

# Dynamic Fault-Tolerance Management in Failure-Prone and Battery-Powered Systems

Phillip Stanley-Marbell, Diana Marculescu  
Department of Electrical and Computer Engineering  
Carnegie Mellon University  
Pittsburgh, PA 15213-3890

{pstanley, dianam}@ece.cmu.edu

## ABSTRACT

Emerging VLSI technologies, as well as emerging platforms, are giving rise to systems with inherently high potential for runtime failure. Such failures range from intermittent electrical and mechanical failures at the system level, to device failures at the chip level. Techniques to provide reliable computation in the presence of failures must do so while maintaining high performance, with an eye toward energy efficiency, and when possible, maximizing battery lifetime in the face of battery discharge non-linearities. This work presents one approach for achieving reliable computation in the face of failure, and presents a set of metrics, for characterizing system behavior in terms of energy efficiency, reliability, computation performance and battery lifetime. The proposed technique for reliable computation in the presence of failures, *Dynamic Fault-Tolerance Management (DFTM)*, relies solely on local decisions to attain global reliable computation. The proposed combined metrics, referred to as *ebformability measures* (since they combine the effects of energy, battery lifetime, performance and reliability), are used to evaluate the efficacy of DFTM. For an example platform employed in a realistic evaluation scenario, it is shown that system configurations with the best performance and lifetime, are not necessarily those with the best combination of performance, reliability, battery lifetime and average power consumption.

## Keywords

Energy-Efficiency, Low Power, Batteries, Performance, Fault-Tolerance, Reliability, Performability.

## 1. INTRODUCTION

New applications of VLSI technology pose many challenges for existing CAD methodologies. The emergence of power consumption as a critical system design constraint, particularly in battery powered computing systems, led to the development of a slew of techniques for *power management*. These efforts have further been bolstered by recent attention to the effect of application profiles on battery lifetimes. Progress in device technologies, coupled with reduction in costs, is enabling new classes of applications, such as wireless and wired sensor networks. Wired sensor networks for example, may take the form of flexible substrates with 10's or 100's of low power microcontrollers embedded per  $m^2$ , with power distribution and communication fibers embedded in the substrate. These emerging technologies pose new CAD challenges in much the same way that the burgeoning portable computing device market caused increased attention to power management techniques and algorithms. A new dimension in requirements, is that of reliability in the presence of runtime failures. Runtime failures may be due, for example, to intermittent electrical failures due to wear and tear in a wired sensor network embedded in a flexible substrate. They may likewise be due to changes in weather conditions in a wireless network sensor deployed in the field. Failures due to the depletion of energy resources (such as batteries), although often predictable, may also occur.

Techniques for enabling reliable computation in the presence

of failure are thus necessary. In order to judge the efficacy of various techniques in an appropriate design methodology framework, metrics which combine energy efficiency, performance, reliability and battery lifetime (taking into consideration nonlinearities in battery and DC-DC converter characteristics), are required.

## 2. CONTRIBUTIONS OF THIS WORK

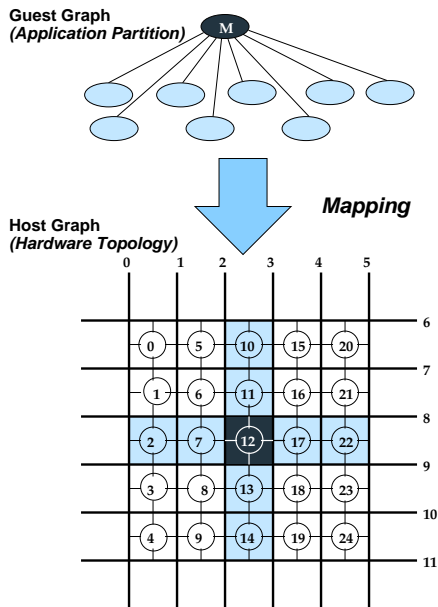
This paper introduces a methodology for dynamically adapting failure-prone battery powered systems to counteract the effects of failures. The proposed technique, *dynamic fault-tolerance management, (DFTM)*, employs only local decisions at devices in a network to achieve the global goal of counteracting the effects of failures. The presented investigation of DFTM employs an offline approach, in which the best system configuration, for the likely prevalent failure conditions, is determined at design time.

In order to determine the efficacy of different configurations of DFTM, we employ results from traditional performance-related reliability measures (generally referred to as *performability measures*) as well as introduce new measures that incorporate energy efficiency, battery discharge effects, performance and reliability, which will henceforth be referred to as the *ebformability measures*. The effectiveness of the proposed DFTM approach, as well as the benefits of employing ebformability measures in a design methodology framework for emerging platforms, is verified through a detailed simulation study. The simulation framework employed, models computation (at the instruction level), communication (at the bit level), runtime failures in both communication and computation, power consumption, and battery discharge effects.

The remainder of the paper begins with a survey of related research in Section 3. A theoretical basis for evaluating dynamic fault-tolerance management is presented in Section 4, followed by a description of the proposed DFTM in Section 5, and a derivation of the ebformability measures in Section 6. Section 7 presents an experimental evaluation of the posited ideas, and the paper concludes with a summary of the key contributions and directions for future research in Section 8.

## 3. RELATED RESEARCH

Particular attention has been paid to average power, peak power, energy consumption, as well as to metrics that combine the above with performance measures, such as the energy  $\times$  delay and energy  $\times$  (delay<sup>2</sup>) metrics [9]. These metrics have enabled the developers of CAD tools to ascertain the relative benefits of algorithms and implementations of hardware, in terms of both performance and power/energy consumption. There have been previous efforts in providing reliable computational substrates out of possibly unreliable components, dating back to von Neumann's seminal work [20]. A significant body of research has addressed combined performance and reliability measures [1], but there hitherto have been no contributions in the area of measures that combine performance, power consumption, battery lifetime and system reliability. Analytic and simulative models for battery life estimation [13, 2] provide means of determining which application workloads will provide longer battery system lifetime, but they neither provide a combined measure of battery life and



**Figure 1: Mapping the *guest graph* to the *host graph*.** Nodes in the *guest graph* (top) represent portions of an application. In the *host graph* (hardware topology, bottom), heavy lines represent shared communication links and circles represent computation nodes. Connections to the communication links are shown with light lines.

performance nor a measure that takes into account battery life, performance *and* reliability.

Unlike efforts aimed at providing guarantees in system performance, employing a central layer of control [11], the proposal of this work, is to provide general fault-tolerance management, *using only identical local policies* at each node in a network. Without a central point of control, a challenge is to provide resilience to faults on the macro scale, from decisions performed at individual nodes.

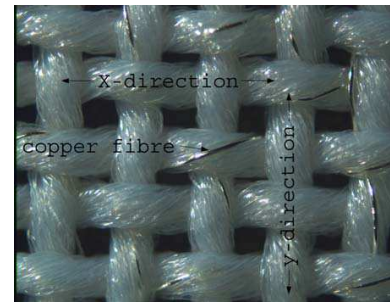
The application domains that stand to benefit greatly from both techniques for reliable computation in the presence of failures, and metrics for judging the efficacy of such techniques, are the emerging technologies of wireless sensor networks [5] as well as wired sensor networks such those embedded into flexible substrates [12].

#### 4. WORST CASE PERFORMANCE LIMITS FROM RE-CONFIGURATION FOR FAULT-TOLERANCE

Prior work in dealing with faulty arrays of computing elements has shown that it is possible to provide an effectively fault-free substrate in the presence of failures [20], and to do so with constant slowdown [4]. The parameters contributing to slowdown in [4] are used here as metrics for reasoning about potential benefits of the proposed DFTM techniques.

Techniques for providing fault tolerance usually employ *redundancy*—in the presence of failures, redundant devices or spares are employed to provide correct system behavior. Invariably, portions of the executing application must be moved from failing devices to redundantly deployed ones. In *gracefully degrading systems*, the redundantly deployed devices might also be employed for computation, to provide better system performance in the absence of failures.

The re-mapping of applications to the underlying hardware substrate to counteract the effect of failures, may be expressed formally as the problem of finding an *embedding* of a fault-free *guest graph* in a faulty *host graph*, following the arguments in [4]. The *guest graph* represents the structure of an application to be executed on a networked system. The nodes in the graph represent components of an application and the links represent communication dependencies between components—there is a link



**Figure 2: Detail of a flexible substrate with embedded communication links from Kirstein *et al*[8].** Conductors can be used for both communication, and for supplying power to processing elements, which can be embedded in the substrate to mimic the topology in Figure 1.

between any two nodes that must directly communicate (e.g. exchange values) at any point during the lifetime of the application. For example, the top half of Figure 1 depicts a *guest graph* for an application which is comprised of 9 units of concurrency, which communicate. In normal operation, the node shaded dark (labeled *M*) communicates with the lightly shaded nodes<sup>1</sup>.

The *host graph* corresponds to the hardware substrate on which the application will be executed. The application must be *mapped* to the hardware substrate, such that the units of concurrency in the application are assigned to hardware components. In mapping the nodes of the *guest graph* to those of the *host graph*, it might be necessary to map multiple nodes from the *guest graph* to a single node in the *host graph*. The maximum number of nodes from the *guest*, that are mapped to a single node in the *host*, for a given embedding, is referred to as the *load*, *l*. The interconnection topology between nodes in the *host graph* limits which nodes may communicate directly with each other. Therefore, for a particular embedding, it may become necessary to map nodes in the *guest graph* to those in the *host graph* such that two nodes that are adjacent in the *guest graph* are no longer so when mapped to the *host graph*. The maximum increase in the distance between two nodes in the *guest graph* when they are mapped to the *host graph*, is referred to as the *dilation*, *d*. A particular embedding might also lead to the need for multiple nodes in the *host graph* to share edges that were previously unshared in the *guest graph*. The increase in the number of nodes sharing a given edge in an embedding is referred to as the *congestion*, *c*.

It has previously been proven [4] that for a given embedding,  $\Pi$ , the *slowdown*, *s*, a measure of the degradation in system performance, is given by:

$$s = \mathcal{O}(c + l \cdot d)$$

In attempting to negate the effects of runtime faults, DFTM will attempt to perform *re-mapping*, moving components of the application across hardware components—in other words, performing a new assignment of the *guest graph* to a *host graph*. In order to maximize its effectiveness, DFTM must therefore attempt to minimize each of *c*, *d* and *l*.

To put these metrics in perspective, the mapping of a *guest graph* to a substrate that consists of computing devices is illustrated in Figure 1. This topology corresponds to an actual physical platform (Figure 2) which incorporates communication conductors as well as power distribution conductors in a woven fabric [8], as a substrate for the attachment of computing devices. Such a hardware substrate may contain large numbers of computing devices (of the order of 100's of processing elements per m<sup>2</sup>), embedded in a flexible medium. The processing elements will typically be physically small 8- or 16-bit microcontrollers, DSPs or small programmable logic devices. Such a hardware substrate

<sup>1</sup>This particular graph is that of the driver application (beamforming for, e.g., active antenna arrays) that will be used in the evaluations in Section 7.

has uses ranging from intelligent building materials in home and office environments, to aerospace and military applications.

The node labeled 12 in Figure 1 has 20 neighbors with which it may communicate directly over one of the shared communication links. Node 12 can communicate with all nodes except nodes 0, 4, 20 and 24, over one of the shared links to which it is connected. The shaded nodes in the host graph (hardware topology) in the lower half of Figure 1, represent an embedding of the guest graph in the host graph. This embedding has a load,  $l = 1$ , a dilation  $d = 1$  and a congestion  $c = 0.3333$ , since the node labeled  $M$  in the guest graph is connected to its neighbors through up to 3 possible paths.

## 5. DYNAMIC FAULT-TOLERANCE MANAGEMENT

In traditional low power and portable computing systems, *dynamic power management* [14] exploits variations in application requirements, to adjust performance and power consumption of a system to application behavior. By employing simple rules and predictions (e.g. if the system has been idle for  $x$  minutes, spin down the disk), traditional power management techniques enable longer lifetime in the presence of workload variations and energy resource constraints. Dynamic fault-tolerance management (DFTM) aims to encompass a broader range of constraints besides power consumption—to take advantage of variations in both *application* and *environment* behavior, enabling maximal application lifetime with a possible tradeoff for performance. Environment behaviors may include not just limited energy resources and battery performance, but also runtime failures, which (although not prevalent in traditional computing systems), will be a key consideration in emerging technologies. In a battery powered networked system that may witness a large number of runtime failures, for example, DFTM must determine actions to be performed to maximize system lifetime.

Unlike in the case of power constraints where decisions are based solely on local resources, failures in individual components are generally addressed by providing an external component—*redundancy*—such that failing components may be superseded by functional ones. Approaches to *manage fault tolerance* as opposed to those that *manage power consumption*, must therefore consider reconfiguration of system resources.

A structured approach to these reconfiguration decisions will be essential for tractability in defining algorithms for fault-tolerance management. In this work, we propose a structuring of policies which provide, in addition to local decisions (exemplified by traditional power management techniques), a framework for policies for system reconfiguration. Policies to be enabled at each node in the system, must enable the minimization of the effects of *load*, *congestion* and *dilation*, minimizing the system slowdown, but without requiring global coordination between devices. Dynamic fault-tolerance management, as proposed in this work, therefore consists of three components:

- **Local Decision Policies (L-Class)** : *Changing the local behavior* of a node in the current embedding, such that the node’s behavior with respect to its neighbors decreases one or all of the congestion,  $c$ , load,  $l$  or dilation,  $d$ . For example, a decision of whether or not to forward data between links to which a node in a network is connected, will affect congestion and dilation.
- **Re-mapping Decision Policies (M-Class)** : *Determining when to change* an embedding,  $\Pi$ , to an alternate embedding,  $\Lambda$ .
- **Re-mapping Destination Policies (D-Class)** : *Finding the appropriate alternate embedding*,  $\Lambda$ . For example, experiencing excessive faults at a node in a network, may necessitate transfer of execution to an alternative redundantly deployed device.

**Table 1: A possible set of DFTM Policies. Local policies (L0–L2), re-mapping decision policies (M0–M3) and migration destination policies (D0–D4).**

Policy	Description
L0	Do not forward packets.
L1	Do not accept migrating applications.
L2	Never cache information.
M0	Battery too low : migrate.
M1	Too many node faults : migrate.
M2	Too many collisions : migrate.
M3	To many carrier sense errors : migrate.
D0	Pick a random redundant node to migrate to.
D1	Migrate to redundant node with lowest number of collisions.
D2	Migrate to neighbor with lowest number of carrier-sense errors.
D3	Migrate to neighbor with most energy.
D4	Migrate to redundant node with most direct links to communication target.

In this paper, a specific implementation of the aforementioned classes of policies is presented, targeted at battery powered networked embedded systems, comprising of large numbers of nodes, with a fraction of the nodes being redundantly deployed. In the presence of failures, execution of components of an application may be relocated to these redundant devices. Since multiple policies are defined for each class, with the possibility that multiple policies might be relevant at the same instant (i.e. policies might not necessarily be orthogonal in some settings), it will be necessary, where appropriate, to define priorities for the different policies. Such a priority scheme is not pursued in this work, and is a direction for future research.

To adapt a system to time-varying failure rates and modes, in order to provide *fault-free macroscopic behavior* from a *faulty substrate*, it is essential to perform on-line monitoring of failures. For a networked system consisting of battery powered devices, with high probabilities of failures in the interconnection links and devices, the statistics that may be monitored include: (1) *Remaining battery capacity*, (2) *Link carrier sense errors*, (3) *Link collisions* and (4) *Node faults*.

Table 1 provides a set of DFTM policies for a system with the aforementioned failure statistics monitored. Nodes must employ heuristics to ascertain these properties at their neighbors, based on locally measured values. For example, in the case of a device connected to multiple communication links, if one of those links is experiencing a significant number of failures, the device can infer that nodes attached to that same interface will be experiencing similar conditions.

### 5.1 L-Class Policies

The local decision policies or *L-class* policies aim to adapt the execution of applications to prevailing conditions, without performing application re-mapping. The L0 policy, which determines whether or not nodes forward packets, has a direct effect on the performance of the system as a whole, while minimizing work performed locally, and hence extending the local lifetime. If there exists a node in the system whose only communication path is through a node with the L0 policy enabled, for example, due to failures in its other links, then such a node will be effectively disconnected from the network. Thus, rather than greedily enabling local decision policies to minimize consumption of local energy resources, devices must take into consideration the role they play in the system as a whole. Rather than permanently enabling L0 to conserve energy resources, a node in a network may periodically or randomly enable this policy. The L1 policy is relevant to redundantly deployed nodes in a system. A node with L1 enabled will not permit applications to be re-mapped onto it. This can be

desirable if it is more important for the node to use its energy, for example, to forward packets. Finally, the L2 policy determines whether or not a node should cache information. Caching data might reduce the need to communicate in some applications, but can constrain nodes from aggressive power management—e.g. going into a deep sleep mode might lead to loss of such cached data, thus the requirement to cache data might preclude devices from entering a deep-sleep state.

## 5.2 M-Class Policies

In systems which contain redundantly deployed nodes, it is possible to remap execution of applications from nodes witnessing adverse conditions to those experiencing more favorable conditions. The *M-class* policies determine *when* to perform such re-mapping. The M0 policy in this case specifies to attempt to remap an executing application when battery levels fall below a critical threshold. The threshold associated with an M0 policy must be conservative enough to ensure that the re-mapping process completes before energy resources are completely exhausted. The M1, M2 and M3 policies cause application remapping to occur when a threshold in number of faults that have occurred in a node, link collisions and link carrier-sense errors respectively, have exceeded their associated, specified threshold.

## 5.3 D-Class Policies

The remapping destination policy strives to determine a node that an application should be re-mapped to. The D1, D2 and D4 policies are well suited to situations in which links in a system fail and it is desirable that applications adapt around these failures. The D3 policy is relevant to all systems with limited battery resources.

The first step in this investigation of DFTM is to employ an offline approach, in which the best set of policies for the likely prevalent conditions are determined for a system at design time. An online approach in which the policies to be activated are themselves determined by some other *meta-policies* is a challenging area of future research.

Before discussing the experimental evaluation of a system with a subset of the above policies implemented, new measures which enable a combined evaluation of performance, power consumption, reliability and the effect of the application power consumption profile on battery life are described next.

## 6. ENERGY- AND BATTERY-CONSCIOUS PERFORMANCE AND RELIABILITY MEASURES

For fault-tolerant systems in which it is possible to trade off performance for reliability (gracefully degrading systems [3]), it has been necessary to employ measures that combine both system performance and reliability [1] to determine the benefits of different designs. For systems in which performance, power<sup>2</sup> and battery lifetime<sup>3</sup> may be traded off for reliability, similar measures are required to ascertain the usefulness of CAD methodologies for managing fault-tolerance in addition to energy, for individual components or an entire system. This section proposes such measures, which will be used subsequently to evaluate the performance of the previously proposed DFTM methodology.

*Assumptions:* It is assumed that failures in the system under study are exponentially distributed, i.e., the probability of failure of a component is independent of its past histories of failures. This is a reasonable assumption for intermittent electrical failures. The assumption enables the use of Markovian analysis to derive expressions for the failure probabilities over time. For mechanical failures, or those induced by the aging of a battery subsystem, such an assumption will not hold, and alternative means of *deriving* the expressions must be employed—the applicability and

meaning of the derived measures will however not change. In the following,  $F$  is the set of failure states, a subset of the states in the Markov model, which are indexed with the variable  $i$ . The initial system state is always denoted by  $I$ .

The behavior of the networked embedded application under study can be characterized as consisting of a collection of distinct states in a Markov model, each state corresponding to a given level of performance. For example, in a gracefully degrading system with  $N$  nodes, 3 or more of which must be functioning in order for the system to be considered “alive”, the system may be modeled as a set of  $N + 1$  states,  $0..N$ , of which three states,  $0, 1, 2$ , are the set of failing states,  $F$ .

Based on data obtained from observing the system, or from simulation, the transition probabilities between states in the Markov model may then be used to obtain the transition probabilities between states after  $n$  steps (for a discrete time Markov chain) or over time (for a continuous time Markov chain). For a discrete time Markov chain, the steady-state probabilities are given by the *Chapman-Kolmogorov* equation [18]:

$$P^{(n)} = P^{(l)} P^{(n-l)}, \quad \text{for } 0 < l < n. \quad (1)$$

where  $P$  is the matrix of the one-step transition probabilities. The probability of being in a given state after  $n$  steps, can be solved for using either direct or iterative methods such as the *power method* [18], using the initial conditions for state probabilities,  $P_I(0) = 1$  and  $P_i(0) = 0$ , for some initial state  $I \neq i$ . For example, in the experimental evaluation of Section 7, the initial conditions employed will be  $P_8(0) = 1$  and  $P_i(0) = 0, \forall i \neq 8$ . The variation of probability of being in a given state as the system evolves, can now be used to determine the measures of interest.

In extending traditional reliability measures<sup>4</sup> to include performance in gracefully degrading systems, prior work [1] performed a transformation from the time domain to the computation domain, to obtain a *computation availability*,  $T$ , as shown below. For the inclusion of the effect of power consumption, as proposed herein, a further transformation to the time-power domain is necessary, to obtain the *computation availability per Watt*,  $T_{pw}$ :

$$T = \alpha \cdot n \quad (2)$$

$$T_{pw} = \alpha_{pw} \cdot n = \frac{\alpha}{\text{Avg. Power Consumption}} \cdot n \quad (3)$$

where  $\alpha$  is the *computation capacity*, the amount of useful computation per unit of time performed in a given state and  $n$  is the number of time steps. Similarly,  $\alpha_{pw}$  is the amount of useful computation per Watt of power dissipated in a given state. The quotient of performance and power is employed in  $\alpha_{pw}$  rather than the product (as in the case of, say, the energy-delay product), because for congruence with  $\alpha$ , it is desired for larger values of  $\alpha_{pw}$  to be better.

$C_i(T)$ , the capacity function, is the probability that the system executes a task of length  $T$  before its first failure, given that the state at the start of computation was  $i$ :

$$C_i(T) = \sum_{j \notin F} P_j^*(T) \quad (4)$$

$P_j^*(T)$  (or  $P_j(n)$  expressed in terms of  $T = \alpha \cdot n$ ) is the probability of being in a given state after  $T = \alpha \cdot n$  amount of computation. Naturally, larger values of  $C_i(T)$  are desirable for a given  $T$ .

$C_i(T)$  does not take into account the limitation on lifetime imposed by an energy source. It is therefore applicable to systems that either are not battery-powered, or in which an infinite supply of redundant devices with fresh batteries are available, for re-mapping. Even in systems which are not battery powered, the power consumption is still of interest, since it dictates, for example, the cost of cooling and indirectly affects the reliability of the

<sup>2</sup> Average and peak power, and overall energy consumption.

<sup>3</sup> Battery lifetime is a non-linear function of the variation in power consumption over time.

<sup>4</sup> A traditional reliability metric, the *availability* [1], is given by the limit of the sum of the steady state probabilities of being in a non-fail state. The *Mean Time To Failure (MTTF)*, is equivalent to the availability with absorbing failure states.

system. The capacity function per Watt,  $Cpw_i(Tpw)$  is given as:

$$Cpw_i(Tpw) = \sum_{j \notin F} P_j^*(Tpw), \quad (5)$$

where

$$P_j^*(Tpw) = P_j(n) \quad \text{with} \quad Tpw = \alpha_{pw} \cdot n$$

In a battery powered system, the variation of  $C_i(T)$  will be affected by the battery state of charge profile, and will be bounded by the battery life. The *battery-aware capacity function*,  $Cbatt_i(T)$ , is defined as:

$$Cbatt_i(T) = \sum_{j \notin F} P_j^*(T) \cdot \zeta_{batt}(T) \quad (6)$$

where  $\zeta_{batt}(T)$  is the normalized variation of the state of charge of the battery with the amount of computation. It is obtained by transforming the variation of the battery state of charge versus time curve, (which can be derived from the data-sheet for a particular battery cell), into the computation domain. In addition to the battery-aware capacity function, it might be important in a battery-powered system to also consider a *battery-aware capacity per Watt*,  $Cbattpw_i(Tpw)$ :

$$Cbattpw_i(Tpw) = \sum_{j \notin F} P_j^*(Tpw) \cdot \zeta_{batt}(Tpw) \quad (7)$$

$C_i(T)$ ,  $Cbatt_i(T)$ ,  $Cpw_i(Tpw)$  and  $Cbattpw_i(Tpw)$  are used to calculate the *Mean Computation Before Failure (MCBF)*, *Mean Computation Before Battery Failure (MCBBF)*, *Mean Computation per Watt Before Failure (MCPWBF)* and *Mean Computation per Watt Before Battery Failure (MCPWBFB)* respectively:

$$MCBF = \sum_0^{\infty} C_I(T), \quad (8)$$

$$MCBBF = \sum_0^{\infty} C_I(T) \cdot \zeta_{batt}(T) \quad (9)$$

$$MCPWBF = \sum_0^{\infty} Cpw_I(Tpw), \quad (10)$$

$$MCPWBFB = \sum_0^{\infty} Cpw_I(Tpw) \cdot \zeta_{batt}(Tpw) \quad (11)$$

where  $I$  is the initial system state at  $n = 0$  (and  $T = 0$  or  $Tpw = 0$ ).

The *computation reliability*,  $R^*(n, T)$  is the probability the system executes a task of length  $T$ , given that the system state is  $i$  at time-step  $n$ . Likewise the *computation reliability per Watt* is  $Rpw^*(n, Tpw)$ :

$$R^*(n, T) = \sum_{i \notin F} C_i(T) P_i(n) \quad (12)$$

$$Rpw^*(n, Tpw) = \sum_{i \notin F} Cpw_i(Tpw) P_i(n) \quad (13)$$

Similar definitions can be given for *computation reliability before battery failure* and *computation reliability per Watt before battery failure*. They are omitted here for brevity, since they will not be employed in subsequent evaluations in Section 7. It is desirable for a system to have both its  $R^*(n, T)$  and  $Rpw^*(n, Tpw)$  decline slowly with increasing  $n$ ,  $T$  and  $Tpw$ . In other words, a higher expected reliability with increasing task size (amount of computation) is desirable. Similarly, for a fixed amount of computation, a larger expected reliability with increasing time (for example, due to slower computation) is desirable.

The above measures can be used to determine the efficacy of a system in providing fault-tolerance with the best power consumption and longest battery lifetime, and will be used in the next section to evaluate the proposed DFTM. Instrumental to the process of determining the metrics are the one-step state transition probabilities ( $p_{ij}$  for some  $i$  and  $j$ ). In the following section,

they are determined from a detailed cycle-accurate simulation of the system, for a given set of DFTM policies in effect. The distribution of the probabilities will be different for different tuples of policies. Likewise the power consumption, performance and battery life will vary with different policies, and the above measures enable the determination of the best policy in terms of combined performance, power, reliability and battery life.

## 7. EXPERIMENTAL EVALUATION

### 7.1 Driver Application

DFTM enables individual nodes to adapt to faults, *using* redundant computing resources, by re-mapping executing applications from failing systems to redundantly deployed ones. In this work, the re-mapping of the guest graph to a host graph is achieved by *lightweight code migration*, although other techniques such as *remote execution* may be substituted—the ideas of DFTM are not tied to any particular re-mapping approach.

In what we term *lightweight code migration* [17], in contrast to traditional process migration [10], applications are implemented in a manner in which they can be *asynchronously restarted* while maintaining *persistence* for important state. By placing information that must be persistent across restarts in the initialized and uninitialized data segments of the application, it is possible to maintain state across migration while only transferring the program code, initialized data and uninitialized data segments.

The driver application used in the subsequent analysis of the efficacy of DFTM is *beamforming* [19]. The goal in beamforming is to detect the location of a signal source, and “focus” a set of sensors (e.g. microphones) on this source. In a traditional implementation, sampled 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 sample is processed (*filtered*), and this processing is largely independent of the processing of other samples. In a system with a processing device at the location of each sensor (*slave node*), the application may be partitioned by applying the filter operation at each sensor before sending the samples to a central device (*master node*) for final processing (e.g. summation). Figure 1 previously illustrated the logical and physical organizations, for a wired network of sensors used to perform beamforming.

### 7.2 Platform

The framework used in the evaluation of DFTM is a cycle accurate simulator of computation, communication, power consumption, battery discharge characteristics and node/link failures [15]. The modeling of the battery and DC-DC converter subsystem employ a discrete-time battery model based on [2]. The simulation of instruction execution and associated power consumption is based on [16].

For example, to model the topology shown in Figure 1, 25 processing nodes and 12 communication links are instantiated in the simulator. The operating frequencies and voltages of the processing nodes may be specified, affecting both the performance (time scale) and power consumption during simulation. The instantiated communication links are configured with specific link speeds, link maximum frame sizes, transmit and receive power consumption and other parameters. Each instantiated processor, e.g. node 0 in the lower half of Figure 1, is configured to have 4 network interfaces, and these interfaces are attached to specific network communication links, links 6, 1, 7 and 0 (lower half of Figure 1) in the case of node 0. Both the nodes and the links may be configured with failure probabilities for intermittent failure, as well as maximum failure durations. In the case of Figure 1, the links are used as multiple access communication links, however, they may also be used as direct links between nodes. The simulation of processing clock cycles is synchronized with that of the data transmission, thus the modeling of communication and computation are cycle accurate with respect to each other. A few simulation parameters of relevance are listed in Table 2. When

**Table 2: Relevant parameters employed in experimental evaluation.**

Attribute	Value
Operating Frequency	60 MHz @ 3.3V
Idle Mode	15MHz @ 0.85V
Battery Capacity	0.5 mAh
Battery Parameters	Panasonic CGR18 family
DC-DC Conv. Efficiency	Maxim MAX1653
Communication Power	250mW (RX/TX)
Link Speed	200 Kb/s
Link Maximum Frame Size	1024 bits

not actively performing computation, nodes in the system place themselves into an *idle mode*, to conserve battery resources.

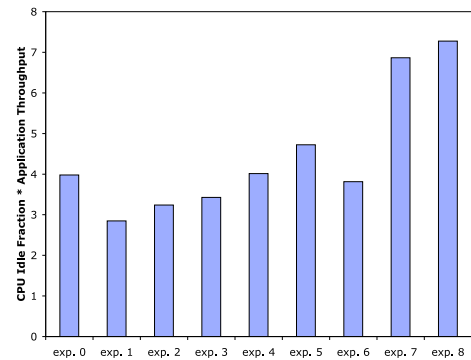
**Table 3: Experiments used to investigate the efficacy of a subset of the proposed DFTM policies. The topology on which the beamforming application executes is that from Figure 1.**

Exp.	Config.	Description
0	No DFTM	No intermittent faults, only low batt. Migrate to a pre-assigned redundant node, when battery levels run low.
1	No DFTM	Intermittent faults in links, rate 1E-8. Migrate to a pre-assigned redundant node, when battery levels run low.
2	No DFTM	Intermittent faults in link #9, rate 1E-6. Migrate to a pre-assigned redundant node, when battery levels run low.
3	(M0, D0)	Intermittent faults in link #9, rate 1E-6. DFTM policy: migrate on low battery to a random redundant node.
4	(M0, D1)	Intermittent faults in link #9, rate 1E-6. DFTM policy: migrate on low battery to redundant node with fewest collisions.
5	(M0, D3)	Intermittent faults in link #9, rate 1E-6. DFTM policy: migrate on low battery to redundant node with most energy.
6	(M0, D4)	Intermittent faults in link #9, rate 1E-6. DFTM policy: migrate on low battery to redundant node with most links to master node.
7	(M2, D1)	Intermittent faults in link #9, rate 1E-6. DFTM policy: migrate on too many collisions, to the redundant neighbor with fewest number of collisions.
8	(M3, D2)	Intermittent faults in link #9, rate 1E-6. DFTM policy: migrate on too many carrier sense errors, to the redundant neighbor with fewest number of carrier-sense errors.

The experiments conducted can be categorized into three groups. The first group (Exp. 0–Exp. 2 in Table 3) of experiments serves as a baseline, and illustrates the performance of the system in the absence of DFTM. The second group (Exp. 3–Exp. 6 in Table 3) investigate the efficacy of the different DFTM D-class policies (i.e. migration destination decisions), for a system with a localized failing link<sup>5</sup> and migration initiated only on low battery levels (i.e. M0 from the M-class of policies). The last grouping (Exp. 7–Exp. 8 in Table 3) investigates the performance of the system with the logical grouping of DFTM M-class and D-class policies, that relate to failing links.

<sup>5</sup>Without loss of generality or applicability of the results, we have chosen to induce failures in link #9.

The failure rates are the failure probabilities per simulation time step of 16ns. When links fail, nodes that attempt to transmit data on the failed link incur a *carrier-sense error*. Such nodes will retry their transmissions after sleeping for a random period. When nodes attempt to transmit on a link which is occupied (another node is in the process of transmitting data), they incur a *collision error*, and similarly sleep for a random period before retrying. In choosing failure rates to provide appreciably adverse conditions for DFTM, a failure rate of 1E-6 for the configured faulty link was employed in all the experiments with DFTM. In previous investigations of the application and simulated hardware, it was observed that a failure rate of 1E-7 was the breakpoint at which the system was not able to hide the additional cost imposed by the failures, under the available slack. Failure rates in the range 1E-6 to 1E-8 are similar to those of first generation networking and computer hardware [6, 7], and seem reasonable choices therefore for emerging hardware technologies in failure-prone environments.



**Figure 3: Variation of  $\alpha$  (product of CPU idle-time fraction and application average sample throughput) with DFTM policy.**

### 7.3 Effect of DFTM Policy on Performance

The CPU occupancy is a measure of the attainable system *load*,  $l$ , as defined in Section 4, and is indicative of the possibility of mapping multiple applications to a single processing element, given the requisite system software support. The product of the *idleness* ( $1 - \text{CPU occupancy}$ ), and the application throughput (average number of samples per round in beamforming application) is what is used as the computation capacity,  $\alpha$ , defined in Section 6. Larger values of  $\alpha$  indicate better combined performance and efficiency in using computation resources.

Figure 3 shows the variation in  $\alpha$  across experiments. The DFTM policies that exhibit the best computation capacity,  $\alpha$ , are the (M0, D3), (M2, D1) and (M3, D2) policies, in order of increasing performance. From Table 3, these three policies aim to maximize available energy resources and minimize communication errors. Thus, the result is to be expected—in the presence of limited energy resources and faulty communication links, they provide the most efficient solutions<sup>6</sup>.

The variation in overall system lifetime across different DFTM policy tuples is shown in Figure 4. The system which witnesses the longest lifetime is Exp. 5, the (M0, D3) policy tuple, which aims to maximize available energy resources, with a 13.7% improvement over the baseline (Exp. 2). Comparing the lifetime trends to those for computation capacity in Figure 3, it is immediately apparent that the system configuration with the longest lifetime is not the most computationally efficient. The results for the computation capacity per Watt (not shown) also indicate it does not exhibit the best combined computation and power consumption. Both of these results however, neither provide a measure of which set of policies provides better reliability in the limit, nor do

<sup>6</sup>The computation capacity per Watt,  $\alpha_{pw}$ , not plotted here for brevity, witnesses an identical trend. In some systems however, it may be that the trends for  $\alpha$  and  $\alpha_{pw}$  might differ, providing a different tradeoff.

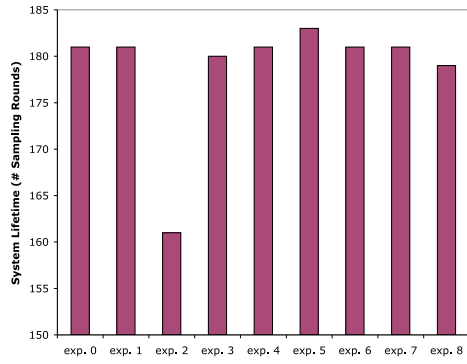


Figure 4: Effect of DFTM policy on system lifetime.

they provide a measure of the combined reliability, power consumption and battery life.

#### 7.4 Reliability, and Mean Computation

The previous set of results evaluated the performance of the various DFTM policy tuples in terms of traditional measures of performance. The policy tuple leading to the longest system lifetime (M0, D3) may not indeed provide the best performance (from Figure 3, this was attained by Exp. 8, the (M3, D2) policy). However, neither of these pieces of information provide any insight into which system is more reliable during its lifetime, and which system has the best combination of performance, battery lifetime and reliability. The measures derived in Section 6 however make it possible to reach such conclusions with a combination of constraints.

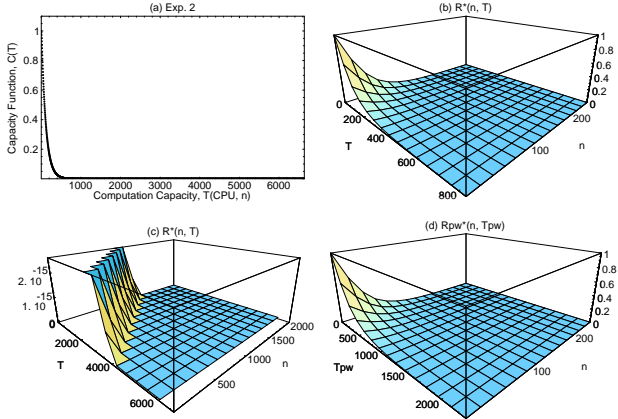


Figure 5: Variation of Capacity  $C_i(T)$ , Reliability  $R^*(n, T)$  and Power-Aware Reliability Function  $R_{pw}^*(n, T_{pw})$  with task length,  $T$ , and task length per Watt,  $T_{pw}$ , for baseline system without DFTM (Exp. 2).

Figure 5 illustrates the variation of the capacity,  $C_i(T)$ , reliability,  $R^*(n, T)$  and reliability per Watt,  $R_{pw}^*(n, T_{pw})$  functions (as described in Section 6) with time and computation performed, for the baseline system without DFTM. Figure 5(a) shows the variation of the capacity function, the probability that the system executes a task of a given length, starting from its initial conditions. Based on the Markov model constructed from the experimental data for that system configuration, the probability of the system executing a task of a given length approaches zero as the task length ( $T$ ), approaches 1000 units. The unit of task length, is the product of the average number of samples received, the average fraction of CPU idle time (a measure of how processor efficient a given configuration is) and the corresponding number of time steps.

The reliability of the baseline system, the probability that the system executes a task of a specified length  $T$ , given that the system is in a non-failed state at time step  $n$ , is shown in row of Fig-

ure 5(b). The front-left face of the cube represents equivalent of the capacity function, since the system starts of, in that case, with  $n = 0$ . The rear-left face of the cube likewise gives the amount of computation that can be obtained with increasing time steps, of a task of length zero. Figure 5(c) provides detail on the region of greatest change for  $R^*(T, n)$ , which occurs as its value reaches zero. From Figure 5(c), it can be seen that for  $T > 4000$  or  $n > 1200$ , the system has a reliability of 0.

Figure 5(d) shows the system reliability per Watt.  $T_{pw}$  is the quotient of the task length and the average power consumption. The ability of the system to execute a task of a given length per Watt of power consumed, is limited at 5000 units per Watt of power consumed. In Figure 5 and subsequent figures, the range of  $T_{pw}$  is greater than the range of  $T$ , since the average power consumption of the nodes in the system are all fractions, less than 1 Watt.

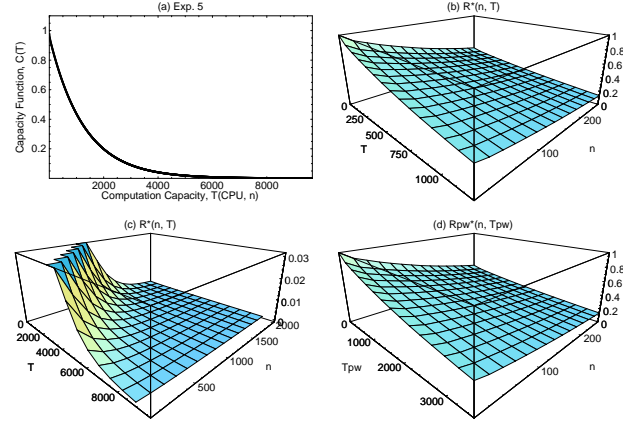


Figure 6: Variation of Capacity  $C_i(T)$ , Reliability  $R^*(n, T)$  and Power-Aware Reliability Function  $R_{pw}^*(n, T_{pw})$  with task length,  $T$ , and task length per Watt,  $T_{pw}$ , for DFTM policy (M0, D3) which aims to maximize battery life.

Equivalent trends for the policy setting which achieves the longest lifetime (Exp. 5. See Figure 4) are shown in Figure 6. Compared to the baseline system without DFTM, this configuration exhibits a much slower decline in the probability of the system being in a non-failure state—thus, it provides a better reliability of executing a task of a given length for increasing task lengths, and for increasing durations of time in which to do so. As can be seen from the detail plots for the reliability ( $R^*(n, T)$ ), the system maintains a non-zero probability of being in a nonfailure state past  $T = 8,000$ , and approaching 30,000 units per Watt of energy dissipated.

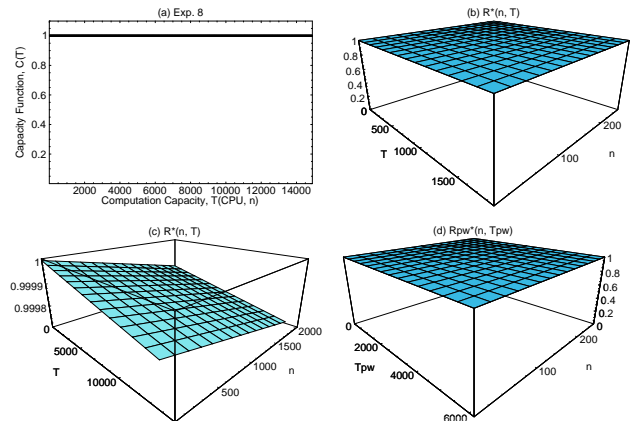
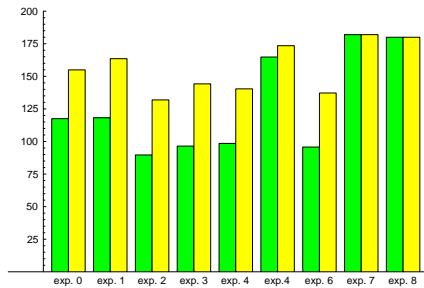


Figure 7: Variation of Capacity  $C_i(T)$ , Reliability  $R^*(n, T)$  and Power-Aware Reliability Function  $R_{pw}^*(n, T_{pw})$  with task length,  $T$ , and task length per Watt,  $T_{pw}$ , for DFTM policy (M3, D2).

The configuration with the greatest lifetime however does not witness the best trend in combined reliability and power consumption. The equivalent trends for the policy configuration (M3, D2), which witnesses the best performance (Exp. 8, see Figure 3), are shown in Figure 7. From the experimental data, the probability of the constructed Markov process being in a non-failure state, remains effectively unity, over time.



**Figure 8: Variation of Mean Computation Before Battery Failure (MCBBF, shown with dark colored bars) and Mean Computation per Watt Before Battery Failure (MCPWBBF, shown with light colored bars) across experiments, with system lifetime limited by a single battery (single re-mapping step).**

The above results might suggest that, for a given goal in computation to be performed per Watt of power consumed, in the presence of failures, the configuration (M3, D2) has a clear advantage over all other policy configurations. The above measures however, although including both the effects of average power consumption, reliability and performance, do not include the effects of the power consumption profile on the battery lifetime. Due to the non-linearities of battery discharge characteristics<sup>7</sup>, a manifestation of the underlying battery electrochemical processes, it is necessary to also consider the effects on a battery system.

The mean computation before battery failure (MCBBF), represents the limiting amount of computation that can be obtained from a particular battery. (i.e. for a given batteries variation in state of charge with discharge, captured by  $\zeta$  defined in Section 6.) The MCBBF for the different system configurations investigated are shown in Figure 8. The surprising result, which could not be reached without considering the effect of the policy on battery discharge, is that, there is little discernible difference between the (M2, D1) policy (i.e. Exp. 7) and the (M3, D2) policy (i.e. Exp. 8). The reason for this is that, although the (M2, D1) policy leads to a better battery lifetime (Figure 4) and has better smaller probability of being in a failure state over time (not shown in the figures), the (M3, D2) policy exhibits better performance (Figure 3).

## 8. SUMMARY AND FUTURE WORK

This paper presented a novel approach for achieving reliable computation in the face of failure. The proposed approach, *Dynamic Fault-Tolerance Management (DFTM)*, is presented in conjunction with a new set of metrics for characterizing *ebformability*, a combination of system energy-efficiency, battery lifetime, performance and reliability. The proposed metrics can be used to assess the quality of various design methodologies and tools for emerging platforms characterized by joint energy, reliability and performance constraints.

Using the proposed techniques, it was shown that techniques providing best performance do not necessarily provide the best combined performance, reliability, power consumption and battery life. For battery powered devices, inclusion of battery discharge characteristics into the model enables better judgment as to the potential computation that may be performed by a system in the presence of runtime failures, before it reaches an absorbing failure state.

<sup>7</sup>DC-DC converters, which are needed to stabilize the output voltage of battery cells also exhibit non-linearities in efficiency across different current draw profiles.

The first step in this investigation of DFTM was to employ an offline approach, in which the best set of policies for the likely prevalent conditions are determined for a system at design time. An online approach in which the policies to be activated are themselves determined by some other *meta-policies* is a challenging area of future research. This paper proposed several classes from which DFTM policies may be defined. Since multiple policies may be defined for each class, with the possibility that multiple policies might be relevant at the same instant (i.e. policies might not necessarily be orthogonal in some settings), it will be necessary, where appropriate, to define priorities for the different policies. Such a priority scheme is not pursued in this work, and is a direction for future research.

## Acknowledgments

This research was supported in part by DARPA Information Processing Technology Office under contract F33615-02-1-4004 and Semiconductor Research Corporation under grant 2002-RJ-1052G.

## 9. REFERENCES

- [1] M. D. Beaudry. Performance-related reliability measures for computing systems. *IEEE Transactions on Computers*, c-27(6):540–547, June 1978.
- [2] L. Benini, G. Castelli, A. Macii, E. Macii, M. Poncino, and R. Scarsi. A discrete-time battery model for high-level power estimation. In *Proceedings of the conference on Design, automation and test in Europe, DATE'00*, pages 35–39, January 2000.
- [3] B. R. Borgerson and R. F. Freitas. A reliability model for gracefully degrading and standby-sparing systems. *IEEE Transactions on Computers*, c-24:517–525, May 1975.
- [4] R. J. Cole, B. M. Maggs, and R. K. Sitaraman. Reconfiguring arrays with faults part I: worst-case faults. *SIAM Journal on Computing*, 26(6):1581–1611, December 1997.
- [5] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar. Next Century Challenges: Scalable Coordination in Sensor Networks. In *Proceedings of the 5th annual International Conference on Mobile Computing and Networking*, pages 263–270, 1999.
- [6] F. E. Heart, R. E. Kahn, S. M. Ornstein, W. R. Crowther, and D. C. Wadia. The Interface Message Processor for the ARPA Network. *Proceedings of the 1972 SJCC, AFIPS Conference Proceedings*, 40:551–567, 1972.
- [7] F. E. Heart, S. M. Ornstein, W. R. Crowther, and W. B. Barker. A New Minicomputer/Multiprocessor for the ARPA Network. In *Proceedings of the 1973 NCC, AFIPS Conference Proceedings*, pages 529–537, 1973.
- [8] T. Kirstein, D. Cottet, J. Grzyb, and G. Troster. Textiles for Signal Transmission in Wearables. In *Workshop on Modeling, Analysis and Middleware Support for Electronic Textiles*, October 2002.
- [9] A. J. Martin, M. Nyström, and P. Penzes. ET2: A Metric For Time and Energy Efficiency of Computation. In *Power-Aware Computing*, 2001. R.Melhem and R.Graybill, ed., Kluwer Academic Publishers.
- [10] D. Milojević, F. Douglass, Y. Paindaveine, R. Wheeler, and S. Zhou. Process Migration. *ACM Computing Surveys*, 32(3):241–299, September 2000.
- [11] M. Perillo and W. Heinzelman. Optimal Sensor Management Under Energy and Reliability Constraints. In *Proc. of the IEEE Wireless Communications and Networking Conference*, March 2003.
- [12] E. R. Post, M. Orth, P. R. Russo, and N. Gershenfeld. E-broidery: Design and fabrication of textile-based computing. *IBM Systems Journal*, 39(3&4):840–860, 2000.
- [13] D. Rakhmatov, S. Vruthula, and D. A. Wallach. Battery Lifetime Prediction for Energy-Aware Computing. In *International Symposium on Low Power Electronics and Design, ISLPED'02*, pages 154–159, August 2002.
- [14] T. Simunic, L. Benini, P. W. Glynn, and G. De Micheli. Dynamic power management for portable systems. In *Mobile Computing and Networking*, pages 11–19, 2000.
- [15] P. Stanley-Marbell. Myrmigki Simulator Reference Manual. Technical report, CSSI, Dept. of ECE, Carnegie Mellon, 2003.
- [16] P. Stanley-Marbell and M. Hsiao. Fast, flexible, cycle-accurate energy estimation. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 141–146, August 2001.
- [17] P. Stanley-Marbell and D. Marculescu. Exploiting Redundancy through Code Migration in Networked Embedded Systems. Technical report, CSSI, Dept. of ECE, Carnegie Mellon, April 2002.
- [18] W. J. Stewart. *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press, 1994.
- [19] B. D. Van Veen and K. M. Buckley. Beamforming: a versatile approach to spatial filtering. *IEEE ASSP Magazine*, 5(2):4–24, April 1988.
- [20] J. von Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. *Automata Studies*, pages 43–98, 1956.