Implementing Fault-Tolerant Sensors*

Keith Marzullo

TR 89-997
May 1989

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501

*This work was supported by the Defense Advanced Research Projects Agency (DoD) under ARPA order 6037, Contract N00140-87-C-8904.

Implementing Fault–Tolerant Sensors*

Keith Marzullo†
Cornell University
Department of Computer Science

April 27, 1989

Abstract

One aspect of fault–tolerance in process control programs is the ability to tolerate sensor failure. This paper presents a methodology for transforming a process control program that cannot tolerate sensor failures to one that can. Additionally, a hierarchy of failure models is identified.

Keywords: fault–tolerance, process control systems, real–time distributed systems.

1 Introduction

A *process control program* communicates and synchronizes with a physical process. Typically, the program reads values from the physical process through sensors and writes values through actuators, as shown schematically in figure 1.

*Submitted to the *10th Real-Time Systems Symposium*, Los Angeles, December 1989.

†This work was supported by the Defense Advanced Research Projects Agency (DoD) under ARPA order 6037, Contract N00140-87-C-8904 The views, opinions, and findings contained in this report are those of the authors and should not be construed as an official Department of Defense position, policy, or decision.

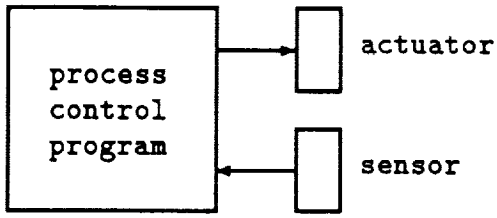

Figure 1: A real–time program

This paper concerns sensor failures. We assume that a programmer writes the program of figure 1 assuming that sensors return accurate values, and provide a methodology for transforming this program to one that tolerates sensors that return inaccurate values.

There are two ways to tolerate sensor failures:

1. Based on the sensor's specification, the control program can determine that a sensor has failed from the value provided by the sensor. For example, if the control program used a sensor to measure the temperature of a reaction vessel and the thermometer read a value too high to be realistic, then the control program would know the thermometer has failed.

2. The sensor can be replicated, either physically or logically, as shown in figure 2. The values of the replicated sensors are compared and a correct sensor value calculated. For example, instead of one thermometer, there could be two thermometers and one pressure gauge on the vessel. From Boyle's law $PV = nRT$, we can calculate three independent temperature readings (or pressure readings, if desired). As long as no more than one sensor fails, the control program can continue to execute correctly. [Sch86a]

These two approaches can either be used independently or together. However, the second approach is better suited for process control programs, because the program always gets an answer in a predictable amount of

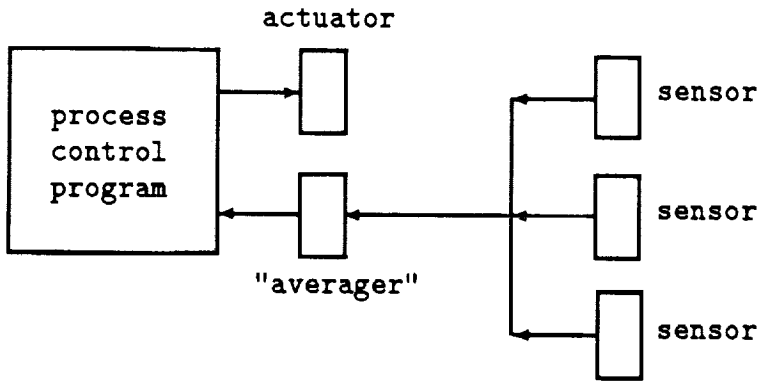

Figure 2: Replicated sensors

time. The first approach can only be used if no deadline will be missed due to the loss of sensor information.

The approach we develop is as follows:

1. Write the process control program with reference to the actual state variables of the physical system. For example, the program controlling the reaction vessel would refer to the temperature T.

2. For each physical variable referenced by the control program through a sensor, replace it with a reference to an *abstract sensor*. An abstract sensor is an interval that contains the physical variable. This step can not be done automatically; the specification of the process control program will have to be changed to refer to abstract sensors.

3. Implement a set of abstract sensors based on a set of *physical sensors*. A physical sensor is a device that "reads" a physical state variable. This step cannot be done automatically, since it may take some knowledge of the physical process to implement abstract sensors.

4. Apply a *fault-tolerant averaging algorithm* to these abstract sensor values in order to calculate derive an abstract sensor that is correct.

> The algorithm assumes that out of n sensors no more than some parameter f of them are incorrect. The relation between n and f (outside of $n > f$) depends on the way sensors can fail. We will assume that abstract sensors fail independently of each other.

This paper is a generalization of the work done by the author and presented in [MO83,Mar84]. This earlier work looked at the problem of *clock synchronization* in a distributed system. A clock is a special kind of sensor, in that the physical process it senses can be expressed simply. In this paper, algorithm 1 and theorem 3 are taken directly from [Mar84]. Section 2 is new, as well as the different failure models discussed in section 3.

The methodology presented in this paper can be thought of as a generalization of the *state machine approach* [Sch86b,Lam84]. A related problem, *inexact agreement*, is presented in [MS85], but the goals are different. Our goal is to dynamically calculate bounds on the accuracy of a sensor value, not to have multiple processors agree on a sensor value. A different approach to agreement among sensors is taken in [Mac84], in which sensor failure is not considered.

In section 2, we define a method of representing sensors that makes them amenable to replication. In section 3, we discuss sensor failure models and present a sensor averaging algorithm. Section 4 contains a demonstration of our methodology.

2 Physical Variables and Sensors

A variable in a computer process is quite different from a state variable in a physical process. A computer variable takes on values from a finite domain, and can assume only a bounded number of values in any finite time period. A physical state variable, however, may take on any real value and can have an unbounded number of values within any non-zero time period. A convenient way to represent physical state variables in computer programs is as functions. The domain of a physical variable is typically *time*, but it can be some other physical variable, depending on the safety properties of

interest[1].

There are three distinct values associated with a physical variable: the physical variable itself, the value returned by a sensor called here a *physical sensor* and the value of a physical variable derived from the value of a sensor called here an *abstract sensor*. In this section, a relation among these values is more precisely defined.

2.1 Physical and Abstract Sensors

A *physical sensor* is a device used by a computer to sample a physical variable. For example, a computer controlling a reaction vessel might have a thermometer as a physical sensor. The computer may obtain values from the sensor either by polling it for its current value or by being asynchronously alerted when a certain value is attained. It is convenient to think of a physical sensor as a mechanism that returns *pairs* of values, such as temperature and time, or position and velocity. By doing so, one can talk of the physical variable as being a function, such as $T(t)$ or $v(x)$, and the physical sensor producing samples $(\hat{T}, \hat{t} : \hat{T} = T(\hat{t}))$ or $(\hat{v}, \hat{x} : \hat{v} = v(\hat{x}))$. Either the sensor device or the controlling computer process may make this association.

A physical sensor is not a very convenient mechanism. For example, with the thermometer attached to the vessel:

- The sensor has a limited accuracy, giving an uncertainty to the temperature. This uncertainty is increased by delays incurred by schedulers and networks.
- The control program may be interested in a temperature at a time the thermometer was not sampled. A value may be interpolated, but to do so requires knowledge of the physical process.
- The sensor may have interesting perceptive properties; for example, it might generate an interrupt if the temperature rises above 100 degrees. This is an important property of the sensor: it allows for an

[1] In the application of section 4, for example, the velocity of a train can be expressed as a function of location.

accurate determination of when 100 degrees is reached. There may be other ways to make the same kind of precise measurement, however. It would be convenient if the control program could be the same for any method of measurement, as long as the measurement is accurate enough.

We define an *abstract sensor* as a piecewise continuous function from a physical variable to a dense interval of physical values[2] We will indicate an abstract sensor as a function such as $\overline{T}(t)$. When possible, we will simply write $\overline{T}$ if we are interested in the "current" value; that is, the sensor value for the current value of t. Control programs will compute with abstract sensors rather than physical sensors. An abstract sensor $\overline{T}$ is *correct* if it always bounds the value of the actual physical value. More precisely,

$$\overline{T} \text{ correct over } D \stackrel{\text{def}}{=} \forall d \in D : \mathbf{min}\ \overline{T}(d) \leq T(d) \leq \mathbf{max}\ \overline{T}(d)$$

Given a physical sensor, it may not be easy to implement an abstract sensor. In general, it may require considerable knowledge about the physical process being monitored. For example, suppose the manufacturer of the thermometer claims it returns a value $\hat{T}$ with an accuracy of ϵ degrees, and the computer can read the sensor's value within δ seconds of the thermometer being sampled. If the time the computer program receives $\hat{T}$ is $\hat{t}$, all we know is that $\exists t_0 : \hat{t} - \delta \leq t_0 < \hat{t} : \hat{T} - \epsilon/2 \leq T(t_0) \leq \hat{T} + \epsilon/2$. This, alone, is not sufficient information to define an abstract sensor $\overline{T}(t)$, since we don't know how to interpolate values between successive sensor readings.

Suppose, however, we know from the physical process being monitored that $|\frac{dT}{dt}| \leq \Delta$. This bound on the change of T allows us to interpolate

[2] An abstract sensor $T(d)$ can be represented as a pair of function $T_{min}(d)$ and $T_{max}(d)$, where $T(d)$ is the interval $[T_{min}(d) .. T_{max}(d)]$. With this representation,

$$\mathbf{min}\ T(d) = T_{min}(d),\ \mathbf{max}\ T(d) = T_{max}(d),\ \text{and}\ |T(d)| = T_{max}(d) - T_{min}(d)$$

intermediate values with a known accuracy. The abstract sensor $\overline{T}(t)$ can be defined as

$$\hat{T} - \epsilon/2 - \Delta(t - \hat{t} + \delta) \leq \overline{T}(t) \leq \hat{T} + \epsilon/2 + \Delta(t - \hat{t} + \delta)$$
$$\text{for } t \geq \hat{t}$$

Unlike physical sensors, abstract sensors can be compared with each other. In section 3, this property will be used to construct a fault-tolerant abstract sensor.

2.2 Abstract Sensors in Specifications

A specification of a process control program may have to be changed when expressed using abstract sensors. It is not possible to take a control program written in terms of physical variables and for each reference to a physical variable substitute a reference to the corresponding abstract sensor. For example, consider a reaction vessel with a pressure relief valve. One safety condition might be that whenever the pressure p is greater than some ceiling p_{max} the valve is open (R); or, $p > p_{max} \equiv R$. The condition $\overline{p} > p_{max} \equiv R$ does not make sense; what does it mean for an interval (p) to be greater than a value?

Let S be a condition on the system and V be the set of physical variables in S that will be accessed through abstract sensors. We need another condition S' that contains no references to any $v_i \in V$ but may instead contain references to $\overline{v_i}$. The only *a priori* constraint we have on S' is that it reduces to S when the abstract sensors have perfect accuracy:

$$(S' \wedge (\forall i : |\overline{v_i}| = 0)) \Rightarrow S^{\forall i: v_i}_{\forall i: \overline{v_i}}$$

There are several ways such an S' can be constructed; for example, we could replace all references to v_i in S with references to the midpoint of $\overline{v_i}$. However, if we assume that all values in $\overline{v_i}$ have the same likelihood of being valid, we have two possibilities. We can either require that *all* points in $\overline{v_i}$ satisfy S or that *there exists at least one* point in $\overline{v_i}$ that satisfies S. More

precisely, for each physical variable v_i the condition S can be generalized as:

$$S' \stackrel{\text{def}}{=} \forall v_i \in \overline{v_i} : S \text{ or}$$
$$S' \stackrel{\text{def}}{=} \exists v_i \in \overline{v_i} : S$$

The generalization of S cannot be done automatically, since it is really a refinement of the problem specification. One approach is to consider the states that are excluded by S, and then expand this set if possible. In the example above, we are probably most interested in avoiding an explosion of the vessel. If so, the condition we want is $(\exists p \in \overline{p} : p > p_{max}) \equiv R$, and we would accept the risk that the pressure valve may open prematurely. As another example, we might want to assert that a catalyst is injected (C) only when the pressure is above a minimum value, or $C \Rightarrow (p > p_{min})$. In this case, the state we are trying to avoid is one where the catalyst is injected at too low a pressure, so we would generalize this condition to $C \Rightarrow (\forall p \in \overline{p} : p > p_{min})$. Note that in this case we admit no states that violate the original condition.

3 Fault–Tolerant Abstract Sensors

Given n independent abstract sensors and a failure model, we would like to construct an abstract sensor that is provably non–faulty. We will first consider the simple failure model of *arbitrary failures*[3]. We will assume a sensor is either *faulty*, in which case it can return any value, or the sensor is correct and always returns a correct value. We assume that no more than f of the n sensors can be faulty.

3.1 Arbitrary Failures

Let $\overline{T}_i$ and $\overline{T}_j$ $(i \neq j)$ be two abstract sensors for the same physical value T. If $\overline{T}_i$ and $\overline{T}_j$ are both correct, then by definition T must be in both

[3]This failure model has also been called *Byzantine failures* or *malicious failures.*

intervals. Put another way, the intervals $\overline{T}_i$ and $\overline{T}_j$ must intersect, and their intersection must contain T.

If we have no more than f faulty sensors, any set of $n-f$ or more mutually intersecting sensors may be correct, since they each share a common value. Conversely, any point not contained in at least $n-f$ intervals cannot be the correct value. Let l be the smallest value contained in at least $n-f$ intervals and h be the largest value contained in at least $n-f$ intervals. The correct value T is then bounded by l and h. This gives us our sensor averaging algorithm.

Algorithm 1 *Fault-tolerant Sensor Averaging*

> *Specification:* Let $\mathcal{S}$ be a set of the values of n abstract sensors of the same physical state variable, read at the same point in their domain (*e.g.* at the same time). Given a maximum number of faulty sensors f, find the smallest interval $\cap_{f,n}(\mathcal{S})$ that contains the correct physical value.
>
> *Implementation:* Let l be the smallest value contained in at least $n-f$ of the intervals in $\mathcal{S}$ and h be the largest value contained in at least $n-f$ of the intervals in $\mathcal{S}$. Let $\cap_{f,n}(\mathcal{S})$ be the interval spanning $l \leq \cap_{f,n}(\mathcal{S}) \leq h$ if l and h exist; otherwise, let $\cap_{f,n}(\mathcal{S})$ be $\emptyset$.

Algorithm 1 is inexpensive - it can be easily implemented in $O(n \log n)$ time, which is a lower bound. One implementation is given in [MO83].

This definition of $\cap_{f,n}(\mathcal{S})$ can contain values that we know cannot be the correct value. For example, figure 3 shows the intersection of three intervals a, b and c. If $f = 1$, the correct value must be within $I1$ or $I2$. By algorithm 1, however, we define the correct sensor value to be I. We do this to preserve the "shape" of the sensor as seen by the control program. Our program is written for an abstract sensor, which is a single interval, and the interval defined by algorithm 1 is the smallest single interval that contains the correct value.

The width of a sensor's value determines its inaccuracy. The following theorem gives an upper bound on the number of faulty sensors one can have

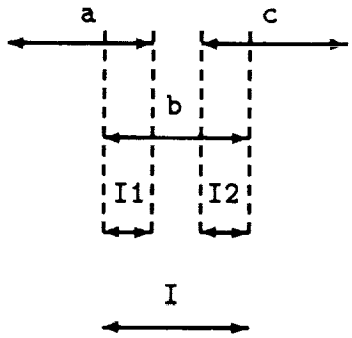

Figure 3: Intersection with $n = 3$ and $f = 1$

and still have a bound on the inaccuracy on the result. Define the operations $\mathbf{min}_i$ and $\mathbf{max}_i$ to be the i^{th} smallest and largest values of their operand (a set of values) respectively. Note $\mathbf{min}_i = \mathbf{max}_{n-i+1}$. For example, if $\mathcal{S} = \{13, 14, 15\}$ then $\mathbf{min}_3(\mathcal{S}) = \mathbf{max}_1(\mathcal{S}) = 15$.

Theorem 1 *If* $f < \left\lfloor \frac{n+1}{2} \right\rfloor$ *and* $\cap_{f,n}(\mathcal{S}) \neq \emptyset$, *then*

$$|\cap_{f,n}(\mathcal{S})| \leq \mathbf{min}_{2f+1}\{|s| : s \in \mathcal{S}\}$$

If $f \geq \left\lfloor \frac{n+1}{2} \right\rfloor$, the derived interval can be more inaccurate than any sensor in the system. An example is shown in figure 4[4].

Theorem 1 bounds the accuracy of the derived sensor in terms of the accuracy of the values read. However, if this bounding sensor is faulty, one might worry about the bound being meaningful. Consider the sensors shown in figure 5. If abstract sensor c is faulty and we allow faulty sensors to be arbitrarily inaccurate, the derived sensor can be more inaccurate than any correct sensor.

[4]Unless stated otherwise, all the proofs of the theorems given in this section are presented in section 3.4.

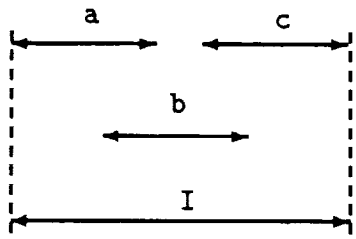

Figure 4: Intersection with $n = 3$ and $f = 2 \geq \left\lfloor \frac{n+1}{2} \right\rfloor$

In many applications, there exists a limit on the inaccuracy of a sensor. For example, if our abstract temperature sensors are all polled at roughly the same time, the accuracy of the abstract sensors will not differ significantly from each other. Thus, the bound of theorem 1 is reasonable. If sensors can have widely differing accuracy, however, fewer failures can be tolerated. Theorem 2 gives this bound.

Theorem 2 *Let C be the (unknown) subset of S that are correct. If $f < \left\lfloor \frac{n}{3} \right\rfloor$ and $\cap_{f,n}(S) \neq \emptyset$, then*

$$|\cap_{f,n}(S)| \leq \mathbf{min}_{f+1}\{|s| : s \in C\}$$

Under the conditions of theorem 2, a minimum of four sensors is necessary to tolerate a single faulty sensor. Figure 6 is an example of this case.

3.2 Probabilistic Interpretation

The previous section gives upper bounds on the inaccuracy of the correct abstract sensor calculated by algorithm 1. However, the actual accuracy can be much better. If an abstract sensor $\overline{s}_i(t)$ is correct, we can define the probability distribution function of the physical value in the range $\overline{s}_i(t)$. Let this probability that $s(t) = s' \in \overline{s}_i(t)$ be $f_i(s')ds$. The *expected value* of

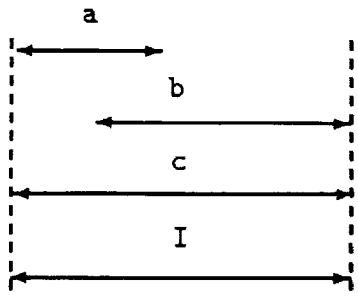

Figure 5: Intersection with sensor c faulty

the accuracy of the fault–tolerant sensor, $E(|\cap_{f,n}(\mathcal{S})|)$, has the following property.

Theorem 3 *Assume that each sensor is distributed identically and is independent of each other. If*

$$f_i(\min \overline{s}_i(t)) > 0 \text{ and } f_i(\max \overline{s}_i(t)) > 0$$

then

$$\lim_{n\to\infty} E(|\cap_{f,n}(\mathcal{S})|) = 0 \text{ for any fixed } f$$

The rate of convergence to the correct physical value depends on the actual distribution function. For example, let f_i be the uniform distribution function

$$I_i(s) = \begin{cases} \frac{1}{|\overline{s}_i(t)|} & s \in \overline{s}_i(t) \\ 0 & \text{otherwise} \end{cases}$$

In this case, the expected value of $|\cap_{f,n}(\mathcal{S})|$ approaches zero at a rate of $O(1/n)$; here, even a modest amount of replication can yield a very accurate abstract sensor. A proof of this rate of convergence, along with a proof of theorem 3 can be found in [Mar84].

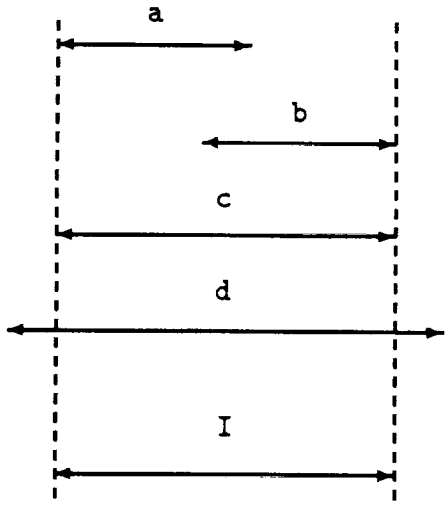

Figure 6: Intersection with $n = 4, f = 1$ and sensor d faulty

As an aside, the probabilistic interpretation of this section might suggest a fuzzy logic approach to this problem [Zad75]. The interpretation is straightforward: define $\wedge_{f,n}(\mathcal{S}, s)$ as the disjunction of the $\binom{n}{n-f}$ terms formed by the conjunction of $n - f$ fuzzy boolean values b_i whose membership is taken from the uniform probability distribution function $I_i(s)$. $\wedge_{f,n}$ is equivalent to the intersection of the $n - f$ cliques[5] in S. If we changed algorithm 1 to return a set of intervals rather than a single interval, $\wedge_{f,n}$ and $\cap_{f,n}$ would be equivalent.

3.3 Other Failure Models

If a sensor is known *a priori* to be faulty, the bounds given in theorems 1 and 2 can be improved. Faulty sensors need not be included in $\mathcal{S}$ when calculating $\cap_{f,n}(\mathcal{S})$, and n and f are reduced accordingly. In particular, if we can identify f faulty sensors, we can calculate a correct abstract sensor

[5] A k–clique is an intersection of k intervals.

that is at least as accurate as the most accurate sensor. More formally, let $\mathcal{W}$ be the subset of $\mathcal{S}$ that are known to be faulty, and let $|\mathcal{W}| = w \leq f$. Then, $\cap_{f-w,n-w}(\mathcal{S} - \mathcal{W})$ is a correct interval, and theorems 1 and 2 hold.

The main problem is determining *when* a sensor is faulty. Ideally, a sensor could be self-diagnosing, and return the value $\emptyset$ when faulty. Following [Sch84], we will call such sensors *fail–stop*. With fail–stop sensors, we can tolerate up to $n-1$ failures and will calculate a sensor that is *at least* as accurate as the most accurate correct sensor and *at best* approaches the exact physical value.

Of course, fail–stop sensors are useful only if they can be implemented. We can use algorithm 1 to detect failed sensors under an arbitrary failure model. This algorithm is very simple: any sensor in $\mathcal{S}$ that does not intersect $\cap_{f,n}(\mathcal{S})$ cannot contain the correct value, and is therefore incorrect and has failed.

Algorithm 2 *Detecting failed sensors.*

> *Specification:* Given n sensors $\mathcal{S}$ and a maximum number of faulty sensors f, find a subset of the sensors in $\mathcal{S}$ that are incorrect.
>
> *Implementation:* Let $\cap_{f,n}(\mathcal{S})$ be the interval calculated by algorithm 1. If $\mathcal{W}$ is the set of sensors that are known to be incorrect, add to $\mathcal{W}$ the sensors
>
> $$\{s : s \in \mathcal{S} \wedge s \cap (\cap_{f,n}(\mathcal{S})) = \emptyset\}$$

It is likely that algorithm 2 will fail to detect some of the incorrect sensors. For example, using algorithm 2 with the sensors in figure 3 yields $\mathcal{W} = \emptyset$, since we cannot tell which of the two sensors a, c is incorrect.

So far, we have assumed that once a sensor fails it remains failed, so once a sensor is added to $\mathcal{W}$ it will remain in $\mathcal{W}$. This assumption may not be realistic for sensors, since an abstract sensor may maintain no state. It seems natural to assume a sensor may occasionally fail in an apparently malicious way, and then "heal" itself and subsequently yield correct values. So, a natural extension to the arbitrary failure model is to assume that

at all times t there exists a set of *faulty sensors* $\mathcal{F}(t)$ such that $\mathcal{F}(t)$ may differ from $\mathcal{F}(t')$ when $t \neq t'$, and $\forall t : |\mathcal{F}(t)| \leq f$. Unfortunately, we cannot construct a correct abstract sensor under these conditions. We must also guarantee that there exists a period Δ such that the number of failures in all intervals of time no longer than Δ the number of failures is bounded:

$$\exists \Delta > 0 \; \forall t : |\{s \in \mathcal{S} : \exists \delta : 0 \leq \delta \leq \Delta : s \in \mathcal{F}(t + \delta)\}| \leq f$$

If algorithm 1 obtains fresh values from each physical sensor used in calculating $\mathcal{S}$ and algorithm 1 runs no longer than Δ seconds, it can still be used to construct a correct sensor. As $\Delta \to \infty$ this model becomes the earlier arbitrary failure model.

3.4 Proofs

To prove theorems 1 and 2, we will need a few lemmas.

Lemma 1 *Let S be a set of intervals containing at least one c–clique. Furthermore, suppose that all c–cliques in S have exactly i intervals in common with each other. Then,*

$$|S| \geq \begin{cases} i & i \geq c \\ 2c - i & i < c \end{cases}$$

Let S be a smallest set of intervals such that all c–cliques in S have exactly i intervals in common. By definition, this set must contain an i–clique, so $n \geq i$. If $i \geq c$, the smallest set consists of only the i–clique, so $n = i$. If $i < c$, the set must contain more than one c–clique, for otherwise the single c–clique has c intervals in common with itself. Since S has the smallest possible number of intervals, it must contain two c–cliques. These two cliques have i intervals in common, so each has $c - i$ intervals not in common with each other. The minimum number of intervals meeting this condition is $n = 2(c - i) + i = 2c - i$. □

Lemma 2 *If $n \geq 2c$, there exists a set of intervals S such that all the c–cliques in S share no intervals.*

Let S contain two distinct c–cliques and $n - 2c$ distinct 1–cliques. This set satisfies the lemma. □

Lemma 3 *Given a set of n intervals S containing at least one c–clique ($n < 2c$), the smallest number of intervals i in common with all c–cliques in S is $n - 2c$.*

Suppose S has no more than $i' < i$ intervals in common with all c–cliques. By lemma 1, the smallest such set contains $2c - i' > n$ intervals, a contradiction. □

Lemma 4 *Let $s \in \mathcal{S}$ be any member of all maximal cliques of $\mathcal{S}$. The closure of the intersection of the maximal cliques is no larger than s.*

The intersection of any maximal clique cannot contain any point outside of s, since by definition that point is not in an intersection containing s and s is a member of each clique. The closure only adds points between the intersections. Since $\mathcal{S}$ is a set of intervals over the reals, s must contain all points between the maximal cliques, so the closure does not add any points in s. Since all the points in the closure are also in s, the closure cannot be larger than s. □

Theorem 1 can now be shown. By algorithm 1 the maximal clique in $\mathcal{S}$ must be at least as large as f, for otherwise $\cap_{f,n}(\mathcal{S}) = \emptyset$. By lemma 3 at least $n - 2f$ intervals intersect all cliques. By lemma 4 the closure of the intersection cannot be larger than any of these $n - 2f$ intervals. The closure, however, may be larger than any of the remaining $2f$ intervals. In the worst case, these remaining intervals are the smallest ones in $\mathcal{S}$, and the theorem follows. Additionally, by lemma 2 we know that the bound on f is an upper bound. □

Theorem 2 can now be shown. By theorem 1,

$$|\cap_{f,n}(\mathcal{S})| \leq \mathbf{max}_{n-2f}\{\forall s \in \mathcal{S}\}$$

For $|\cap_{f,n}(\mathcal{S})|$ to be bounded by a correct sensor, $n - 2f > f$ or $n > 3f$. This corresponds to the faulty sensors being the most inaccurate, so

$$|\cap_{f,n}(S)| \leq \min_{f+1}\{\forall s \in C\}$$

□

4 Example

The methodology presented in this paper is not an automatic one. It may take some work to use abstract sensors, in that the original specification may have to be changed to accommodate abstract sensors, and it may be difficult constructing a set of independent abstract sensors. In this section, we show how a specification can be converted from using physical state variables to using abstract sensors, and how an abstract sensor can be implemented from a physical sensor.

As part of the *Cornell Real Time Reliable Distributed Systems* project, we are developing a correct process control program from its specification. One of the problems we have chosen is that of a train traversing a sequence of n track segments. Associated with each track segment i is a *track circuit* σ_i that, when nonfaulty, is *true* if and only if the train occupies that track segment. Assume that segment i spans the positions c_i through c_{i+1} where $(\forall i : 0 \leq i < n : c_i < c_{i+1})$. The train has position x and velocity v, has zero length[6], starts at position c_0 and moves in the direction of increasing x (towards c_1). Each track segment has a minimum and maximum speed min_i and max_i; if the train exceeds these limits, it will derail. Additionally, there is a random *communications delay* associated with all messages in the system that is bounded by δ seconds.

Our safety condition is

$$S \stackrel{\text{def}}{=} c_i \leq x \leq c_{i+1} \Rightarrow min_i \leq v \leq max_i$$

This condition is expressed in terms of physical variables, so we need to change S. The obvious condition is

$$S' \stackrel{\text{def}}{=} (\exists x \in \overline{x} : c_i \leq x \leq c_{i+1}) \Rightarrow$$

[6]This is not an unreasonable assumption; given a train of length L and a track system K, one can construct another track system K' on which a train of zero length is constrained in exactly the same way as the L-length train is on K.

$$(\forall v \in \overline{v} : min_i \leq v \leq max_i)$$

since this also excludes the unsafe states (at a penalty of running the train conservatively).

We will show how an abstract sensor of position $\overline{x}_i$ can be constructed from the track circuits σ_i. The simplest way to do so is to assume a bound the velocity of the train $v \leq v_{max}$. Define the global array:

var train[i]: {before, in, after} := before;

Define a *polling process* for each track circuit σ_i. Note the delay is represented by a **delay** statement; the implementation must ensure that no more than Δ seconds elapse between successive polls of a sensor. The value of Δ must be small enough that the polling process does not "miss" the train traversing the track segment it is monitoring: $\Delta \leq (c_{i+1} - c_i - \delta v_{max})/v_{max}$. The assertion I is a loop invariant, and t is the current time.

process Poll[i] =
begin
$\{I$: train[i] = before $\Rightarrow 0 \leq x(t) < c_i + \delta v_{max} \wedge$
train[i] = in $\Rightarrow c_i \leq x(t) \leq c_{i+1} + \delta v_{max} \wedge$
train[i] = after $\Rightarrow c_{i+1} < x(t) \leq c_n\}$
do true $\rightarrow$
delay Δ;
if $\sigma_i \wedge$ (train[i] = before) $\rightarrow$ train[i] := in
$\Box \neg\sigma_i \wedge$ (train[i] = in) $\rightarrow$ train[i] := after
$\Box \neg\sigma_i \wedge$ (train[i] = before) $\rightarrow$ **skip**
$\Box \sigma_i \wedge$ (train[i] = in) $\rightarrow$ **skip**
$\Box$ (train[i] = after) $\rightarrow$ **skip**
fi
od;
end

The abstract sensor, defined functionally, comes from the loop invariant I and the distance the train could have moved since the last time σ_i was read:

$$\begin{aligned}\overline{x}_i = \textbf{if } & \text{train[i]} = \text{before} \rightarrow [0 \mathinner{..} c_i + (\delta + \Delta)v_{max}] \\ \Box\ & \text{train[i]} = \text{in} \rightarrow [c_i \mathinner{..} c_{i+1} + (\delta + \Delta)v_{max}] \\ \Box\ & \text{train[i]} = \text{after} \rightarrow [c_{i+1} \mathinner{..} c_n] \\ \textbf{fi}\end{aligned}$$

This is a simplistic abstract sensor for x. One can define a more accurate $\overline{x}_i$ by noting the time a track circuit first comes on. The implementation of this sensor is more complex, but has a structure similar to the one given here.

5 Discussion

This paper presents a four–step process, through which a program written in terms of physical state variables is transformed to one that reads the physical state variable through a set of physical sensors, some of which may be faulty. The degree of sensor replication depends on the failure model we assume. Figure 7 summarizes the maximum number of faulty sensors that can be tolerated for the three failure models considered in this paper.

Failure Model	f_{max}	min n: $f = 1$
arbitrary failures with unbounded inaccuracy	$\lfloor (n-1)/3 \rfloor$	4
arbitrary failures with bounded inaccuracy	$\lfloor (n-1)/2 \rfloor$	3
fail–stop failures	$n-1$	2

Figure 7: Maximum failures for different error models

The methodology presented here is incomplete, however. For example, there are other reasonable failure models that we are investigating. While some of these will undoubtedly reduce to the models presented here, others may not. We have also only considered sensors that read a single physical value from a real domain. There are other kinds of sensors; for example,

a sensor denoting whether or not a door is open, or a sensor that returns the altazimuth coordinates of an airplane. We are currently extending the material in this paper to accommodate these more general sensors.

Acknowledgements This work profited from several discussions the author had with Fred Schneider, Sam Toueg and Jacob Aizikowitz. In addition, Ken Birman, Amitabh Shah, and Mark Wood read and commented on earlier versions of this paper.

References

[Lam84] Leslie Lamport. Using time instead of timeout for fault-tolerant distributed systems. *ACM Transactions on Programming Languages and Systems*, 6(2):254–280, April 1984.

[LSP82] L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, July 1982.

[Mac84] I. M. MacLeod. Data consistency in sensor–based distributed computer control systems. In *Proceedings of the Fourth International Conference on Distributed Computing Systems*, pages 440–446. IEEE Computer Society, May 1984.

[Mar84] Keith Marzullo. *Maintaining the Time in a Distributed System.* PhD thesis, Stanford University, Department of Electrical Engineering, June 1984.

[MO83] Keith Marzullo and Susan Owicki. Maintaining the time in a distributed system. In *Proceedings of the Third Symposium on Principles of Distributed Computing*, pages 295–305. ACM SIGPLAN/SIGOPS, 1983.

[MS85] Steve Maheney and Fred Schneider. Inexact agreement: Accuracy, precision, and graceful degredation. In *Proceedings of the Fourth Symposium on Principles of Distributed Computing*, pages 237–249. ACM SIGACT/SIGOPS, August 1985.

[Sch84] Fred B. Schneider. Byzntine generals in action: Implementing fail-stop processors. *ACM Transactions on Computer Systems*, 2(2):145–154, May 1984.

[Sch86a] Fred B. Schneider. Combining theory and practice. Lecture at IBM Workshop on Fault–Tolerant Distributed Computing, Asolomar, CA, March 1986.

[Sch86b] Fred B. Schneider. The state machine approach: A tutorial. Technical Report TR 86-600, Cornell University, Dept. of Computer Science, Upson Hall, Ithaca, NY 14853, December 1986.

[Zad75] L. A. Zadeh. Fuzzy logic and approximate reasoning. *Synthese*, 30:407–428, 1975.