Ameliorating thermally accelerated aging with state-based application of fault-tolerance in cyber-physical computers

C.M. Krishna, IEEE Fellow
Department of Electrical and Computer Engineering
University of Massachusetts
Amherst, MA 01003
krishna@ecs.umass.edu

Index Terms: Adaptive fault-tolerance, thermally accelerated aging, cyber-physical systems, computer control, thermal processor management.

Abstract

Cyber-physical systems have become ever more prevalent in recent years. Computer control is increasingly being considered for use in applications which are operationally resource-constrained and are very cost-sensitive, while requiring very high reliability. The primary approach to building ultra-reliable systems is to deploy massive redundancy. However, the constraints just mentioned make it very difficult to use the massive everywhere-redundancy approaches used in such traditional (relatively cost-insensitive) applications as aerospace. In this paper, we address such problems by adaptively adjusting fault-tolerance levels according to the current state of the controlled plant. Such an approach imposes far less thermal stress on the computer, thereby enhancing reliability (and thus requiring a smaller number of line-replaceable units to maintain required levels of reliability over any given period of operation).
1 Introduction

This paper focuses on the interplay between the plant and cyber sides of a cyber-physical system (CPS). We show how the dynamics of the controlled plant (the physical side), the nature of its operating environment, and the capabilities of its actuators result in tradeoffs that can be exploited to manage computational resources (the cyber side) to enhance reliability.

CPSs involve a controlled plant, such as an aircraft, reactor, power plant, telecommunications network, or automobile, being controlled by inputs from a control computer. The scale
and scope of CPSs have widened greatly in recent years. Originally, these were mostly to be found in such niche applications as aerospace, where cost pressures were (relatively) low, and the focus was on providing a highly customized ultra-reliable computational platform using massive redundancy, even if this proved very expensive. More recently, CPS applications have migrated to regions where the need for high reliability is retained, but is now made more difficult to achieve due to an increased sensitivity to cost. For example, an extra $1,000 spent on the computer in an aircraft or spacecraft is not noticeable; on the other hand, even an extra $100 added to the price of a car can have a measurable impact on its success in the marketplace. However, the requirement for high reliability remains.

Reliability in CPS can be defined in a much more focused way than in general-purpose applications. Traditional, generic, measures of reliability and availability \cite{20} give way to measures that evaluate precisely the ability of the cyber system to contribute to the reliable functioning of the application. Two related approaches exist for this. Performability specifies levels of performance on the part of the application (called accomplishment levels); the computer is then rated on the probability that it facilitates the meeting of each of these levels \cite{28}. Cost functions consider the degradation of the quality of control that occurs due to computational delays; because this delay contributes to the feedback delay in the controlled plant, its impact can be precisely quantified using traditional control theory performance functionals \cite{40}.

In this paper, we consider using redundancy sparingly, by making it a function of the current state of the controlled plant. Much, if not most, of the time, the controlled plant is deep within its safe operating region, and even an occasional erroneous control input will not render it unsafe. In such regions, one can do with a lower level of redundancy; in regions closer to the edge of the safe operating region, a higher level of redundancy will be required. We explore the implications of this approach on the design tradeoffs involved. In particular, we show that such an approach ameliorates the thermal stress on the computer, thereby reducing its thermally accelerated aging, and contributing to enhanced reliability. At the same time, sufficient fault-tolerance is retained in reserve for use when required, when the
controlled plant operates too close to the edge of the safe operating region.

We show how, when the controlled plant dynamics are taken into account, knowledge of the current plant state can be used to precisely determine the appropriate level of fault tolerance to be employed. This operation can often result in substantially lowered computational workloads; such a reduction translates into lower thermal and other stresses on the computational devices, which in turn results in an enhanced mean time to device failure. A reduced device failure rate, in turn, can be used to provision the computational system more economically. Modeling the tight connection that exists between computational unit performance and the safety of the controlled plant allows us to more precisely obtain operating points for the computational subsystem. In particular, we show how task dispatch frequencies can be selected more precisely than is currently the case. We also demonstrate the link that exists between the range of control force that can be delivered by the CPS actuators and the required redundancy levels. We illustrate our approach with an example of multiple spacecraft flying in formation.

Fig. 1 illustrates some of the connections between the cyber and plant sides of a CPS. The controlled plant dynamics are known, usually in the form of state-space differential equations. Analysis of the plant and its use context provides a definition of what plant performance parameter values are deemed acceptable. The forcing capability of the actuators and the frequency with which the various actuator inputs are updated (i.e., the actuator driver task dispatch rate) can be combined with the dynamics and domain user analysis to generate plant operational subspaces within the plant state space. This, in turn, leads to the total computational workload, which when combined with the hardware provisioning can be used to evaluate device stress and reliability. All of these things impact the cyber system reliability; the controller output quality and timeliness determine controlled plant performance.

This paper is organized as follows. In Section 2, we outline some prior related work on adaptive fault-tolerance. In Section 3, we define the various subspaces of the state-space in which the controlled plant operates. We follow with Section 4, the case study of formation-
Figure 1: Cyber- and Plant-Side Interactions.
keeping in satellites. In Sections 5, and 6, we discuss the implications for reliability, and for energy consumption, respectively; and use the case study to illustrate some tradeoffs. The paper concludes with a brief discussion in Section 7.

2 Related Prior Work

The idea of using adaptive fault-tolerance, rather than a fixed level, goes back at least two decades.

Kim and Lawrence were among the first to comprehensively outline and study the adaptive fault-tolerance problem [18]. They describe technical issues associated with adapting the delivered fault-tolerance to the needs of the application. Another early description of many key issues and design challenges inherent to adaptive fault-tolerance can be found in [13]. The authors discuss matching the fault-tolerance needs of the application to the capability of the computer. Both the needs of the application and the capability of the computer can change, the latter happening as modules within the computer undergo permanent or transient failure.

Chameleon is a software infrastructure within which fault-tolerance can be dynamically adapted to suit the needs of a set of applications running in a distributed environment [16]. Certain objects, called ARMORS, allow the implementation of the appropriate level of fault-tolerance by instantiating and managing redundant computation and communication.

Due to their structural characteristics, object-oriented architectures lend themselves to adaptive fault-tolerance. The basic aim is generally to provide transparent adaptation services. For example, the approach of Fraga et al. considers supporting adaptive fault tolerance within the framework of the Corba Component Model [11]. There is a fault-tolerance manager which is informed of the reliability requirements of the application; it then adjusts the amount of redundancy as appropriate. Another object-oriented approach is outlined in [41].

Note that the above-mentioned work (and other similar works) concentrate on the computer. By contrast, in this work, we explicitly link the cyber and plant sides of CPS.

The need for effective co-design of the controlled plant and the control task scheduling
has been treated qualitatively in [2]. Related to this work, the Simplex approach aims to explicitly link the state of a controlled plant to the dependability response on the part of the real-time computer [5]. The idea behind Simplex is to use multiple versions of the control software, with the simpler, more basic, version considered intrinsically more reliable, and the more complex version less reliable. The advantage with the complex version is that, when it functions correctly, it produces output that is much closer to optimal than the basic version. As a result, when the controlled plant is in a region of state space that is regarded as marginal, i.e., where reliable functioning of the controller is important to maintain safety, one can sacrifice some control quality for safety, and use the basic version. In all other regions, one uses the complex version.

The Simplex approach, and the approach presented in this paper both use the current state of the controlled plant to decide what the computational activity should be. The main difference is that Simplex focuses on software reliability while the present work accounts for all kinds of failure; furthermore, we discuss the impact of adaptive fault-tolerance on computational workload, and the impact that the workload has on thermally-induced failure rates. Under some cases, running at a lower level of fault tolerance actually reduces failure rates. Such a reduction can be made precise by quantifying the tradeoff between the control loop sampling rate and reliability.

3 Controlled Plant State Subspaces

In this section, we provide a description of the controlled plant operating state-space on which the rest of the paper is based. This generalizes and refines an approach to state subspaces outlined in [22].

The control workload on the processor consists of periodic and aperiodic tasks. For example, based on the input from the pilot, an aircraft control computer will periodically update the settings of the control surfaces, e.g., engine thrust, rudder, ailerons, and elevator. If some alarm is tripped, then one or more appropriate aperiodic task may be released to deal with the situation.
The controlled plant operates within a state-space. For it to function correctly, it must be within some designated subset of that state-space, called the Safe State Space, $S^3$. $S^3$ is defined based on the user’s assessment of what is considered safe. Defining the safe state space is the responsibility of the control engineer, and is outside the scope of the present work; we simply point out that it is part of the specifications for safe control that the real-time control computer has to satisfy. Further, in the rest of this paper, we assume that $S^3$ is a convex set of points, i.e., that the straight line between any two points in $S^3$ is entirely contained within $S^3$.

We emphasize that the safe state space is defined based on what is acceptable to the user. For example, the user may specify a maximum g-force that the payload in an aerospace application will be subjected to, or the maximum pressure in some reactor vessel. There is no guarantee that being in the safe state space at the present moment allows the controller to keep it there indefinitely; that outcome depends on the speed of the controller, the amplitude of the control inputs that can be applied, and the rate at which control inputs are applied.

We start by considering the simplest case, of a single control task with no noise component. We then consider more complicated cases.

### 3.1 Single Control Task, No Noise

This state-space has subspaces, $S_F$, $S_{FS}$, and $S_N$. These subspaces indicate what level of fault-tolerance is sufficient when the application is in that subspace. If the controlled plant is in $S_F$, then fault correction is sufficient; if it is in $S_{FS}$, it is sufficient to only provide fault detection; if it is in $S_N$, then it is sufficient to provide no fault detection at all. When the plant is in $S_N$, it is so deep within its safe state space that it will remain safe even if, over the period of the subsequent control loop period, the actuators apply arbitrarily incorrect outputs\(^1\).

\(^1\)Note that $S_F$ is not necessarily the same as $S^3$. If the plant is in $S^3$, it is safe at the moment. If the plant is in $S_F$, not only is it safe at the present moment, but the application of correct control inputs will keep it within $S^3$ up to some designated horizon, $T_h$. 
$S_N \subseteq S_{FS} \subseteq S_F \subseteq S^3$

Figure 2: Subspaces.

Hence, if a plant is in $S_N$, no fault-tolerance is needed; only one copy of the control software needs to be executed. Even if the software fails in the worst possible way, and commands the worst possible actuator action, then the plant remains safe, and can be recovered by correct operation in later periods. If it is in $S_{FS}$ but not in $S_N$, we could use a duplex of processors (i.e., two processors running in parallel with a comparator verifying their outputs are coincident) with two independent control calculations being compared. If a significant (i.e., outside the range of numerical approximations) mismatch is detected between the two outputs, then an error is declared in the computation, and a zero control input can be applied. Finally, if the state is in $S_F - S_{FS}$, then we need to use fault correction, e.g., a triplex of processors (i.e., three processors running in parallel) with majority voting of their outputs. The use of duplexes and triplexes is a classical approach in fault-tolerant systems [20]. While we focus on these approaches in the case study in this paper, the state-based approach can be coupled with any approach to fault masking and fault correction, and is not in any way limited to the use of duplexes and triplexes. All that is required for an amelioration of the thermally induced age acceleration (and the associated rapid wear-out of the control computer components) is that fault-masking imposes greater overhead than no fault-tolerance, and that fault correction imposes greater overhead still. Fig. 2 illustrates the relationship between the subspaces.

Next comes the question of how to obtain these subspaces. For decoupled task sets consisting of single tasks, obtaining the subspaces is fairly straightforward. We pick a large
number of samples within the safe state space. For each such sample, \( x \), the controlled plant state-space model can then be solved numerically. In particular, we proceed as follows.

- For \( S_F \), verify that, starting from \( x \) and with the delivery of correct control inputs, the plant remains in the safe state space up to a specified horizon, \( T_h \). Then, \( x \in S_F \). That is, \( x_0 \in S_F \) if
  \[
  x(t|x(t_0) = x_0) \in S^3
  \]
  for all \( t_0 \leq t \leq T_h \) provided that correct control inputs are applied.

- For \( S_{FS} \), we pick samples \( x_0 \in S_F \). For each such sample, solve the state-space model numerically assuming that the control input is zero throughout the first control update period, \( T \), and check that the state, throughout that period, is in \( S_F \). That is, \( x_0 \in S_{FS} \) if
  \[
  x(t|x(t_0) = x_0, u(t) = 0) \in S_F
  \]
  for all \( t_0 \leq t \leq t_0 + T \).

- For \( S_N \), we pick samples \( x \in S_F \). For each such \( x \), pick the worst-case incorrect control input, \( u_{WC} \). In most cases (with a single control input, and a regular-shaped safe state space), this is fairly straightforward from a cursory examination of the plant dynamics. Then, apply this input, and solve the plant state equations numerically to obtain the state throughout the task period; if it stays in \( S_F \) through this interval, then \( x_0 \in S_N \). More concisely, \( x_0 \in S_N \) if
  \[
  x(t|x(t_0) = x_0, u(t) = u_{WC}) \in S_F
  \]
  for all \( t_0 \leq t \leq t_0 + T \).

For convenience, we define \( \phi_{FS} = S_{FS} - S_N \), and \( \phi_F = S_F - S_{FS} \). \( \phi_{FS} \) and \( \phi_F \) denote the subspace where fault detection, and fault correction, respectively, are necessary as well as sufficient.
3.2 Multiple Control Tasks, No Noise

This approach can be generalized to plants with multiple control inputs. First, one identifies the \textit{decoupled input sets} for each plant. Control input sets $U$ and $V$ are said to be decoupled if an input value in $U$ is essentially orthogonal to every input value in $V$. For example, in the case study in Section 4, we consider a satellite in geosynchronous orbit. The inputs applied to the satellite in its orbital plane are decoupled from those applied orthogonal to the plane, i.e., the value of a control input along the orbital plane will have no bearing at all on its state in the orthogonal. In such a case, subspaces $\{S^U_N, S^U_{FS}, S^U_F\}$ and $\{S^V_N, S^V_{FS}, S^V_F\}$ can be defined for the control input sets $U$, and $V$, respectively.

To avoid excessive proliferation of subspaces, we simply define $S^U_N$ as being the subspace such that if the plant state $x \in S^U_N$, then \textit{none} of the tasks in $U$ requires any fault-tolerance; if $x \in S^U_{FS}$, then safety is assured if each task in $U$ either produces a correct output or a detected error which results in a zero output; and if $x \in S^U_F$, then it is sufficient for all tasks in $U$ to deliver the correct output. (An alternative definition for $S^U_{FS}$ is to replace zero output by a continuation of the previous control inputs.)

In contrast to the single-input case, obtaining $S^U_N$ is much more complicated. Heuristic approaches can be taken, such as sufficiently dense sampling from the space of possible control inputs (described below); however, a formal proof that no set of inputs exists that can drive the system out of its safe state space is a different matter, an open question for future research.

A sampling approach to determine (to any required level of confidence) that there does not exist any set of inputs that will drive the plant out of its safe state space runs as follows. By varying the magnitude of each input, we can vary the direction of the input vector. The traditional approach to specifying direction in $k$-dimensional space is through specifying direction cosines $\alpha_1, \alpha_2, \cdots, \alpha_k$. Pick $\alpha_1, \alpha_2, \cdots, \alpha_k$ randomly in the range $[0,1]$ under the constraint that $\sum_{i=1}^{k} \alpha_i^2 = 1$. Denote by $u_{Lim}(\alpha_1, \cdots, \alpha_k)$ the maximum magnitude input that can be applied in the direction $(\alpha_1, \cdots, \alpha_k)$. We then check to see if the plant remains in
the safe state space upon application of this input over one control period. The greater the
number of samples, the greater our confidence in our classification of this point as being in
$S_N$. In particular, suppose $n$ samples are generated, and $\theta_i$ is the vector of direction cosines
associated with sample $i$. Then, we can declare a point $x_0$ to be in $S^U_N$ if

$$x(t|x(t_0) = x_0, u(t) = u^\theta_{i,Lim}) \in S^U_F$$

for all $t \in [t_0, t_0 + T]$ and for all $i = 1, 2, \ldots, n$. Similarly, if

$$x(t|x(t_0) = x_0, u(t) = 0) \in S^U_F$$

for all $t \in [t_0, t_0 + T]$, and for all $i = 1, 2, \ldots, n$, then we say that $x_0 \in S^U_{FS}$.

3.3 Noise

Noise introduces a stochastic component to the controlled plant. It is reasonable to expect a
certain maximum magnitude to the noise, the value of which is based on the application and
its operating environment. Noise is integrated into our framework by assuming that the safe
state space has been provided with a safety margin to allow for the impact of noise on the
system state; there has been a lot of work on controlling plants in the face of noise reported
in the classical control literature [9], [19]. Define a Noise-Safe Space (NSS) such that, if the
controlled plant state can be retained within NSS in the absence of noise, it can be retained
within $S^3$ in the presence of noise to an acceptably high probability, given the constraints on
the control amplitude. As stated before, obtaining the safe state space is the responsibility
of the control engineer, and is outside the scope of the present work.

When analyzing plants subject to significant noise, we can define subspaces based on
NSS; all points in $S^3 - NSS$ (i.e., in the safe state space but not in the noise-safe space)
will conservatively be held to require full fault-tolerance. Similarly, if a plant may be in any
one of several noise levels, a conservative approach would be to use as the NSS the space
generated by the intersection of the actual NSSs for each of these noise levels.
3.4 Key Variables

The dynamics of the controlled plant, in interaction with the operating environment, impose a given computational burden on the real-time computer. The scheduling and other resource management decisions of the computer will determine the nature of the control response; this response will interact with the plant dynamics to affect the state trajectory of the plant. The exact variables which quantify these interactions will depend on the particular application. There are, however, several such variables that will be used with respect to almost any application. These are as follows.

- **Operating Environment**: Ambient temperature and stochastic environmental disturbances commonly modelled as noise.

- **Control System Constraints**: Bounds on the activity of the actuators. For example, the elevator of an aircraft can only move over a given range of angles, and the thrust of a rocket engine is bounded by given constraints.

- **Resource Management Decisions**: Periods of periodic control tasks will determine how often the control is updated. The task allocation and scheduling algorithms determine the task response times. In addition, task response times may be affected by energy- and thermal-management issues, for example, using dynamic voltage scaling for energy management [34], and thermal management algorithms [10], [48].

- **Control Algorithm**: the nature and complexity of the control algorithm determines the quality of the output that is generated, and the computational workload that is generated per control task iteration.

In the rest of this paper, we will discuss some of these tradeoffs as they pertain to adaptive redundancy management.
4 A Case Study of Formation Keeping in Satellites

4.1 Introduction

There are many emerging space applications which require multiple small, inexpensive, and lightweight satellites to fly together in formation, in order to cooperate on some joint task. The control problem associated with formation-keeping has attracted considerable attention in the aerospace control community [4],[17],[35],[37],[46],[47]. Formation-keeping is the control of satellites in such a way that some prescribed configuration is preserved. For example, we may require in a two-satellite configuration that the second satellite differ in its $x$ coordinate from the first by $10,000 \pm 50$ metres, while keeping their $y$ and $z$ coordinates equal to within a tolerance of 50 metres. Similarly, the rate of change of these has been limited to 50 metres/sec as well. (These tolerances are for example purposes only; the actual tolerance used, together with other parameters such as satellite mass and actuator capability, will vary depending on the mission.) As the satellites move, the second will have to make adjustments to maintain this formation. The computational burden consists of processing sensor input to determine the relative positions and in calculating the appropriate control input to be applied.

In this case study, we adopt the model of Kapila, et al. [17], for a set of satellites. In such a system, we have a leader satellite with follower satellites maintaining a certain position with respect to the leader. Each follower satellite periodically checks its position with respect to the leader; how this is done is outside the scope of this paper. We focus here with how the control is managed, not how the sensors are configured and operated.

The state-space equations of a satellite can be written as follows [17]. We have a 6-dimensional state space, $x = [x_1 \ x_2 \ x_3 \ x_4 \ x_5 \ x_6]^T$; $x_1, x_3, x_5$ are the three position coordinates in a Cartesian frame, and $x_{2n} = \dot{x}_{2n-1}$ for $n = 1, 2, 3$. $x_1$ and $x_3$ are Cartesian coordinates in the plane defined by the satellite’s orbit around the Earth; $x_5$ is orthogonal to the orbital plane. The origin of this system is the desired position of the satellite. The mass of the satellite is one unit; for non-unit masses, one simply scales up the control force linearly with
the mass.

$u$ is the control input vector, $[u_1 \ u_2 \ u_3]$, and consists of forces that can be applied along each of three dimensions (corresponding to $x_1, x_3, x_5$, respectively). That is, $u_1, u_2$ are control forces applied within the spacecraft orbital plane; $u_3$ is applied perpendicular to that plane. We are not concerned here with how these forces are applied: there are many actuator options available to a satellite, including thrusters, momentum wheels, and reaction wheels [7]. $\omega$ parameterizes the rotational motion of the satellite in orbit. For a satellite in geosynchronous orbit which rotates in synchrony with the earth, $\omega = 2\pi$ radians per day $\approx 7.2722 \times 10^{-5}$ radians per second.

$$\dot{x}(t) = Ax(t) + Bu(t) \quad (1)$$

$$A = \begin{bmatrix}
0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 2\omega & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 \\
0 & -2\omega & 3\omega^2 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & -\omega^2 & 0
\end{bmatrix} \quad (2)$$

$$B^T = \begin{bmatrix}
0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix} \quad (3)$$

This is an approximate model, predicated on the relative distances between the satellites being much smaller than the orbit radius, which is true for our case study.

Note from the nature of matrix $A$ that the control in the $x_1, x_3$ plane is decoupled from the control in the $x_5$ direction. As a result, we can treat the problem of positioning in the $x_5$ direction as mathematically independent of (i.e., decoupled from) the control problem in the $x_1, x_3$ plane. For a detailed discussion of this model, see [17].

### 4.2 Control Issues

The control envisaged is periodic, and uses the zero-order hold (ZOH) approach [3]. Every period of $T$ seconds, the controller samples the state, calculates the appropriate control
inputs, and applies them to the follower spacecraft; under ZOH, the control is kept constant over the sampling period. The aim is to keep the position error at sampling instants below some given threshold.

The problem of control is to strike an appropriate balance between the position error and the amount of control effort expended. Fuel and propellant must be carefully conserved in a spacecraft; in many cases, the useful lifetime of the spacecraft is limited by the point at which it runs out of fuel or propellant. For concreteness in our example, we have selected the following quadratic cost functional to be minimized over the period of operation, $L$.

$$J(x, u) = \int_0^L \left( \sum_{i=1}^{6} x_i^2(t) + \sum_{j=x,y,z} u_j^2(t) \right) dt \quad (4)$$

This is a fairly standard cost criterion in control theory; the appropriate control signals are obtained by solving a Riccati equation, obtaining feedback matrix $K$ such that the control input vector, $u = -Kx \ [3],[9],[12]$. Note that the value of $K$ will depend on the period with which the control input is updated, i.e., on the value of $T$. (Obviously, this is just an example, and our remarks are not limited to such a performance functional; we could just as well have chosen any other optimization criterion, or any other control algorithm.) To save space, we will concentrate on control using $u_z$; results for control inputs in the other directions are qualitatively similar.

### 4.3 Subspaces without Noise

Fig. 3 shows the subspaces $S_F$ and $S_N$. The shaded area in the figure is $S_F - S_N$, i.e., it is the region where we cannot do without any fault-tolerance. Inside the shaded area is $S_N$, where no fault-tolerance is needed. Outside the shaded area is the region where the plant cannot be controlled to be within the safe state space of $x_5 \in [-50, 50], x_6 \in [-50, 50]$. As $u_{Lim}$, the maximum control force possible, increases, the shaded portion expands. That is, the system remains adequately controllable over a larger portion of its safe state space. However, the unshaded inside portion shrinks, indicating that fault-tolerance can be avoided for only a
smaller percentage of the state-space. The intuition behind this result should be quite clear: greater control force allows the system to be more controllable if that control is correctly deployed; however, it can do greater harm if it is incorrectly applied. Fig. 4 shows the region where fault correction is required. As $u_{Lim}$ increases, the safe state space expands because the system has a greater capability to bring larger deviations under control. Because it is the increased value of control input that allows us to expand the safe state space, it is imperative
that the system apply correct actuator inputs when in this expansion portion. Accordingly, fault correction is required when operating within this portion of the state space.

As the control task period increases, we can expect the impact of incorrect inputs to be greater. As a result, we will need to use fault-tolerance for a greater fraction of the state space. Fig. 5 illustrates this need; when the spacecraft state is in the shaded region, we must use fault-correction.

Fig. 6 shows the fraction of $S^3$ that is occupied by $S_F$ for various values of $u_{Lim}$, and also the fraction of $S_F$ that is occupied by $S_N$ and $S_{FS}$. The curves of $S_F$ for $T = 1$ and $T = 2$ are almost identical, and therefore not shown separately. As $u_{Lim}$ increases, the size of $S_F$ increases because the control capability increases. However, this relationship also makes the system more vulnerable when the control input is wrongly directed; this vulnerability is shown by the rapid fall of the fraction of $S_F$ occupied by $S_N$ with an increase in $u_{Lim}$, and also with an increase in $T$. In calculating workload, we assume that one copy of the task is executed when the plant is in $S_N$, two copies when it is in $\phi_{FS}$, and three copies when it is in $\phi_F$.

### 4.4 Control in the Presence of Noise

Without noise, the control algorithm quickly brings the deviation from the desired inter-satellite spacing down to 0, and there is no further activity needed. However, in reality, control has to be applied indefinitely, owing to stochastic disturbances moving the satellites
To study the impact of noise, it is not enough to carry out a static evaluation of the fraction of the state-space that falls into each of the three subspaces; we have to study the state trajectory of the follower satellite to determine what fraction of each of the subspaces is occupied over time. Due to the complexity inherent to this problem, we use simulation.

In this simulation, we have assumed a magnitude-limited Gaussian noise causing disturbances to both the $x_5$ and $x_6$ state variables. (As explained already, the system’s trajectory in the $x_5, x_6$ dimensions is decoupled from its trajectory in the orbital plane, so we can consider movement along these dimensions in isolation from the others.) We model these noise-derived disturbances as occurring according to a normal distribution with a mean of 0, and standard deviation of $\sigma$, with a magnitude limit of $\pm \sigma$ meters. Disturbances are modeled as impulses every millisecond.

For concreteness, we will assume here that the NSS is given by the region in the $(x_5, x_6)$ space of $[-50, 50] \times [-50, 50]$ for all the cases considered, except where otherwise stated.

Fig. 7 shows the fraction of time over which the spacecraft is in subspaces $S_N$ and $\phi_{FS}$ for $\sigma = 0.2, 0.3, 0.4$. As might be expected, as the noise level increases, the fraction spent in the $S_N$ subspace shrinks.

Fig. 8 shows the workload that is imposed on the system as a function of the control update period, $T$. When $T$ is very small, more iterations need to be run per second; however, because the plant is in $S_N$ more often, the number of copies of the control task that are
Feedback Delay = 0.1s; $u_{Lim} = 10$ Newtons/kg.

Figure 8: Mean Number of Task Copies Executed Per Unit Time.

![Graph of Computational Workload vs. Control Update Period, $T$ (sec)](image)

Figure 9: Impact of $u_{Lim}$.

required is less. As $T$ increases, the frequency with which iterations are executed decreases; however, because the plant is more often in less favourable subspaces, the number of redundant copies of the control task required per iteration tends to increase. The resolution of this tradeoff obviously depends critically on the disturbances due to noise, and also on the maximal control effort possible, $u_{Lim}$. Fig. 9 shows the impact of $u_{Lim}$ on the root mean square error of the satellite position, and the control input that is expended. For a greater amount of noise (i.e., satellite position fluctuation), an increase in the $u_{Lim}$ is used to increase the control input; however, for a smaller amount of noise, greater values of $u_{Lim}$ are not used. As the range of control inputs increases, the positioning error tends to decrease.
5 Reliability Implications

It has long been known that workload affects processor reliability: see, for example, [15], [44] for early work in the field. More recently, a great deal of attention has been paid to thermally-induced failure [24], [43]. There are many such failure causes accelerated by higher temperature, including electromigration [1], [32], [38], [36], dielectric breakdown [6], [31], temperature cycling [14], and negative bias temperature instability [39]. A detailed discussion of these failure modes is outside the scope of this paper; we concentrate here on the implications of adaptive fault-tolerance on the processor temperature, and therefore on reliability. Similarly, other stressors that may depend on the operating environment, such as physical shocks [25, 49], can easily be incorporated into the overall modeling approach.

We express this aging behaviour by means of the age acceleration factor, \( \alpha(T_{\text{abs}}(t)) \), where \( T_{\text{abs}}(t) \) is the absolute processor temperature at time \( t \). The effective age of a circuit whose chronological age is \( \xi \) is given by \( \int_0^\xi \alpha(T_{\text{abs}}(t))dt \); the thermal age acceleration factor (TAAF) over this period is then given by the time-averaged value of \( \alpha(\cdot) \).

For the purposes of this discussion, we will treat the processor as a monolith, and not consider the temperature discrepancies that exist between the various subunits of the processor (functional units, integer and floating-point register files, reorder buffer, issue queues, etc.), and to separately consider lateral, and vertical, heat flows. It is straightforward to extend this analysis to such finer granularity, but that is not the focus of the present paper. It would also be of interest to consider the heat flows from one core to another on the same chip, but that again is outside the scope of this work.

We will adopt the well-known equivalent circuit model for heat flow to relate the processor temperature to the power consumed [21]. We treat the processor as a single node dissipating \( \omega(t) \) watts of power at time \( t \). Its thermal-equivalent capacitance is \( C \), and the thermal-equivalent resistance to heat flowing from the processor to the ambient environment is \( R \). Denote the ambient temperature at time by \( T_{\text{amb}} \). Then, as is well known, the processor
temperature, $T_{\text{abs}}(t)$ follows the differential equation [21]

$$C \frac{dT_{\text{abs}}(t)}{dt} = \omega(t) - \frac{T_{\text{abs}}(t) - T_{\text{amb}}}{R}. \quad (5)$$

Suppose $\omega(t) = w$, a constant. Then, the steady-state temperature (achieved in the limit as $t \to \infty$) is given by $T_{ss}(w) = T_{\text{amb}} + Rw$. Solving the differential equation yields

$$T_{\text{abs}}(t) = T_{ss}(w) + (T_{\text{abs}}(0) - T_{ss}(w))e^{-t/RC}. \quad (6)$$

(To avoid confusion, we stress that $T_{ss}(\cdot)$ is a function of the power, while $T_{\text{abs}}(\cdot)$ is a function of time.)

We can now write the differential equation of the aging function by simple application of the chain rule:

$$\frac{d\alpha(T_{\text{abs}}(t))}{dt} = \alpha'(T_{\text{abs}}(t)) \frac{dT_{\text{abs}}(t)}{dt} = \alpha'(T_{ss}(w)) + [T_{\text{abs}}(0) - T_{ss}(w)]e^{-t/(RC)}$$

$$\left( T_{ss}(w) - T_{\text{abs}}(0) \right) (RC)^{-1} e^{-t/(RC)}. \quad (7)$$

Integration of the above equation yields the aging factor, $\alpha(t)$. Note that this derivation has assumed the power consumption to be constant. If power consumption is variable, then the above equation must be modified appropriately, as a function of time.

Let us now turn to the question of how adapting the level of fault-tolerance affects the reliability. Ideally, we would like a situation in which high-workload periods (the periods where the level of fault-tolerance is high) are surrounded by low-workload periods which are fairly long. Returning to the satellite case study, in particular the control in the $(x_5, x_6)$ state space, Fig. 10 provides an example. Here, we show the probability distribution function, i.e., the cumulative probability distribution of the sojourn time per visit to $S_N$. Fig. 10(a) shows how, as the noise level increases, the sojourn time in $S_N$ drops. Fig. 10(b) shows the impact of the control update period for $\sigma = 0.4$; as this period increases, the sojourn time drops rapidly. We can use TAAF to indicate the impact on reliability. In this example, we
Feedback Delay = 0.1s; $u_{lim} = 10$ Newtons/kg; $T=1s$.

Note that the x-axis ranges are different.

Figure 10: Distribution Function of Sojourn Time per Visit to $S_N$.

Ambient Temperature = 300°K; $E_a = 0.4$ eV/°K.

Idle power consumption = 1 watt.

Figure 11: Thermal Age Acceleration Factors.

Assume that the aging function $\alpha(\cdot)$ is proportional to $\exp(-E_a/(kT_{abs}))$ where $E_a$ is the activation energy, $k$ is Boltzmann’s constant, and $T_{abs}$ is the absolute temperature of the processor. This is the well-known Arrhenius equation, widely used to study temperature-caused acceleration in a number of processes, including semiconductor failure [8], [30], [33], [42]. In this example, we treat the processor as a monolith, with a thermal equivalent.

\[2\] Note that we are NOT limiting this approach to Arrhenius-type aging functions; other aging functions exist for such effects as dielectric breakdown, and these can be substituted instead. The procedure, however, and the general conclusions drawn, will be the same.
resistance of 5, and thermal capacitance of 0.3 units. The computational system consists of a triplex; if the controlled plant is operating in $S_N$, or $\phi_{FS}$, only one, or two, copies of the control task are run, respectively. For running $n < 3$ copies, we select the $n$ coolest processors in the system.

Fig. 11 compares the TAAF of the adaptive fault-tolerance approach with various ambient noise levels against the baseline (traditional, non-adaptive) approach which turns on all redundant copies all the time, regardless of controlled plant state. It also indicates the quality of control provided. Here, the quality of control is expressed in terms of the root mean square error of the satellite state-space position in the $(x_5, x_6)$ subspace. From this figure, we can make the following observations. As $T$ increases (i.e., as the frequency at which the control tasks are run decreases), there is initially a noticeable decrease in the TAAF for the adaptive case. This result is because the average number of copies required remains close to the minimum of 1; most of the time, the system stays in $S_N$. As $T$ increases further, the system stays in $\phi_{FS}$ and $\phi_F$ for an increasing amount of time, and the computational workload increases. As a result, even though we have a decrease in the task dispatch frequency, the increase in the number of task copies required per each dispatch instant approximately makes up for it. What this result means is that, even though we are reducing the dispatch frequency (and paying for it in terms of a reduced quality of control), we gain nothing noticeable in the TAAF. By contrast, the baseline case shows a continuing drop in TAAF as $T$ increases. We show this result for two levels of power consumption: 25 watts, and 50 watts.

The impact of the limit on control effort is shown in Fig. 12. Lower $u_{Lim}$ limits controlla-
bility, especially when environmental noise effects are prominent and need to be controlled; it causes the system to spend more of its time closer to the edge of its allowed state-space, and thus require more redundancy. As $u_{Lim}$ increases, this problem diminishes. Beyond a certain value of $u_{Lim}$, however, the risk to the system from an erroneous control input increases, tending to a need for greater redundancy. Greater redundancy translates to a higher computational burden, which increases processor thermal stress, and ages the processor faster.

6 Energy Implications

The energy consumed depends on the number of redundant copies executed, as well as the execution time. The execution time depends on the chosen voltage level. Dynamic voltage scaling is a very well known approach by which to trade off processor energy consumption against the execution time; for a survey of typical approaches taken in real-time systems, see [45]. The energy consumption depends quadratically on the inverse of the time taken: e.g., if we scale voltage so that the execution time doubles, the energy consumption drops by a factor of four. The speed of the processor affects the system response time, i.e., the feedback delay. As the feedback delay increases, the computational energy decreases while the quality of control decreases and the control effort tends to increase. It is to be expected that the probability of being in $S_N$ will decrease as the feedback delay increases. Thus, the attempt to save energy by reducing voltage and clock frequency can be offset (to a limited extent) by the increased need to execute more redundant copies.

To see this concretely, let us return to the satellite example. Fig. 13 shows the fraction of time spent in the subspace $S_N$ as a function of the feedback delay $\tau$, and the task period $T$, for noise level $\sigma = 0.3$. Similar results for other noise levels are not shown for reasons of space; however, they indicate (as might be expected) that as $\sigma$ increases, the deleterious effects of an increase in feedback delay and in the task period begin earlier. We can see from this result that the incremental impact of feedback delay is not large so long as it is a small fraction of the task period. The impact of task period is dominant. The intuitive reason
Figure 13: Impact of Feedback Delay.

\[ \sigma = 0.3 \]

Figure 14: Computational Energy Consumed as Function of Task Period and Noise Level.

Plots show energy relative to control update period \( T = 0.5 \)s.

Labels indicate value of feedback delay, \( \tau \).

For this result is the zero-order hold policy that is followed in this example. The output is held constant over the entire control update period. Hence, with a feedback delay of \( \tau \), the effective age of the calculation \( t \) seconds into the control update period is \( t + \tau \). Fig. 14 shows the computational energy consumed as a function of the task period. For very low periods, the energy consumed is large because task iterations need to be run at a higher frequency. As the period duration lengthens, the probability of remaining in \( S_N \) drops, but not by much, and so the energy consumption drops. For values of \( T \) beyond a certain point, however, the increase in the required number of redundant task copies more than compensates for the decrease in the task dispatch frequency, and the energy consumption rises again. Energy consumption is not therefore simply inversely proportional to the task dispatch period, which is what it would be if we did not adaptively change the amount of fault-tolerance.
7 Discussion

Our approach consists on subdividing the operating state-space of the controlled plant based on how much fault-tolerance is required. These subspaces are determined by a number of factors: (a) the controlled plant dynamics, as expressed by its state equations, (b) the quality and complexity of the control algorithm used, (c) the rate at which the control tasks are dispatched, (d) the characteristics of the operating environment (e.g., the noise levels), and (e) the range of the control forces that can be deployed. We have shown how these factors interact. For example, as the range of possible control forces increases, the quality of control tends to improve; however, the potential harm done by misdirected control also increases, thereby requiring more redundancy. As the control task update rate increases, the number of task iterations per second increases, but the amount of redundancy per iteration tends to decrease. As the feedback delay increases, the quality of control degrades (slightly), but there is the potential for large energy savings and meaningful improvements in reliability due to lower processor operating temperatures. These tradeoffs illustrate the tight connection that exists between the controlled plant and the control computer, and emphasize the need for co-design of the cyber and plant sides of a cyber-physical system.

Acknowledgements

The author would like to thank the associate editor, the editor and reviewers for their careful reading of the draft manuscript and their constructive suggestions for improvement. This work was partially supported in part by the National Science Foundation under grants CNS-0931035 and CNS-1329831.

References


C.M. Krishna received his PhD in Electrical Engineering from The University of Michigan. Since then, he has been on the faculty of the Department of Electrical and Computer Engineering at the University of Massachusetts, Amherst. His principal areas of interest are fault-tolerance, cyber-physical systems, and computer networks. He has coauthored textbooks on real-time systems and fault-tolerant computing.