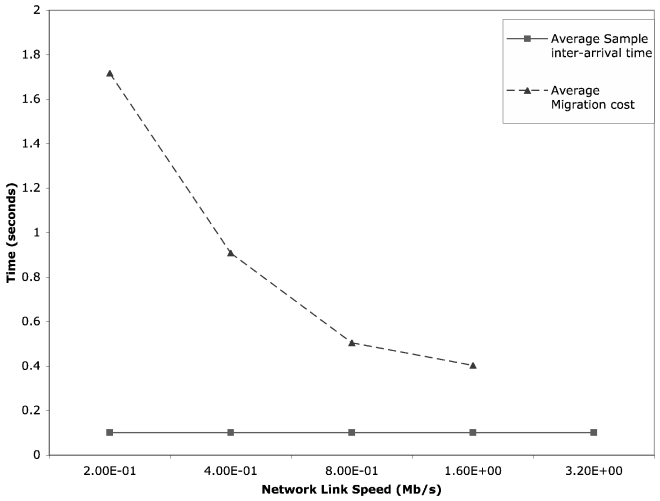


Fig. 21. Effect of link speed on cost of migration and sample receipt rate for beamforming application.

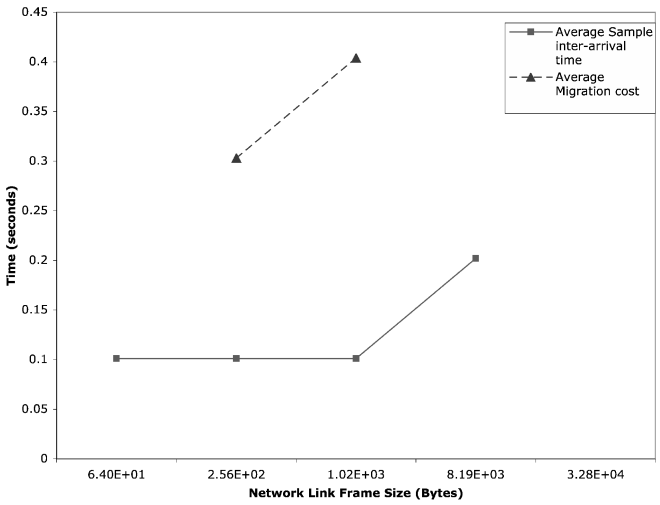


Fig. 22. Effect of link frame size on cost of migration and sample receipt rate for beamforming application.

from 64 bytes to 32 Kbytes. In terms of performance, the beamforming application witnesses no benefits as the frame size is increased from 64 bytes to 1K byte. This is because the samples that are exchanged between the master and slaves during normal operation are only 4 bytes in size. As the frame size is increased further, however, the network link is more often occupied transmitting large frames (with only 4 bytes of useful information). There is a degradation in performance beyond 1 Kbyte frame sizes and the system ceases to function at frame sizes beyond 8K.

The effect of increasing frame size on cost of migration is more subtle. The amount of data to be transmitted during migration is 14 KB for the beamforming slave. Increasing the frame size would therefore ideally be desirable as it would mean fewer frames would have to be transmitted to effect migration. At small frame sizes (64 bytes), migration cannot successfully occur. This is because, at such small frame sizes, the framing overhead (frames have 36 byte headers) is so large that the nodes deplete their energy resources before they are able to successfully migrate. Though not presented here, the overhead incurred in terms of computation, data bytes transmitted, and power consumption for different frame sizes can be analyzed to provide more insight.

At frame sizes above 1K, migration is also no longer successful. This is due to the fact that the time for which the nodes back off before retransmitting is proportional to the frame size, thus the throughput in transmitting large frames is very sensitive to the occurrence of collisions and, once again, the nodes deplete their energy resources before they can successfully complete migration. Furthermore, if the frame size is larger than the amount of application code and data to be migrated, then frames will contain useless data adding as padding bytes and this will reduce the useful throughput on the links.

7 Summary

New technologies often pose new challenges in terms of system architectures, device architectures, and, sometimes, models of computation. The technology of interest in this paper was that of electronic textiles, which are flexible meshes of material containing a large number of unreliable, networked computing elements. The challenges addressed in this paper were those of design methodologies and system architectures, modeling, and fault tolerance for electronic textiles.

The *Model of Colloidal Computing* was introduced as a methodology for harnessing the capabilities of e-textiles, both at the system and at the device level. E-textiles inherently have high defect rates, as well as high fault rates and, thus, must by necessity provide mechanisms for extracting useful work out of the unreliable substrate. By employing a detailed simulation infrastructure designed to enable the simulation of the computation, communication, power consumption, and battery discharge characteristics of e-textiles, the dynamic adaptation of applications in the presence of faults was investigated in order to support the ideas of the proposed computing model. Two techniques to enable adaptation in the presence of faults, *code migration* and *remote execution*, were studied and the effect of various system parameters, such as failure rates, interconnect link speed, and link frame size, were analyzed.

It was shown in the analysis that, in the presence of redundancy in the nodes deployed in a fabric, code migration and remote execution provide feasible means of adapting to failures. For the driver application (beamforming) and the processing element speeds (60MHz) and network speeds (0.2 to 3.2 Mb/s) investigated, it was observed that, for both the regular operation and migration phases of the application, failure rates as high as 1E-7 per 16ns in the nodes and 1E-8 per 16ns in the links could be tolerated.

The two techniques provide a trade off between additional cost of communication versus storage (memory) overhead at the individual devices. A trade off point was observed to exist for systems with 30 percent redundancy —having more redundancy makes it more expensive to perform code migration, thereby making remote execution more favorable.

References

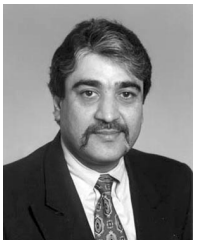
[1] A. Agarwal, "Computational Fabric," *Proc. ASPLOS-IX Wild & Crazy Ideas Session,* Nov. 2000.
[2] E.R. Post and M. Orth, "Smart Fabric, or Wearable Clothing," *Proc. First Int'l Symp. Wearable Computers,* pp. 167-168, Oct. 1997.
[3] E.R. Post, M. Orth, P.R. Russo, and N. Gershenfeld, "E-Broidery: Design and Fabrication of Textile-Based Computing," *IBM Systems J.,* vol. 39, nos. 3/4, pp. 840-860, 2000.
[4] M. Gorlick, "Electric Suspenders: A Fabric Power Bus and Data Network for Wearable Digital Devices," *Proc. Third Int'l Symp. Wearable Computers,* Oct. 1999.
[5] J. Rantanen, T. Karinsalo, M. Mäkinen, P. Talvenmaa, M. Tasanen, and J. Vanhala, "Smart Clothing for the Arctic Environment," *Proc. Fourth Int'l Symp. Wearable Computers,* pp. 15-23, Oct. 2000.
[6] "Georgia Tech Wearable Motherboard (GTWM)," http://www.gtwm.gatech.edu, 2002.
[7] A. Rudenko, P. Reiher, G.J. Popek, and G.H. Kuenning, "The Remote Processing Framework for Portable Computer Power Saving," *Proc. ACM Symp. Applied Computing,* pp. 365-372, Feb. 1999.
[8] U. Kremer, J. Hicks, and J. Rehg, "Compiler-Directed Remote Task Execution for Power Management," *Proc. Workshop Compilers and Operating Systems for Low Power (COLP '00),* Oct. 2000.
[9] D. Milojicic, F. Douglis, Y. Paindaveine, R. Wheeler, and S. Zhou, "Process Migration," *ACM Computing Surveys,* vol. 32, no. 3, pp. 241-299, Sept. 2000.
[10] J. White, *Mobile Agents,* J.M. Bradshaw, ed. MIT Press, 1997.

[11] D. Johansen, R. van Renesse, and F. Schneider, "Operating System Support for Mobile Agents," *in Proc. Fifth IEEE Workshop Hot Topics in Operating Systems,* 1995.
[12] D. Milojicic, W. LaForge, and D. Chauhan, "Mobile Objects and Agents," *Proc. USENIX Conf. Object-Oriented Technologies and Systems,* pp. 1-14, 1998.
[13] A. Cerpa and D. Estrin, "ASCENT: Adaptive Self-Configuring Sensor Networks Topologies," *Proc. 21st Int'l Ann. Joint Conf. IEEE Computer and Comm. Soc.,* June 2002.
[14] Y. Xu, J. Heidemann, and D. Estrin, "Geography-Informed Energy Conservation for Ad Hoc Routing," *Proc. Int'l Conf. Mobile Computing and Networking,* pp. 70-84, July 2001.
[15] R. Marculescu and D. Marculescu, "Does $Q = MC^2$? (On the Relationship between Quality in Electronic Design and Model of Colloidal Computing)," *Proc. IEEE/ACM Int'l Symp. Quality in Electronic Design,* Mar. 2002.
[16] R. Rajagopalan and P.C. Hiemenz, *Principles of Colloid and Surface Chemistry.* Marcel Dekker, 1992.
[17] J. von Neumann, "Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components," *Automata Studies,* pp. 43-98, 1956.
[18] M.D. Beaudry, "Performance-Related Reliability Measures for Computing Systems," *IEEE Trans. Computers,* vol. 27, no. 6, June 1978.
[19] P. Stanley-Marbell and M. Hsiao, "Fast, Flexible, Cycle-Accurate Energy Estimation," *Proc. Int'l Symp. Low Power Electronics and Design,* pp. 141-146, Aug. 2001.
[20] L. Benini, G. Castelli, A. Macii, E. Macii, M. Poncino, and R. Scarsi, "A Discrete-Time Battery Model for High-Level Power Estimation," *Proc. Conf. Design, Automation, and Test in Europe,* pp. 35-39, Jan. 2000.
[21] R.M. Stallman, "Using and Porting GNU CC," Free Software Foundation, 1995.
[22] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, "System Architecture Directions for Networked Sensors," *Proc. Ninth Int'l Conf. Architectural Support for Programming Languages and Operating Systems,* pp. 93-104, Nov. 2000.
[23] B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris, "Span: An Energy-Efficient Coordination Algorithm for Topology Maintenance in Ad Hoc Wireless Networks," *Proc. Seventh ACM Int'l Conf. Mobile Computing and Networking,* pp. 85-96, July 2001.
[24] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensors Networks," *Proc. Sixth Ann. Int'l Conf. Mobile Computing and Networking,* 2000.
[25] Y. Yu, R. Govindan, and D. Estrin, "Geographical and Energy Aware Routing: A Recursive Data Dissemination Protocol for Wireless Sensor Networks," Technical Report UCLA/CSD-TR-01-0023, Computer Science Dept., Univ. of California Los Angeles, May 2001.
[26] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley, "The Design and Implementation of an Intentional Naming System," *Proc. ACM Symp. Operating Systems Principles,* pp. 186-201, 1999.
[27] Advanced Micro Devices, "Am29PDS322D Page-Mode Flash Memory Datasheet," 2001.

Phillip Stanley-Marbell received the BSc degree (1999) and MSc degree (2001) from Rutgers University and is currently a PhD student at Carnegie Mellon University, Pittsburgh, Pennsylvania. His research interests include energy-aware microarchitectures, virtual machines, and fault-tolerant systems. He is the author of a textbook on the Inferno operating system and its Limbo programming language (John Wiley & Sons, 2003). He is a student member of the IEEE and ACM.

Diana Marculescu received the MS degree in computer science from the "Politehnica" University of Bucharest, Romania, in 1991, and the PhD degree in computer engineering from the University of Southern California in 1998. She is currently an assistant professor of electrical and computer engineering at Carnegie Mellon University, Pittsburgh, Pennsylvania. Her research interests include energy aware computing, CAD tools for low power systems, and emerging technologies (such as electronic textiles or ambient intelligent systems). She is the recipient of a US National Science Foundation Faculty Career Award (2000-2004) and a member of the executive board of the ACM Special Interest Group on Design Automation (SIGDA). She is a member of the IEEE and ACM.

Radu Marculescu received the PhD degree from the University of Southern California, Los Angeles, in 1998. In 2000, he joined the Electrical and Computer Engineering Faculty of Carnegie Mellon University, Pittsburgh, Pennsylvania, where he is now an assistant professor. His research interests include system-level design methodologies with emphasis on low-power issues, networks-on-chip, and ambient intelligence. He has recently received two best paper awards from the 2001 edition of the Design and Test Conference in Europe (DATE) and 2003 edition of Asia and South Pacific Design Automation Conference (ASP-DAC). He was also awarded the US National Science Foundation's Career Award in 2000 and Carnegie Institute of Technology's Ladd Research Award 2002-2003. He is a member of the IEEE.



Pradeep K. Khosla received the BTech (Hons) from the Indian Institute of Technology, Kharagpur, India, in 1980, and both the MS (1984) and PhD (1986) degrees from Carnegie-Mellon University, Pittsburgh, Pennsylvania. He is currently the Philip and Marsha Dowd Professor of Engineering and Robotics, head of both Electrical and Computer Engineering Department and Information Networking Institute and founding director of the Center for Computer and Communication Security at Carnegie Mellon. From January 1994 to August 1996, he was on leave from Carnegie Mellon and served as a DARPA program manager in the Software and Intelligent Systems Technology Office (SISTO), Defense Sciences Office (DSO), and Tactical Technology Office (TTO). His research interests are in the areas of internet-enabled distributed and composable simulations, collaborating autonomous systems, agent-based architectures for embedded systems, software composition and reconfigurable software for real-time embedded systems, distributed robotic systems, distributed information systems, and intelligent instruments for biomedical applications. He is a recipient of the Inlaks Foundation Fellowship in 1982, the Carnegie Institute of Technology Ladd award for excellence in research in 1989, two NASA Tech Brief awards (1992, 1993), the ASEE 1999 George Westinghouse Award for Education, Siliconindia Leadership award for Excellence in Academics and Technology in 2000, the W. Wallace McDowell award, and was elected an IEEE fellow in January 1995. He was appointed a distinguished lecturer for the IEEE Robotics and Automation Society for 1998-2002.