Byzantine Fault Tolerance, from Theory to Reality

Conference Paper · September 2003

DOI: 10.1007/978-3-540-39878-3_19 · Source: springerlink.metapress.com

4 authors, including:

Brendan Hall
Honeywell
34 PUBLICATIONS **509** CITATIONS

SEE PROFILE

Byzantine Fault Tolerance, from Theory to Reality

Kevin Driscoll[1], Brendan Hall[1], Håkan Sivencrona[2], Phil Zumsteg[1]

[1]Honeywell International
3660 Technology Drive, Minneapolis, MN 55418
{brendan.hall,kevin.driscoll,phil.j.zumsteg}@Honeywell.com
[2]Chalmers University of Technology
Department of Computer Engineering, SE-412 96 Göteborg, Sweden
sivis@computer.org

Abstract. Since its introduction nearly 20 years ago, the Byzantine Generals Problem has been the subject of many papers having the scrutiny of the fault tolerance community. Numerous Byzantine fault tolerant algorithms and architectures have been proposed. However, this problem is not yet sufficiently understood by those who design, build, and maintain systems with high dependability requirements. Today, there are still many misconceptions relating to Byzantine failure, what makes a system vulnerable, and indeed the very nature and reality of Byzantine faults. This paper revisits the Byzantine problem from a practitioner's perspective. The intention is to provide the reader with a working appreciation of Byzantine failure from a practical as well as a theoretical perspective. A discussion of typical failure properties and the difficulties in preventing the associated failure propagation is presented. These are illustrated with real Byzantine failure observations. Finally, various architectural solutions to the Byzantine problem are presented.

1 What You Thought Could Never Happen

In English, the phrase "one in a million" is popularly used to describe the highly improbable. The ratio itself is difficult to comprehend. The easiest way to give it reason is to equate it to real-world expectations. For example, the probability of winning the U.K. National Lottery is around one in fourteen million; the probability of getting struck by lightning in the U.S. is around one in six hundred thousand [1]. It is not safe to rely on intuition for reasoning about unfathomably small probabilities (for example, the 1-in-1,000,000,000 maximum failure probability for critical aerospace systems[1]). It is problematic in two ways: (1) real-world parallels are beyond typical human experience and comprehension; (2) faults that are not recognized, such as Byzantine faults, are incorrectly assumed to occur with zero or very low probability. The lack of recognition causes additional issues in that it allows the manifestation of such faults to pass unnoticed or be otherwise misclassified, reinforcing the misconception of low probability of occurrence.

[1] Usually written as a failure rate of 10^{-9}/hr

The lack of recognition leads to repeating the "Legionnaire's Disease" phenomenon. After its "discovery" in 1976, a search of medical records found that the disease had seldom occurred for many decades. The lack of shared knowledge and the disease's rarity made each occurrence appear to be unique. Only after 1976 was it realized that all these "unique" occurrences had a common cause. Similarly, an observation of a Byzantine failure will not be recognized as being an instance of a known class of failure by those who are not intimately familiar with Byzantine failures.

The intent of this paper is to redress this situation. Drawing from the authors' experiences with Byzantine failures in real-world systems, this paper shows that Byzantine problems are real, have nasty properties, and are likely to increase in frequency with emerging technology trends. Some of the myths with respect to the containment of Byzantine faults are dispelled and suitable mitigation strategies and architectures are discussed.

2 The Byzantine Army Is Growing

The microprocessor revolution has seen electronic and software technologies proliferate into almost every domain of life, including an increasing responsibility in safety-critical domains. High assurance processes (e.g. DO-254 [2]) have been matured to manage the development of high integrity systems. Using high assurance processes is relatively expensive compared to equivalent levels of functionality in commercial counterparts. In recent years, there has been a push to adopt commercial off-the-shelf (COTS) technology into high integrity systems. The timing of the COTS push is alarming, considering the decreasing reliability and dependability trends emerging within the COTS integrated circuit (IC) arena. With increasing clock frequencies, decreasing process geometries, and decreasing power supply voltages, studies [3] conclude that the dependability of modern ICs is decreasing. This development reverses the historical trend of increasing IC dependability. Bounding the failure behaviors of emerging COTS ICs will become increasingly more difficult. The expected lifetime of ICs is also decreasing, as modes of "silicon wear-out" (i.e. time-dependent dielectric breakdown, hot carrier aging, and electro-migration) become more significant. The anticipated working life of current ICs may be on the order of 5-10 years. Although viable in the commercial arena, this is a world away from the requirements for high integrity systems, which have traditionally had deployment lifetimes ranging over multiple decades. Strategies to mitigate the problems are in development [4]. It is safe to assume that the anticipated rate of IC failure will increase and the modes of failure will become ever more difficult to characterize.

Another significant trend is the move towards more distributed, safety-critical processing system topologies. These distributed architectures appear to be favored for the emerging automotive "by-wire" control systems [5], where a new breed of safety critical communications protocols and associated COTS are being developed [6]. Such technologies, if developed in compliance with the traditional high assurance processes, show promise to answer the cost challenges of high integrity applications. However, it is imperative that such technologies are developed with full knowledge of all possible failure modes. Distributed control systems, by their very nature, require consensus among their constituent elements. The required consensus might be low-level (e.g. in the form of mutual synchronization) or at a higher level (e.g. in the form

of some coordinated system action such as controlling a braking force). Addressing Byzantine faults that can disrupt consensus is thus a crucial system requirement.

We expect Byzantine faults to be of increasing concern given the two major trends described in this section: (1) Byzantine faults are more likely to occur due to trends in device physics; (2) safety-critical systems are becoming more vulnerable due to increasing emphasis on distributed topologies. It is therefore imperative that Byzantine faults and failure mechanisms become widely understood and that the design of safety-critical systems includes mitigation strategies.

3 Parables from the Classical Byzantine Age

The initial definition of Byzantine failure was given in a landmark paper by Lamport et al [7]. They present the scenario of a group of Byzantine Generals whose divisions surround an enemy camp. After observing the enemy, the Generals must communicate amongst themselves and come to consensus on a plan of action—whether to attack or retreat. If they all attack, they win; if none attack, they live to fight another day. If only some of the generals attack, then the generals will die. The generals communicate via messages. The problem is that one or more of the generals may be traitors who send inconsistent messages to disrupt the loyal generals from reaching consensus.

The original paper discusses the problem in the context of oral messages and written signed messages that are attributed different properties. Oral messages are characterized as follows:

A1. Every message that is sent is delivered correctly (messages are not lost).
A2. The receiver of a message knows who sent it.
A3. The absence of a message can be detected.

For messages with these properties, it has been proven that consensus cannot be achieved with three generals, if one of the generals is assumed to be a traitor. A solution is presented in which each of the four generals exchange information with his peers and a majority vote makes selections over all of the data exchanged. This solution is generalized to accommodate multiple traitors, concluding: to tolerate m traitorous generals, requires $3m + 1$ generals utilizing $m + 1$ rounds of information exchange.

Written, signed messages assume all of the properties (A1-A3) of the oral messages, and are further characterized by the properties below:

A4. A loyal general's signature cannot be forged.
A5. Anyone can verify the authenticity of a signature.

Assuming the signed message properties above, it is shown that consensus is possible with just three generals, using a simple majority voting function. The solution is further generalized to address multiple fault scenarios, concluding: to tolerate m traitorous generals requires $2m + 1$ loyal generals and $m + 1$ rounds of information exchange.

The initial proofs presented in the paper assume that all generals communicate directly with one another. The assumptions are later relaxed to address topologies of less connectivity. It is proven that for oral messages, consensus is possible if the gen-

erals are connected in a *p* regular graph, where $p > 3m - 1$. For signed (authenticated) messages, it is proven that consensus is possible if the loyal generals are connected to each other. However, this solution requires additional rounds of information exchange to mitigate the lack of direct connectivity. It is shown that the required communication equates to $(m + d - 1)$ rounds, where *d* is the diameter of the sub-graph of loyal generals.

Since its initial presentation, nearly two decades ago, the Byzantine Generals problem has been the subject of intense academic study, leading to the development and formal validation of numerous Byzantine-tolerant algorithms and architectures. As stated previously, industry's recognition and treatment of the problem has been far less formal and rigorous. A reason for this might be the anthropomorphic tone and presentation of the problem definition. Although the authors warned against adopting too literal an interpretation, much of the related literature that has followed the original text has reinforced the "traitorous" anthropomorphic failure model. Such treatment has resulted in the work being overlooked by a large segment of the community. Practicing engineers, who intuitively know that processors have no volition and cannot "lie," may quickly dismiss concepts of "traitorous generals" hiding within their digital systems.

Similarly, while the arguments of unforgeable signed messages make sense in the context of communicating generals, the validity of necessary assumptions in a digital processing environment is not supportable. In fact, the philosophical approach of utilizing cryptography to address the problem within the real world of digital electronics makes little sense. The assumptions required to support the validity of unbreakable signatures are equally applicable to simpler approaches (such as appending a simple source ID or a CRC to the end of a message). It is not possible to prove such assumptions analytically for systems with failure probability requirements near 10^{-9}/hr.

The implications of the Byzantine Generals' Problem and the associated proofs to modern systems are significant. Consider a system-level requirement to tolerate any two faults (which includes Byzantine faults). Such a system requires seven-fold ($3m + 1 = 3(2) + 1 = 7$) redundancy. This redundancy must include all elements used in data processing and transfer, not just processors. The system also must use three rounds ($m + 1 = 2 + 1 = 3$) of information exchange. This corresponds to a twenty-one times increase in the required data bandwidth over a simplex system (seven-fold redundancy times three rounds of information exchange each).

To more easily identify the problem, concise definitions of Byzantine fault and Byzantine failure are presented here:

Byzantine fault: a fault presenting different symptoms to different observers.

Byzantine failure: the loss of a system service due to a Byzantine fault.

Note that for a system to exhibit a Byzantine failure, there must be a system-level requirement for consensus. If there is no consensus requirement, a Byzantine fault will not result in a Byzantine failure. For example, many distributed systems will have an implied system-level consensus requirement such as a mutual clock synchronization service. Failure of this service will bring the complete system down. Asynchronous approaches do not remove the problems; any coordinated system actions will still require consensus agreement. In addition, adequately validating asynchronous systems to the required assured failure rate of 10^{-9} failures per hour is usually much more difficult than for synchronous systems.

4 Meet the Byzantine Generals (They're real and they like to travel.)

Byzantine problems are not mythical. On the contrary, Byzantine faults in safety-critical systems are real and occur with failure rates far more frequently than 10^{-9} faults per operational hour . In addition, the very nature of Byzantine faults allows them to propagate through traditional fault containment zones, thereby invalidating system architectural assumptions. To illustrate these concepts, a discussion of physical Byzantine fault properties is presented below.

A typical example of a Byzantine fault is a digital signal that is stuck at "½", e.g. a voltage that is anywhere between the voltages for a valid logical "0" and a valid logical "1". Fig. 1 shows a logic gate's transfer function for a common 3.3 volt logic circuit family. For clarity, the transfer function has been shown with a much less steep slope than is typically found in these digital logic circuits.

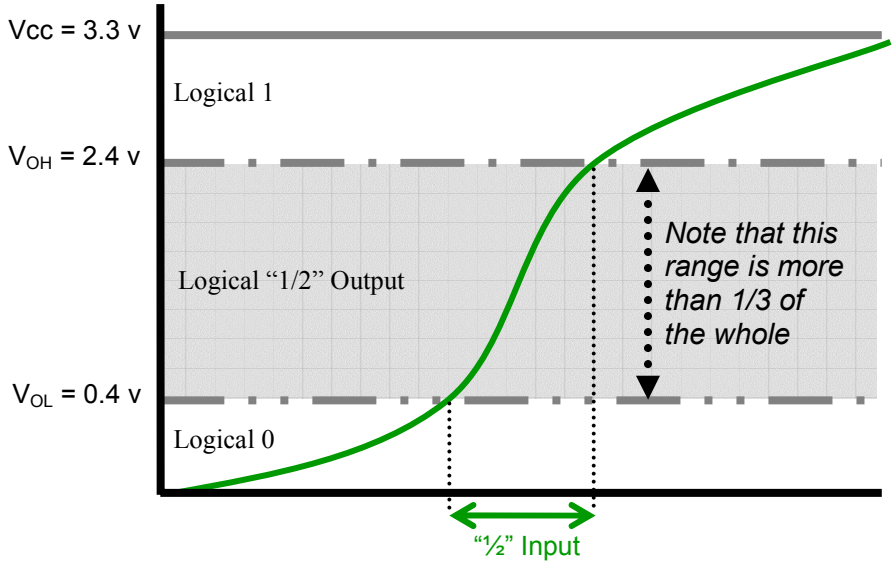

Fig. 1 Gate Transfer Function with "1/2" Areas Defined

Stuck at "1/2" behavior is commonly observed with CMOS bridging faults [8] or signal path "opens" (the most common type of fault). A similar behavior can be seen in a metastable flip-flop, which oscillates rapidly between a "0" and a "1", existing in neither state long enough to exhibit a valid output voltage. In fact, "stuck at 1/2" faults rarely produce constant (DC) signals. Because these voltages are in the range of the largest gain for a gate's transfer function, minute amounts of noise on a gate's input become a large amount of noise on the gate's output. Fig. 2 shows this effect; again, understating the slope (gain) of the transfer function for clarity of the figure. This happens because "digital circuits are just analog circuits driven to extremes." For any "½" signal output from a logic circuit, one can use the inverse of that circuit's transfer function to find an input voltage which will produce that output. Schmidt

triggers can help, but are not guaranteed to be effective. The high gain of modern digital circuits means that small noise riding on a ½ input signal becomes large noise on the output. Thus, ½ signals tend to be oscillatory, and their amplitude can easily exceed a Schmidt trigger's hysteresis.

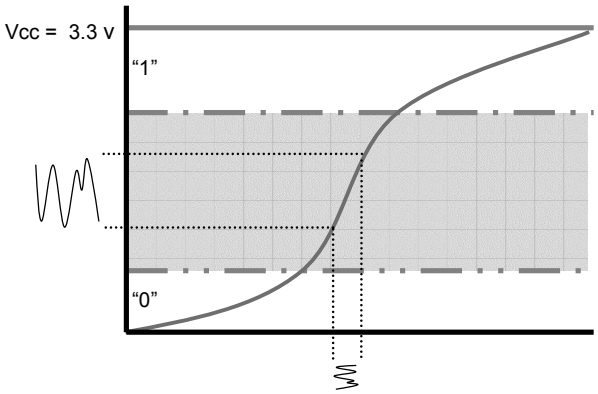

Fig. 2 Gate Transfer Function Showing "1/2" Noise Amplification

Receivers downstream of these signals may interpret them as either a "0" or a "1" depending on their respective thresholds, biases, gains, and timing. These ambiguous logic levels can propagate through almost any digital circuit. Digital gates can have differing transfer functions due to manufacturing tolerances, voltage variations, temperature variations, etc. (See Fig. 3) If one does a Failure Modes and Effects Analysis (FMEA) and from a point that could be adversely affected by a 1/2 input one backtracks through potential 1/2 fault propagation paths, one must consider the entire range of all possible transfer functions. Thus, for the case shown in Fig. 3, the range of input that must be considered is the sum of all three transfer functions shown.

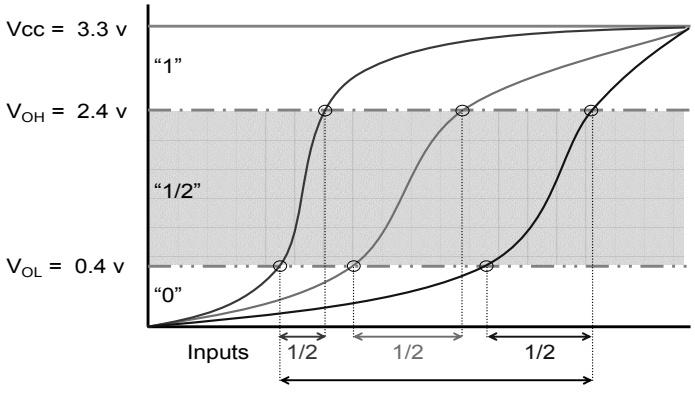

Fig. 3 Transfer Function Variation

As indicated in Fig. 4, such a signal can propagate through multiple stages of logic and still remain at an ambiguous level. The differences in the resulting values at the right side of the figure are due to normal manufacturing and environment differences, which permit their threshold voltages (V_T) to be anywhere in the range of V_{IL} to V_{IH}. Propagation through an arbitrary logic stage or a flip-flop cannot be guaranteed to "clean up" the logic level.

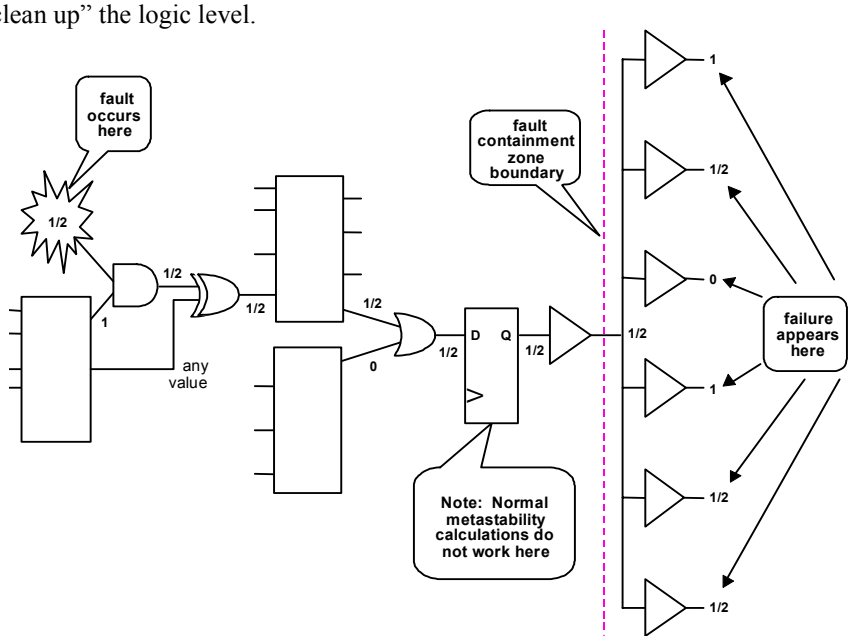

Fig. 4 Byzantine Failure Propagation

The only form of logic that will always prevent an ambiguous logic signal from propagating is "masking logic", the behavior of which removes the causality of the ambiguous logic level behavior. Examples of masking logic are an AND gate with one of its other inputs at a logic "0", and an OR gate with one of its other inputs at logic "1". In either case, the gate's output is solely determined by the masking action of the dominate signal ("0" for AND, "1" for OR). With a dominate input, other gate inputs can have no output effect (even a Byzantine input). The well-known 3-input majority logic "voter" is an illustrative composite example. If one input is ½ and the other two inputs are both 0 or both 1, then the output is 0 or 1 respectively (due to masking within the voter). When one input is ½ and the other two inputs are different values, the output can be 0, ½, or 1, depending on the specific gain and threshold voltages of the voter gates and the specific characteristics of the ½ signal. This is illustrated in Fig. 5 which shows the three main cases for the 2-of-3 majority logic function (Z = AB + BC + CA).

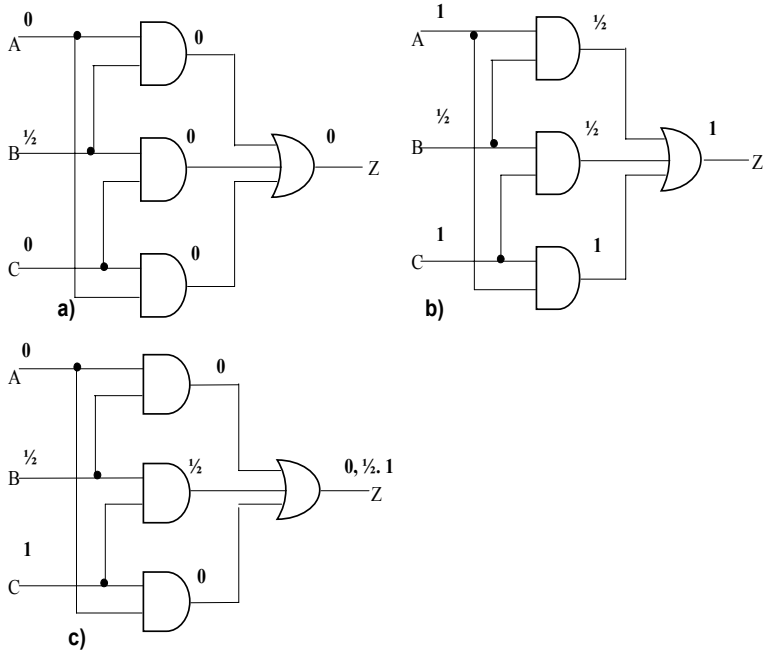

Fig. 5 Byzantine Fault Masking In a 2-of-3 Majority Logic Function

An XOR gate has no dominate input value; thus a ½ can always pass through it. A flip-flop with its data (or other input) at ½, can output ½ continuously. The commonly used metastability calculation [10] does not apply here because that equation assumes well-formed clock and data inputs that have known frequencies and uniformly distributed skews between the clock and the data.

To further illustrate the Byzantine propagation capability, one can envision a "Schrödinger's CRC" similar to the "Copenhagen" misinterpretation of "Schrödinger's Cat" [9] where the CRC is simultaneously correct for any interpretation of Byzantine data. The behavior of a ½ bit on a CCITT-8 CRC circuit is shown in Fig. 6. This figure shows 8 data bits (with one of the data bits to be transmitted stuck at ½) followed by 8 bits of CCITT-8 CRC. Because the transmitter's CRC calculation is a linear (XOR) combination of its data bits, each CRC bit affected by the ½ data bit can also be ½. The switching threshold voltages are shown for two receivers (***a*** and ***b***). These thresholds fall within the legal range of V_{IL} to V_{IH}. The resulting data received by ***a*** and ***b*** are different, but each copy has a correct CRC for its data. Other receivers of this signal can be divided in three camps: those that agree with ***a***, those that agree with ***b***, and those that see a bad CRC. While the failure behaviors in this

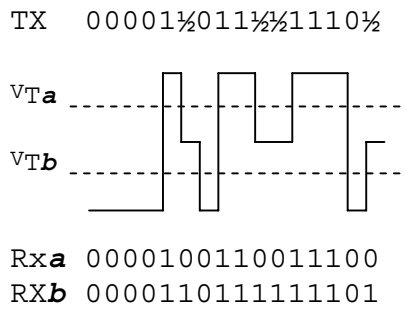

Fig. 6 A "Schrödinger's CRC"

example appear to be complex, they are all the natural result of a single stuck-at-½ fault somewhere in the transmitter. Thus, CRCs can provide no guarantee of protection against Byzantine fault propagation.

Although the ½ problem is easier to describe, the more common problems are in the time domain. These can be either on the micro-scale where data changing edges occur at the same time as sensing clock edges (i.e. the metastability problem) or on the macro-scale where an event occurs exactly at its deadline. For example, events in a real-time system must occur before a deadline. An event occurring exactly at its deadline can be seen as timely by some observers and late by other observers. Different observations occur because time is continuous, and no matter how tightly the receivers are synchronized to each other, the faulty signal can arrive between any arbitrary sets of deadline expirations. Tighter clock synchronization cannot solve this problem; it only can reduce the probability to some unknown level, which is useless in critical systems where proof of adequate design is needed.

Value and temporal domain affects can be combined in producing Byzantine faults. For example, as shown in Fig. 3, a rise time that has become artificially lengthened can present both amplitude and time symptoms. Fig. 7 shows how a long rise time (possibly due to a weak driver, increased line capacitance, etc.) can convert a normal transition between threshold voltages into a time difference (ΔT).

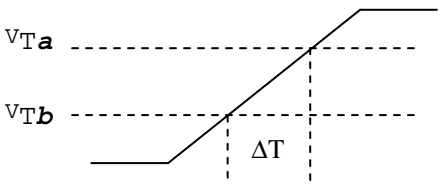

Fig. 7 Variation in Amplitude and Time

When making the required conservative estimate for the probability that a Byzantine fault effect will escape a fault containment (FC) zone[2], the failure probabilities of all the FC zone's components must be considered. For example, if all 9 components to the left of Fig 7's FC zone boundary resided in different integrated circuits, each with a one hour exposure failure probability of 10^{-6}, the probability of a Byzantine fault escaping the containment zone could be $1 - (1 - 10^{-6})^9 = 8.9 \times 10^{-6} \cong 10^{-5}$. A system with four such fault containment zones could have a failure probability of $1 - (1 - 8.9 \times 10^{-6})^4 = 3.6 \times 10^{-5}$ from Byzantine problems alone. This failure probability is so high that it would not meet the requirements for most dependable systems. Note that typical fault containment zones have many more than 9 components. Note that we say "could" instead of "would" because the actual probabilities are not known—all that is known is the chip failure rates. Because the proportion of failures that are Byzantine rather than benign is typically not known to any degree, one must assume the worst case. Note that the certification agencies for large transport aircraft do not allow one to make an unsubstantiated guess about the relative probabilities of different failure modes for electronic components.

5 The Generals Are Attacking

The Byzantine Generals have been observed attacking systems. The authors have personal experience observing Byzantine failures. As alluded to earlier, being aware of Byzantine behavior enabled these observations. Had this not been the case, the observed behaviors may have been dismissed as an unknown or extremely rare "anomaly". It is hoped that discussion of the Byzantine observations will show that the problem is real and dispel some myths about the characteristics of these faults.

5.1 The Time Triggered Architecture

The Time Triggered Architecture (TTA) [11] is a generic time-triggered computer architecture for fault-tolerant distributed real-time systems. Developed from over 20 years of research, TTA is targeted to address the needs of the emerging automotive "by-wire" industry. A key component of this architecture is a dedicated communications controller that implements a deterministic fault tolerant communications protocol: TTP/C. Due to its low target cost and high dependability characteristics, TTP/C has recently found significant acceptance within the aerospace arena. TTP/C can be characterized as a TDMA-based serial communications protocol that guarantees synchronization and deterministic message communication. In addition to these services, TTP/C is unique in that it provides a membership service at the protocol level. The function of the membership service is to provide global consensus on message distribution and system state.

[2] A fault containment zone is a set of components within a boundary designed to stop fault effect propagation.

Addressing the consensus problem at the protocol level can greatly reduce system software complexity; however, placing a requirement for protocol level consensus leaves the protocol itself vulnerable to Byzantine failure. Such were the findings of the Fault Injection Techniques in the Time Triggered Architecture (FIT) project [12].

As part of the FIT project, a first generation time-triggered communication controller (TTP-C1) was radiated with heavy ions [13]. The errors caused by this experiment were not controlled; they were the result of random radioactive decay. The reported fault manifestations were bit-flips in register and RAM locations.

During the many thousands of fault injection runs, several system failures due to Byzantine faults were recorded [12]. The dominant Byzantine failure mode observed was due to marginal transmission timing. Corruptions in the time-base of the fault-injected node led it to transmit messages at periods that were slightly-off-specification (SOS), i.e. slightly too early or too late relative to the globally agreed upon time base. A message transmitted slightly too early was accepted only by the nodes of the system having slightly fast clocks; nodes with slightly slower clocks rejected the message. Even though such a timing failure would have been tolerated by the Byzantine tolerant clock synchronization algorithm [14], the dependency of this service on TTP/C's membership service prevented it from succeeding. After a Byzantine erroneous transmission, the membership consensus logic of TTP/C prevented nodes that had different perceptions of this transmission's validity from communicating with each other[3]. Therefore, following such a faulty transmission, the system is partitioned into two sets or cliques—one clique containing the nodes that accepted the erroneous transmission, the other clique comprising the nodes that rejected the transmission.

TTP/C incorporates a mechanism to deal with these unexpected faults—as long as the errors are transient. The clique avoidance algorithm is executed on every node prior to its sending slot. Nodes that find themselves in a minority clique (i.e. unable to receive messages from the majority of active nodes) are expected to cease operation before transmitting. However, if the faulty node is in the majority clique or is programmed to re-integrate after a failure, then a permanent SOS fault can cause repeated failures. This behavior was observed during the FIT fault injections. In several fault injection tests, the faulty node did not cease transmission and the SOS fault persisted. The persistence of this fault prevented the clique avoidance mechanism from successfully recovering. In several instances, the faulty node continued to divide the membership of the remaining cliques, which resulted in eventual system failure.

In later analysis of the faulty behavior, these effects were repeated with software simulated fault injection. The original faults were traced to upsets in either the C1 controller time base registers or the micro-code instruction RAM [15]. Later generations of the TTP/C controller implementation have incorporated parity and other mechanisms to reduce the influence of random upsets (e.g. ROM based microcode execution). SOS faults in the TTA are also mitigated with a central guardian, as discussed in Section 6.3.

[3] In TTP/C, agreement on global state is a prerequisite for communication.

5.2 Multi-Microprocessor Flight Control System

The Multi-Microprocessor Flight Control System (MMFCS) was developed by Honeywell Labs during the late 1970's. The system pioneered the concepts of self-checking pairs and utilized a dual self-checking pair bus distribution topology (total 4 busses) between nodes of fail-silent, self-checking processing boards. The system's self-checking pair comparisons of processors, bus transmission, and bus reception enabled the precise detection of Byzantine faults and the ability to differentiate them from other classes of faults.

During testing of the MMFCS system prototype, Byzantine failures were observed with a mean repetition period of 50 seconds with a variance of 10 seconds. Because this was in a 20 Hz control loop, the probability of any loop experiencing a Byzantine fault was 1/1000. The root cause of these failures was isolated to the marginal behavior of the physical layer that had been pushed to the limits in the initial prototype set-up.

A common fallacy is to assume random events are uniformly distributed in time (e.g. if we have a 1 in 500 year weather event this year, it will be about another 500 years before it happens again). In reality, the precipitating conditions which cause an event may well persist, so that the probability of clustering is high. Similarly, there is a myth that Byzantine faults are so rare that they will occur in isolation. The MMFCS observations (and those from the TTA FIT above), seriously contradict that myth. In both of these cases, the fault was persistent—typical behavior of an SOS fault. Such faults form a subset of Byzantine faults that occur repeatedly (either permanent or intermittent) due to a shift in a device's characteristics such that it is on the edge of being able to provide its intended service correctly.

5.3 Quad-Redundant Control System

A further example illustrates the fact that having enough good hardware is not sufficient to achieve Byzantine fault tolerance; information exchange is required. The system outlined in Fig. 8 comprised quad redundant processing elements that act on shared data collected by remote data concentrators (DC). Each data concentrator communicated via its own dedicated bus. On first examination, the system would appear to be classical in its approach; however, there was no congruence exchange between the processing elements. The data was used from the data concentrators as is. It was initially assumed that all processing would receive the same data, as they were connected to the same source.

This system failed due to a Byzantine fault that was caused by an incorrect termination resistance on one of the DC-to-processor links. This bad termination caused reflections on one of the data concentrator buses. The processing elements located at nodes and anti-nodes of the reflected interference received different message values. In the worst manifestation, this situation resulted in a 2:2 split of the digital redundancy that forced the system offline, leaving an independent 5^{th} back-up processing function as the only operational unit.

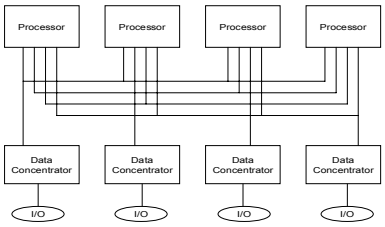

Fig. 8 A Quad Redundant Control System

The above example also illustrates the dangers of Byzantine fault propagation that may invalidate the system failure assumptions; the loss of a single termination resulted in the complete loss of the digital system redundancy. As with the previous examples, the fault that led to the SOS manifestation was hard, and the SOS condition persisted.

5.4 Potential Large Economic Impact Example

If a system is not originally designed to tolerate Byzantine faults, ensuing accidents or recalls due to their occurrence can be very expensive. The possible economic impact is illustrated in an incident where Byzantine failures threatened to ground all of one type of aircraft. This aircraft had a massively redundant system (theoretically, enough redundancy to tolerate at least two Byzantine faults). But, no amount of redundancy can succeed in the event of a Byzantine fault unless the system has been designed specifically to tolerate these faults. In this case, each Byzantine fault occurrence caused the simultaneous failures of two or three "independent" units. The calculated probability of two or three simultaneous random hardware failures in the reporting period was 5×10^{-13} and 6×10^{-23} respectively. After several of these incidents, it was clear that these were not multiple random failures, but a systematic problem. The fleet was just a few days away from being grounded, when a fix was identified that could be implemented fast enough to prevent idling a large number of expensive aircraft.

6 Byzantine Battle Plans

Effective methods for dealing with Byzantine faults can be divided into three types: full exchange (e.g. the SIFT [16], FTMP [17], and SPIDER [18] architectures), hierarchical exchange (e.g. the SAFEbus [19] architecture), and filtering (e.g. TTP star topologies [20]). These methods can be used separately or in conjunction with each other. The first method directly implements the exchanges described in the classical Byzantine papers. The second method uses private exchanges inside subsets of a system's nodes followed by simplified exchanges between the subsets. The third method tries to remove a Byzantine fault's asymmetry via "filtering". An example of each method is given below.

6.1 Full Exchange Example—SPIDER

The Scalable Processor-Independent Design for Electromagnetic Resilience (SPIDER) [18] implements a classical approach to Byzantine fault mitigation using full message exchange and voting. The ultra-Reliable Optical BUS (ROBUS) provides a time-division, multiple-access broadcast bus between N simplex general-purpose nodes (PE n) connected via the ROBUS. The topology is shown in Fig. 9.

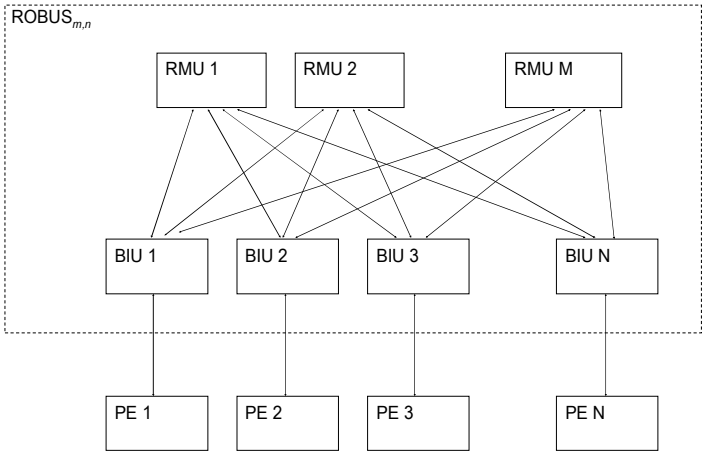

Fig. 9 SPIDER - Classical Full Exchange

The *n* bus interface units (BIU) and the *m* redundancy management units (RMU) provide the fault containment zones, which are required to fail independently. The SPIDER architecture incorporates an interactive consistency (IC) protocol that guarantees consistent message exchange. This protocol has been formally verified to provide containment of Byzantine faults. It requires each BIU to send its message to all RMUs; which in turn, forward the messages to all other BIUs. The BIUs then perform a majority vote to ascertain the message status and validity. Note that it is the "masking logic" of the voters and an assumption that RMUs do not propagate Byzantine signals that combine to provide Byzantine fault tolerance.

SPIDER leverages the IC protocol to implement other system services such as synchronization and a system-level diagnosis. Diagnosis enables re-configuration that further enhances the system fault tolerance by preventing faulty components from participating in votes.

The SPIDER project is being developed as a demonstrator for certification under the new DO-254 guidelines.

6.2 Hierarchical Exchange Example—SAFEbus®

Honeywell's SAFEbus (ARINC 659) [19] uses self-checking pair (SCP) buses and SCP BIUs to ensure that Byzantine faults are not propagated (i.e. a Byzantine input will not cause the halves of a pair to disagree). The hardware configuration is shown in Fig. 10.

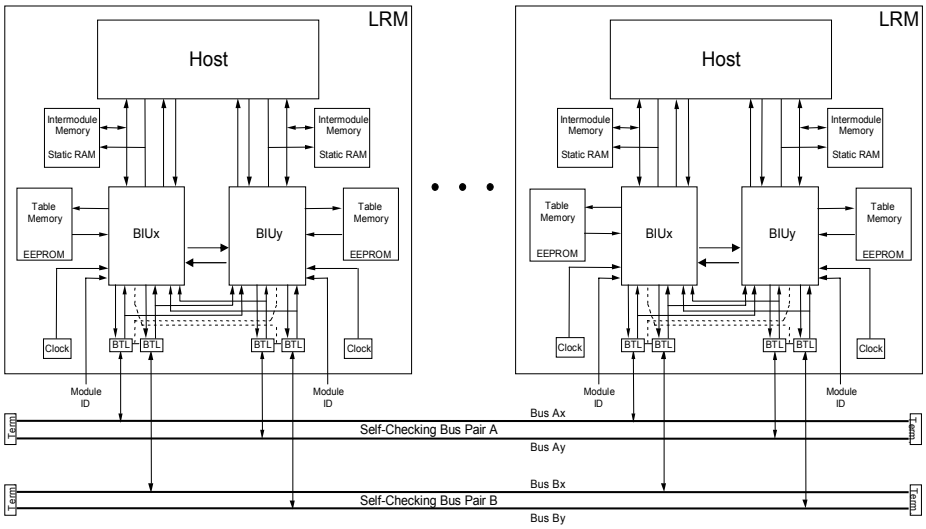

Fig. 10 Two Hosts Connected Via SAFEbus

The private exchange of "syndromes" between BIUs in a pair provides the "masking logic" that prevents Byzantine fault propagation. Each BIU produces a 4-bit syndrome vector that is the result of all 4 ways of comparing an X and a Y bus (see Fig. 10). The BIUs AND their syndromes together and use the result to select the same good input from the busses (if both BIUs agree that there is at least one good input). Consensus among multiple pairs can be achieved by sending only message reception status in a subsequent round of message exchange. SAFEbus® is the only standard bus topology that can tolerate a Byzantine fault, made possible by SAFEbus®'s full-coverage, fault-tolerant hardware.

6.3 Filtering Example—TTP Star

The TTP star topology [20] was developed in response to the FIT project findings described previously. The aims of the star topology are to provide the architecture with a truly independent guardian function and a mechanism to prevent systematic failure due to any persistent SOS node failure.

The TTA star employs centralized filtering to remove the asymmetric manifestation of a Byzantine fault. The star actively reshapes the line signals, transforming ½ signals into valid logic signals. SOS timed transmissions are truncated if they lie too

far away from the guardian's notion of the expected transmission time. Thus, only well-timed transmissions (guaranteed to be temporarily valid to all receivers in the system) will propagate.

The filtering and guardian functions are replicated, one instance for each of the two channels.

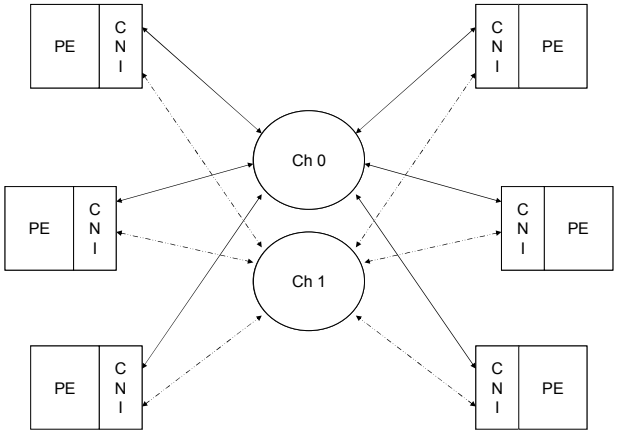

Fig. 11. TTP Star Topology

Note that because the filtering scheme has no "masking logic", it must be designed such that a proof of its Byzantine fault coverage can be constructed. This method cannot be guaranteed to work for all faults. Very careful design is needed to allow pseudo-exhaustive testing to establish a coverage factor meeting system dependability requirements.

6.4 Equivalence of Full Exchange and Filtering

It is important to note that most, if not all, of the existing solutions based on classical full-exchange techniques, have an instance of the filtering method buried inside. That is, they all assume that the second round of exchange via the intermediaries is non-Byzantine and that the intermediaries have suitably filtered the data. Thus, as with the direct filtering approach, these classical solutions must also include pseudo-exhaustive testing as part of their coverage arguments to validate such assumptions.

7 Conclusions

The anthropomorphic style of the "Byzantine Generals Problem" academic literature and the relative rarity of this problem compared to the actual hands-on experience time of most practitioners has led to too many practitioners believing that the Byzantine Generals Problem is either a total myth or is at least so rare that it can be ignored. The Byzantine Generals Problem is not a myth. The Problem is very real. In fact, the

Byzantine Generals Problem can be caused by many common faults -- including the most common of all, a CMOS open. Another myth that Byzantine faults are only isolated transients is contradicted by real experience.

The propensity for Byzantine fault effects to escape normal fault containment zones can make each Byzantine fault a threat to the whole system dependability. System designers could try fault avoidance: Byzantine faults can be prevented from causing a Byzantine Problem by designing systems such that consensus is not needed. However, it is extremely difficult or impossible to design highly reliable systems that do not need some form of consensus. The only practical solution is Byzantine fault tolerance; this is also known as consistency or congruency. To achieve this fault tolerance, a system must have one or more mechanisms specifically designed for dealing with these faults. To tolerate F number of faults, the mechanisms reported in the literature require 3F+1 fault sets, F+1 rounds of communications, and one or more "Byzantine fault filters" that convert Byzantine signals into non-Byzantine signals. To be assured that a system meets its dependability requirements, these filters must be shown to have adequate coverage. This coverage proof requires specialized testing of the electronic circuits used in the filter.

Anyone designing a system with high dependability requirements must thoroughly understand these failures and how to combat them.

8 Acknowledgement

We would like to thank Wilfredo Torres for bringing to our attention the fact that most existing Byzantine agreement mechanisms have within their design one or more "filtering" functions as explained in sections 6.3 and 6.4.

References

1. NOAA/American Red Cross: Thunderstorms and Lightning, safety brochure. (1994)
2. RTCA Inc.: DO-254, Design Assurance Guidance for Airborne Electronic Hardware.
3. C. Constantinescu: Impact of Deep Submicron Technology on Dependability of VLSI Circuits. In: Proc. Dependable Systems and Networks (2002)
4. Systems Standard & Technology Council: Avionics Process Management Committee. http://www.geia.org/sstc/APM/.
5. Kelling, N., Heck, W.: The Brake Project—Centralized Versus Distributed Redundancy for Brake-By-Wire Systems. Paper No 2002-01-0266, SAE (2002)
6. TTTech Computertechnik AG, “Specification of the TTP/C Protocol V1.0”
7. Lamport, L., Shostak, R., Pease, M.: The Byzantine Generals Problem. In: ACM Transactions on Programming Languages and Systems, 4(3): 382– 401 (1982)
8. Lavo, D., Larrabee, B., Chess, T.: Beyond the Byzantine Generals: Unexpected Behavior and Bridging Fault Diagnosis. In: Proc. Int. Test Conference, 611–619 (1996)
9. Bohr, N.: The quantum postulate and the recent development of atomic theory. Nature, 121, 580–89 (1928). Reprinted in Quantum Theory and Measurement.
10. Chaney, T.: Measured Flip-Flop Responses to Marginal Triggering. In: IEEE Transactions of Computers, Vol. C-32, No. 12 (December 1983) 1207–1209.
11. Kopetz, H. Real-Time Systems. Design Principles for Distributed Embedded Applications. Kluwer Academic Publishers, Boston (1997)

12. Fault Injection for TTA. Deliverable 5.1–5.5 Combined Report IST 1999 10748.
13. H. Sivencrona, P. Johannessen, M. Persson, J. Torin, Heavy-ion Fault Injection in the Time-triggered Communication Protocol, Proc. 1st Latin American Symposium on Dependable Computing LNCS 2847, pp. 69-80 (Oct 2003)
14. Pfeifer, H., Schwier, D., von Henke, F. W.: Formal Verification for Time Triggered Clock Synchronization. In: Proc. 7th IFIP International Working Conference on Dependable Computing for Critical Applications (Jan 1999)
15. Ademaj. A, Slightly-Off-Specification Failures in the Time Triggered Architecture. In: 7th IEEE Int. Workshop on High Level Design Validation and Test (Oct, 2002)
16. Wensly, J. H., Lamport, L., Goldberg, J., Levitt, K. N., Melliar-Smith, P. M., Shostak, R. E., Weinstock, C. B.: SIFT : Design and Analysis of fault tolerant computer control for aircraft. In: Proceedings of IEEE 66(10):1240–1255 (1978)
17. Hopkins, A., Smith, T. Lala, J.: FTMP—A Highly Reliable Fault Tolerant Multiprocessor for Aircraft. In: Proceedings of IEEE 66(10):1221–1239 (1978)
18. Miner, P., Malekpour, M., Torres, W.: A Conceptual Design for a Reliable Optical Bus (ROBUS). Proc. 21st Digital Avionics Systems Conference (2002)
19. Hoyme, K., Driscoll, K.: SAFEbus. In: Proc. 11th Digital Avionics Systems Conference. (October 5-9, 1992)
20. Kopetz, H., Bauer, G., Poledna, S.: Tolerating Arbitrary Node Failure in the Time-Triggered Architecture. Doc No 2001-01-0677, SAE (2001)