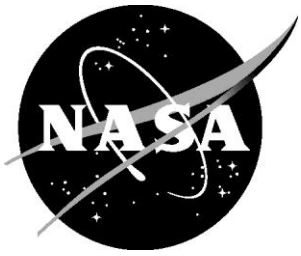
NASA/CR–2013-217963



Application Agreement and Integration Services

Kevin R. Driscoll, Brendan Hall, and Kevin Schweiker
Honeywell International, Inc., Golden Valley, Minnesota

February 2013

NASA STI Program . . . in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA scientific and technical information (STI) program plays a key part in helping NASA maintain this important role.

The NASA STI program operates under the auspices of the Agency Chief Information Officer. It collects, organizes, provides for archiving, and disseminates NASA's STI. The NASA STI program provides access to the NASA Aeronautics and Space Database and its public interface, the NASA Technical Report Server, thus providing one of the largest collections of aeronautical and space science STI in the world. Results are published in both non-NASA channels and by NASA in the NASA STI Report Series, which includes the following report types:

- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA Programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counterpart of peer-reviewed formal professional papers, but having less stringent limitations on manuscript length and extent of graphic presentations.

- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.

- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or co-sponsored by NASA.

- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

- TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services also include organizing and publishing research results, distributing specialized research announcements and feeds, providing information desk and personal search support, and enabling data exchange services.

For more information about the NASA STI program, see the following:

- Access the NASA STI program home page at *http://www.sti.nasa.gov*

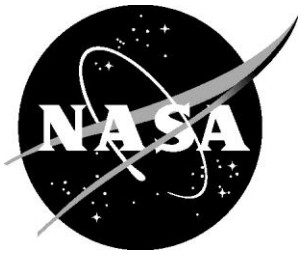
- E-mail your question to help@sti.nasa.gov

- Fax your question to the NASA STI Information Desk at 443-757-5803

- Phone the NASA STI Information Desk at 443-757-5802

- Write to:
  STI Information Desk
  NASA Center for AeroSpace Information
  7115 Standard Drive
  Hanover, MD 21076-1320

NASA/CR–2013-217963



Application Agreement and Integration Services

Kevin R. Driscoll, Brendan Hall, and Kevin Schweiker
Honeywell International, Inc., Golden Valley, Minnesota

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

Prepared for Langley Research Center
under Contract NNL10AB32T

February 2013

Available from:

NASA Center for AeroSpace Information
7115 Standard Drive
Hanover, MD 21076-1320
443-757-5802

Abstract

Application agreement and integration services are required by distributed, fault-tolerant, safety critical systems to assure required performance. An analysis of distributed and hierarchical agreement strategies are developed against the backdrop of observed agreement failures in fielded systems.

Contents

1 Introduction

The documented work was performed under NASA Task Order NNL10AB32T, Validation And Verification of Safety-Critical Integrated Distributed Systems – Area 2. This document is intended to satisfy the requirements for deliverable 5.2.11 under Task 4.2.2.3.

1.1 Scope

This report discusses the challenges of maintaining application agreement and integration services. A literature search is presented that documents previous work in the area of replica determinism. Sources of non-deterministic behavior are identified and examples are presented where system level agreement failed to be achieved. We then explore how TTEthernet services can be extended to supply some interesting application agreement frameworks.

This document assumes that the reader is familiar with the TTEthernet protocol. The reader is advised to read the TTEthernet protocol standard [1] before reading this document. This document does not re-iterate the content of the standard.

2 Background

2.1 Review of Replication and Agreement Service

2.1.1 Definitions

Flight Critical Systems (FCS) have to produce not only correct but timely results in a high assurance setting. Theoretically, there are two ways to meet these requirements: fault-avoidance and fault-tolerance. Fault-avoidance requires the development of an FCS integrated from components that have an extremely unlikely probability of failure when operating in all environments. The second approach, fault-tolerance, continues to provide correct and timely results despite the failure of one or more hardware or software modules [2]. While the fault-avoidance approach was predominant in the early days of computing, it quickly faded from use in FCS due to technological limitations.

Fault-tolerance overcomes these limitations, but requires redundancy in order to function after components have failed. Redundancy can be accomplished in three domains: information, time or space. Redundancy in the space domain is called replication, and is the subject of this report. For the last few decades, developers have used replication to assure that integrity and availability requirements of a modern FCS have been met. The idea of replication to gain fault-tolerance dates back to von Neumann[3] and is based on the assumption that replicated components are independently affected by faults. This assumption requires that the components of the FCS share no resources except for the communication media[4].

In general, replication may take place at different hardware or software levels, using either, or a combination of both. One approach for replication is to have identical hardware components running identical software and comparing the results on a bit for bit basis. A second replication approach is to use identical hardware with independently developed software. A third variant is to have dissimilar hardware performing computations.

Each of these replication approaches create a different set of challenges based upon the time synchronization of the FCS (synchronous, asynchronous, timed asynchronous) and the type of agreement (exact or approximate) that is desired.

For the remainder of this report, unless context or specific language indicates otherwise, the term synchronous will mean the definition that is common to the avionics and communication field (i.e., International Telecommunication Union (ITU)'s recommended definition) and not the

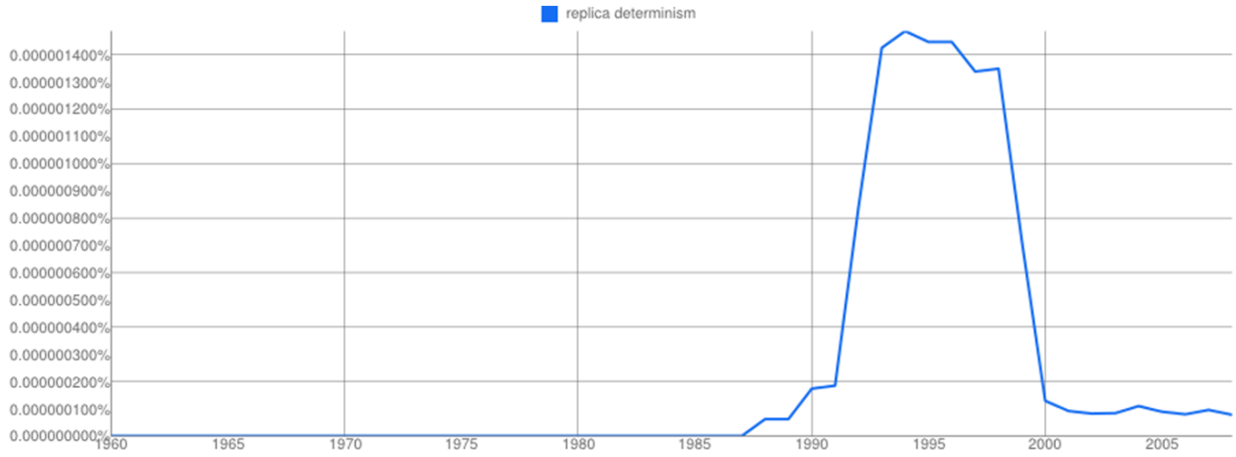


Figure 1. Google Ngram of *replica determinism*

definition of "bounded delay" as is used in papers dealing with consensus and a Byzantine generals problem. Similarly, unless context or specific language indicates otherwise, the term asynchronous will mean both not synchronous and not isochronous.

2.1.2 Discussion of the Literature

To support the review of research performed in the area of application agreement and integration services we used both emerging web search tools and traditional subject area databases. One such tool was based on the analysis of a database of n-grams. An n-gram is a contiguous sequence of n items from a given sequence of text or speech. For example *fault* is a 1-gram or a unigram, *fault tolerant* is a 2-gram, *fault tolerant system* is a 3-gram, and so on. Date stamped n-grams analysis provided us with a meta level tool to spot historic trends in research.

In this report, the n-grams were limited to the books and periodicals that have been scanned and analyzed by Google. Google estimates that approximately nearly 130 million books, magazine and periodicals have been published as of August 2010 [5]. It is also estimated that an additional 50 thousand titles are being published every year. The total collection of all printed material is estimated to include more than 4 billion pages and 2 trillion words in total. As of March 2012, Google had completed the scanning of 20 million of these titles [6]. While there have been legal issues regarding the scanned material, Google has made all the n-grams of the works available freely to the public.

We used Google's Ngram Viewer (http://books.google.com/ngrams) to explore relevant n-grams. Google Ngram is a corpus of n-grams compiled from data from the millions of books and periodicals that Google has scanned and were published between 1800 to 2008 (the latest Google Ngram data as of this writing). For our purposes we limited the Google Ngram Viewer to display results from 1960-2008.

In Figure 1 the Google Ngram for *replica determinism* shows research beginning to accelerate in the late 1980s, peak near 1995 then declines until 2000 where it achieved a steady state. This time line also leads the dot-com speculative bubble, which peaked in March of 2000 [7], and may reflect the research that was done to support the fault-tolerant server farms. It's interesting to note that the peak period of activity is bounded by Lamport's Part-Time Parliament research report in 1989 [8] and the journal release of the same title, after a lengthy delay, in 1998 [9].

Another interesting n-gram plot shown in Figure 2 is a comparison of *fly by wire* and *modular*

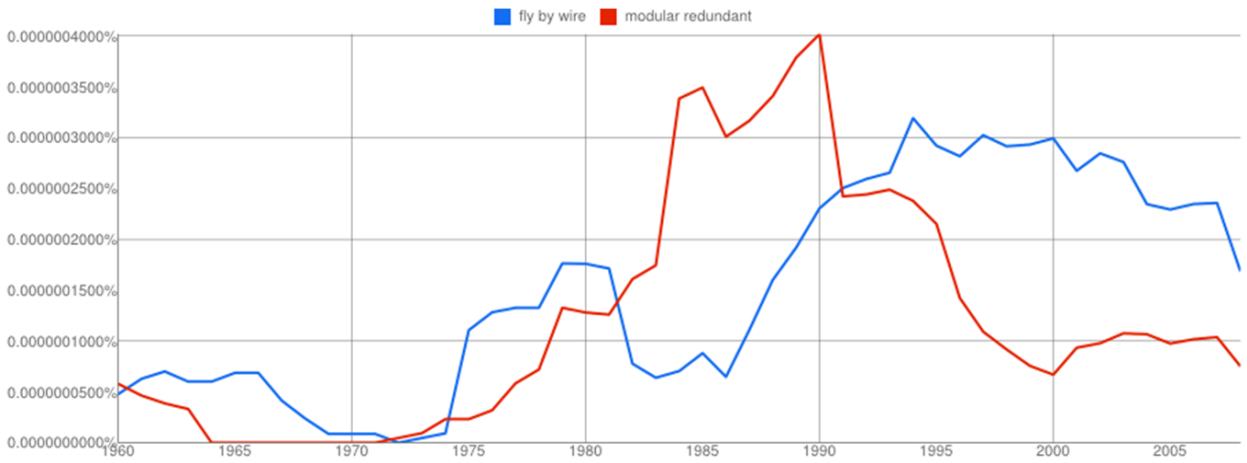


Figure 2. Google Ngram of *fly by wire* and *modular redundant*

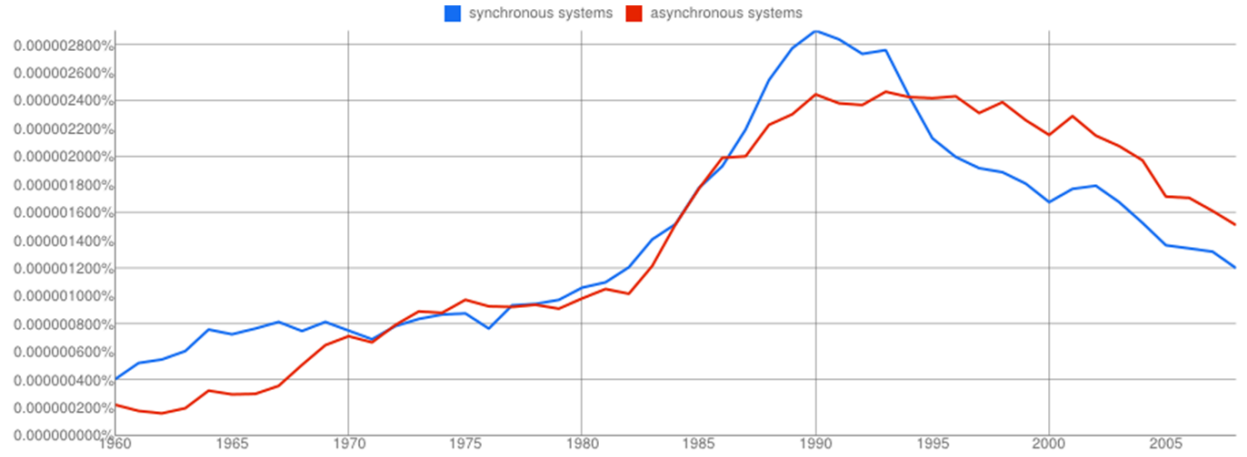


Figure 3. Google Ngram of *synchronous systems* and *asynchronous systems*

redundant n-grams. Fly-by-wire systems first saw service in military aircraft, starting with the F-16 and the space shuttle in the early 1970s, followed by commercial airliners [10]. Modular redundancy is a common form of redundancy management used in FCS. The first peak of the *modular redundant* n-gram corresponds with the first flight of the Airbus 320 in early 1987, the first commercial airliner with full digital fly-by-wire.

The n-grams in this report are shown in Figure 3 and is a comparison between *synchronous systems* and *asynchronous systems*. The activity level between "synchronous systems" and "asynchronous systems" is approximately the same over the time period examined.

It is interesting to note the increase in activity, between 1980 and 1990, corresponding to the development of the Airbus 320 and the Boeing 777, with slightly more occurrences of synchronous systems. In the early 1990s asynchronous systems started to receive slightly more attention.

The final n-grams in this report are shown in Figure 4 and is a comparison between *formal methods*, *theorem prover*, and *model checker*. The overall activity in formal methods peaks in the early 1990s and declines through 2008. The Google Ngram for theorem prover follows the same general trends as formal methods, while the Google Ngram for "model checker" shows a continuous increase in activity since 1980.

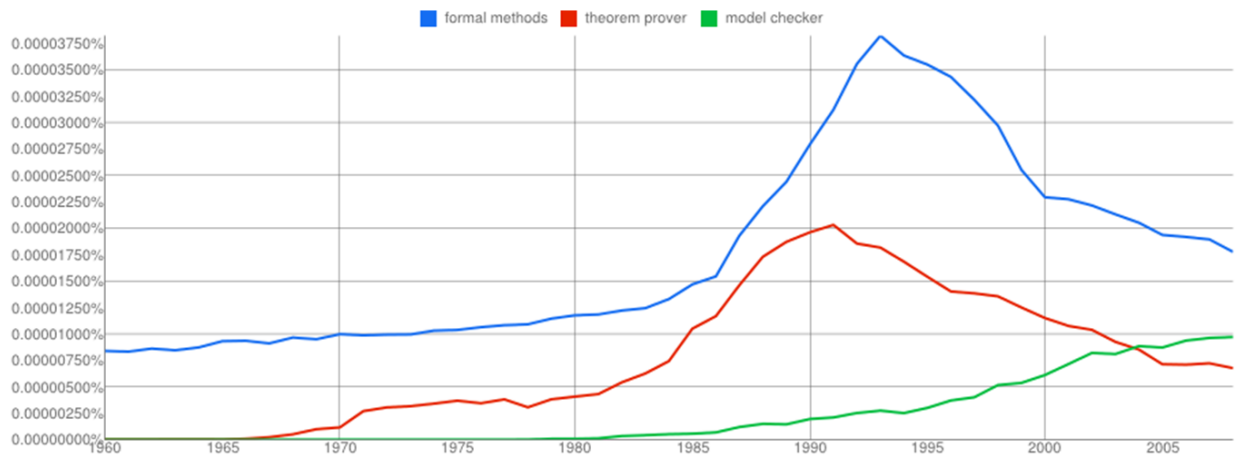


Figure 4. Google Ngram of *formal methods*, *theorem prover* and *model checker*

While the results from the Google Ngram Viewer are interesting, there are some considerations that need to be considered before drawing conclusions. First, Google's book-scanning project is underway, hence some "trends" may reflect the scanning sequence of the documents. A second limitation is that the tool is case specific, so a search of *Formal Methods* and *formal methods* will return different results. Another limitation, as of this writing, is that the tool does not yet support wild cards or stem searching. Ideally, if one wished to track the diffusion of the concept represented by the term *fault tolerant*, the term *fault tolerant ** should include: *fault tolerant computing*, *fault tolerant distributed systems*, *fault tolerant real time systems*, etc. Google Ngram also lacks the ability to group synonyms. For example, *byzantine fault tolerance* would not include *asymmetric fault tolerance*. Finally, n-grams (in general) provide neither context nor suggestions for topically related n-grams, for example a suggestion to examine *atomic broadcast* after a search on *replica determinism*.

The second graphical web search tool examined is the DEVONagent Pro (http://www.devontechnologies.com/products/devonagent/overview.html), currently only available on Apple platforms. DEVONagent Pro performs a web search of input keywords. Relevant documents are located on the web and examined (as far as possible before pay wall restrictions) for additional keywords. The related keywords are then placed on an interactive graph that allows one to surf the web by topically related keywords. This allows the researcher to become quickly aware of related technologies and trends.

Figure 5 shows keywords that are topically linked to *sal-inf-bmc*. Interestingly, *Bakery* is one of the keywords which is not conceptually related to the topic. DEVONagent Pro allows for the easy deletion of unwanted nodes. Figure 6 shows keywords that are topically linked to *fault tolerant replication*. Related keywords, such as *byzantine fault tolerance* are listed that may not have been considered earlier by the researcher.

A third graphical interface tool is TouchGraph(http://www.touchgraph.com/seo/), a Java-based tool that develops graph-based visualizations from multiple databases. The on-line package comes with a rudimentary Google visualization capability which displays the top ranked pages for the keywords requested. Nodes off the keywords can be clicked to visualize the top pages associated with the node. An illustration of the visualization of *fault tolerant distributed systems* is provided in Figure 7. Suggested keywords, *byzantine fault tolerance*, *replica determinism*, *replica non-determinism*, and *atomic broadcast* were clicked to form the graph illustrated.

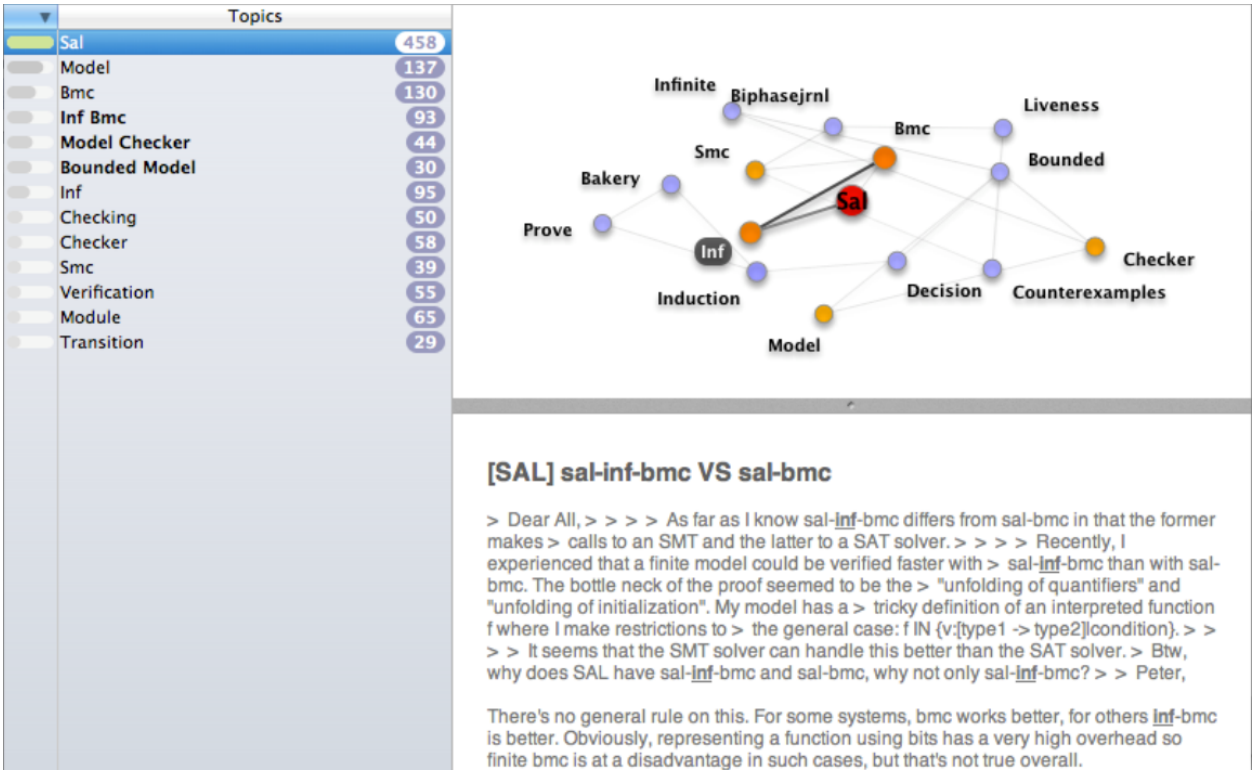


Figure 5. DevonAgent search results for *sal-inf-bmc*

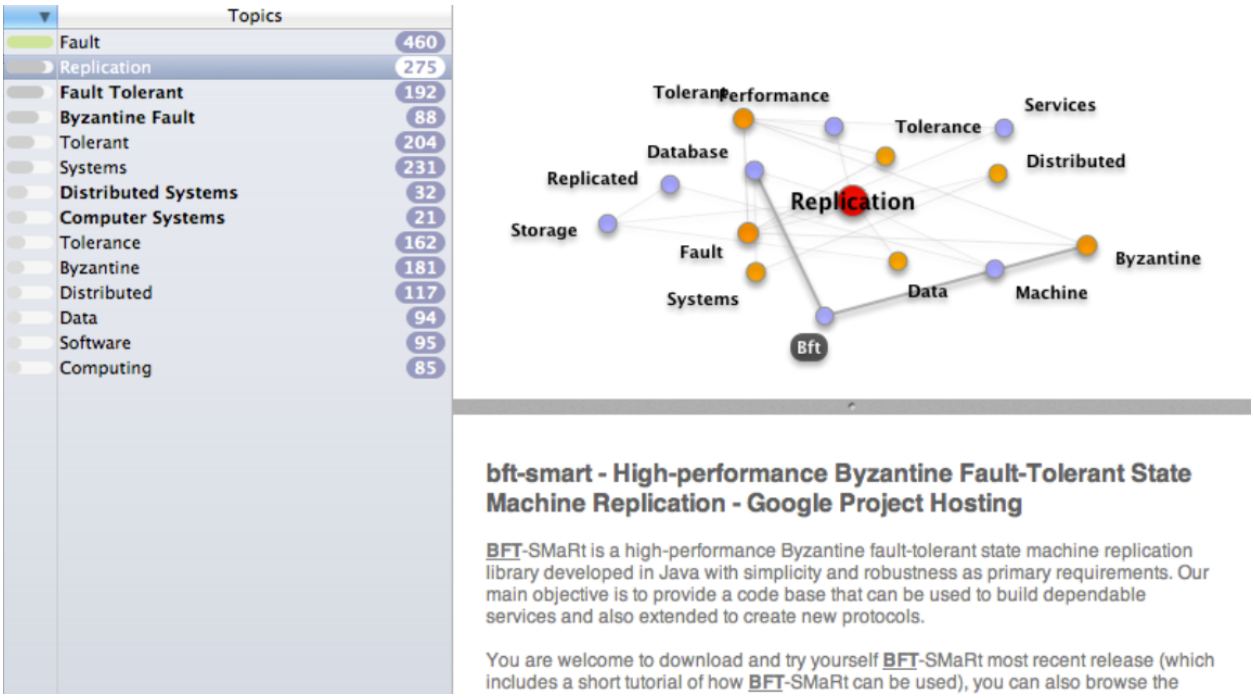


Figure 6. DevonAgent search results for *fault tolerant replication*

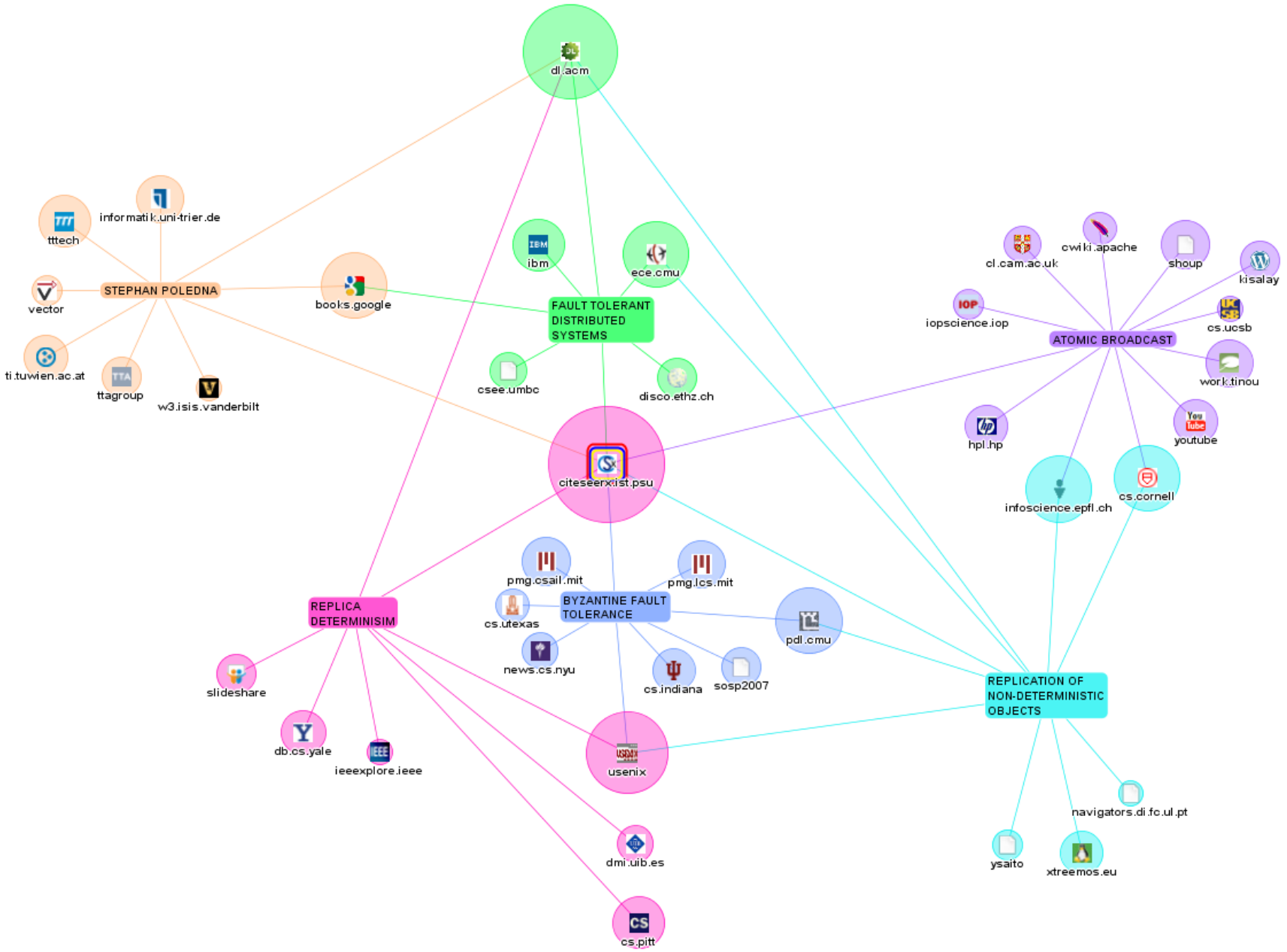


Figure 7. TouchGraph SEO results for *fault tolerant distributed systems*

As a result of our Google Ngram, TouchGraph, and DEVONagent analysis we decided to focus on the keywords of *replica determinism* and *atomic broadcast* for our detailed literature search. Honeywell consults three commercial databases for detailed literature searches: Compendex, Inspec and NTIS. Compendex(COMPuterized ENgineering inDEX) is a database related to nearly 200 distinct areas of engineering disciplines. Published by Elsevier, the database currently contains references to over 5,000 journals, conferences and trade publications. The database currently has more than 9 million artifacts and is updated on a weekly basis with approximately 10,000 new entries per week and contains records going back to 1969. The Inspec database, from the Institute of Engineering and Technology contains nearly 13 million artifacts in the fields of engineering and physics. The National Technical Information Service (NTIS) is supported by the U.S.Department of Commerce and contains nearly 3 million artifacts generated mainly by U.S. Government-sponsored non-classified research. The search of these three databases yielded 192 relevant entries. A selection of these are described below.

Zhao [11] contends that any practical applications contain some degree of non-determinism. When such applications are replicated to achieve Byzantine Fault Tolerant (BFT), their non-deterministic operations must be controlled to ensure replica consistency. Zhao believes that only the most simplistic types of replica non-determinism have been dealt with. He also points out that there lacks a systematic approach to handling common types of non-determinism. Zhao proposes a design and implementation of the mechanisms necessary to handle non-determinism within a Byzantine fault tolerance framework.

Poledna [12] explores the use of replication to achieve fault-tolerance. Poledna defines the problem of replica determinism as the assurance that replicated entities show consistent behavior in the absence of failures. Possible sources for replica non-determinism as well as basic requirements and strategies to enforce replica determinism are discussed. Replica determinism under real-time constraints is surveyed in the context of distributed systems. Synchronous and asynchronous replication strategies are also presented. Poledna's paper forms the major theoretical framework by which we examine application agreement and integration services in the remainder of this report.

Guerraoui and Schiper discuss consensus as a fundamental property that all distributed, fault-tolerant systems must possess [13]. The consensus problem requires agreement amongst a number of processes for a single data value. Their paper examines this topic from both a theoretical and practical point of view. Six common misunderstandings are examined, including: consensus is for theoreticians only; time-outs are enough; there is no life after FLP[1]; the failure detector model is unrealistic; time-free means inefficient; and asynchronous algorithms can not be used for time-critical applications.

2.1.3 Requirements for Replica Determinism

The problem when defining replica determinism is to cover a sufficiently broad range of replication strategies. Most often the definition is restricted to a class of replica strategies where each node within a distributed system is active and delivers outputs.

Poledna [12] defines replica determinism below. Here a server refers to a computational node in a distributed system and a group is a distributed system, such as a flight control system on an airliner.

> Correct replicas are showing correspondence of server outputs and/or service state

[1] In a fully asynchronous system, there is no consensus solution that can tolerate one or more crash failures, even when only requiring the non-triviality property. This result referred to as FLP impossibility after the authors Michael J. Fischer, Nancy Lynch, and Mike Paterson

changes under the assumption that all servers within a group are starting at the same initial state, executing the same service requests within a given time interval.

Therefore, replica determinism requires consensus. Consensus means that replicas start at the same initial state and agree on any inputs (e.g. service requests) that change their state or immediate output.

2.2 Sources of Non-Deterministic Behavior

Replication can occur at either the hardware, software, or a hybrid of the two levels. Non-identical replication of a software components is called N-version programming and is one technique used in FCS design to reduce the probability of common hardware errors, such as the Intel FDIV bug discovered in the mid 1990s [14]. If a given service or result is requested from a set of replicas then the results should be correct even if some of the replicas failed. Therefore, to assure replica determinism we must assure that there is no disagreement among non-faulty replicas.

In the following example of an FCS, multiple computers are used to read an analog sensor that corresponds to an aerodynamic control surface, such as a rudder, or an elevator. Multiple computers can read the sensors, but get slightly different readings. As a result, each computer may derive a different decision. Unless there is a way for the computers to reach consensus, the replication may result in non-determinism.

There are four major classes of drivers for replica non-determinism. These include:

1. Slightly different processing speeds of processing nodes in the distributed system, often caused by the drift of the clock oscillators.

2. Inconsistent inputs, often caused by sensors or non-atomic communication services.

3. Limited accuracy of computer arithmetic may lead to the consistent comparison problem, if services are implemented using independently developed software.

4. Use of non-deterministic functions, such as true random number generators.

Poledna [12] provides (on page 293 of the reference) the following list for potential replica non-determinism. Fortunately, the vast majority of these non-determinism sources do not apply to modern FCS designs. In the following quote from Poledna, the term "servers" can be equated to nodes within an FCS, the term "service requests" with FCS node inputs, and the term "groups" with cliques of nodes that can hear other nodes within their clique but not nodes in other cliques.

- `Inconsistent inputs:` If inconsistent input values are presented to servers then the outputs may as well be different. [...]

- `Inconsistent order:` Similarly, replica determinism is lost if service requests are presented in different order to servers under the assumption that service requests are not commutative. A typical example is a message queue where higher precedence messages may overtake lower precedence ones. If, however, one server has already started to act upon the lower precedence message, the two servers may act on different messages unless there is global coordination.

- `Inconsistent membership information:` Servers may join groups or depart due to errors. If there is inconsistent information on functioning members within a group then replica determinism is easily lost. Consider two servers a and b, server a assumes servers c, d and e are members of group G while server b assumes that c and d are members of group G. Thus

server a will send service requests to c, d and e, but server b will send requests only to c and d. Group G becomes inconsistent because servers c and d are acting on different requests than server e.

- `Non-deterministic program constructs:` Besides intended non-determinism, like random number generators, high level languages such as ADA, OCCAM or Fault-Tolerant Concurrent C (FTCC) have non-deterministic statements for communication or synchronization. [...]

- `Local information:` If decisions within a server are based on local events, that is information which is not readily available to other servers, then a server group loses its determinism. Uncoordinated access to a local clock is a typical example. Since it is impossible to achieve ideal synchronization on a set of distributed clocks, servers may get different clock readings even in the ideal case when all clocks are read at the very same time.

- `Timeouts:` A similar phenomenon can be observed if timeouts are used without global coordination. If the timeout is based on a local clock then the above entry "local information" applies. But even if an ideal global time service is available some servers will decide locally to timeout and others will not because of slightly different processing speeds.

- `Dynamic scheduling decisions:` Minimal processing speed differences of servers may lead to diverging scheduling decisions—even if the decision is based on global information. At the hardware level for example an asynchronous DMA-transfer may be scheduled immediately on one server but is delayed on another server, because its last instruction was not completed. [...]

There are additional items that (while may be implied) are not explicitly stated in Poledna's list. These include the following.

- `Consistent comparison problem:` Since computers can represent only finite sets of numbers, their accuracy is limited. Different implementations, e.g. CPUs from different manufacturers, may produce outputs that are similar, but not identical. Because floating-point calculations are not associative or commutative, different compilers can also cause similar, but not identical outputs. This can cause some degree of non-determinism between replica units.

- `Consistency cost:` Because the cost to reach consensus is so high when using the the well known mechanisms (see section 3 below), designs may not implement the mechanisms that are required to reach consensus. The consensus problem may be completely ignored or may be only partially solved. Partial solutions may or may not meet system dependability requirements. The latter is often caused by designers not understanding the consensus problem.

2.3 Industrial Implications of Practical Application Replication and Agreement

Cloud computing has emerged as a new paradigm in large-scale computing. While there are no known safety critical applications currently hosted on the cloud, the financial incentives for providing fault-tolerant services are large. Amazon's EC2 [15] is currently one of the most widely used cloud computing environments. The point-to-point latency between replicated instances is currently non-deterministic. Due to this variability, several classes of High Performance Computing (HPC) applications, such as bulk-synchronous parallel (BSP) applications, perform poorly because they are bound by the highest observed latency per iteration. Research is underway on state-machine

replication within the EC2 to address this problem. Process replication on EC2 requires that individual processes be deterministic and that they can be replicated without introducing conflicting states. This requires a trade-off between bandwidth, CPU time and memory for latency[16].

Oracle has applied symmetric replication to extend the asynchronous snapshot replication capabilities that were part of their earlier database releases. According to the taxonomy defined by Davidson et al.[17], symmetric replication uses eventual mutual consistency as a correctness criterion.

Poledna [18] investigated the problem of replica determinism in the context of distributed fault-tolerant real-time systems. His examination of fault tolerant practices within the aerospace, automotive, and industrial controls communities revealed that each industry addressed replica determinism in an unstructured, application specific, ad hoc manner.

2.4 Examples of System Agreement Failures

In the history of fault tolerance, there have been many examples of system agreement failures that have caused serious impairments or complete loss of fielded systems. Many of these failures are due to Byzantine faults [19]. In the remainder of this subsection, several examples of faults that led to system disagreement are given. Particular attention should be given to the proximate cause of the disagreement (typically a poorly designed fault detection, isolation, and recovery mechanism) rather than the phenomenological (physics based) event that starts the chain of events leading to the disagreement.

2.4.1 Space Transportation System

NASA's space shuttle has experienced several examples of agreement failures due to incorrect handling of Byzantine faults between its Multiplexer Demultiplexer (MDM) units and its General Purpose Computer (GPC). These faults fall within the class that the shuttle developers called "non-universal I/O error". The MDMs act as remote I/O concentrators for the GPCs. Data from the MDMs are transferred to the GPCs over data busses that are similar to MIL STD 1553. The GPCs execute redundancy-management algorithms that include a Fault Detection, Isolation, and Recovery (FDIR) function having specific handling for the "non-universal I/O error" class of failure. However, these FDIR algorithms were not correctly designed to handle Byzantine faults. Given that there were four GPCs, the shuttle had sufficient redundancy to tolerate a Byzantine fault, if these FDIR algorithms had been designed correctly.

In one of the earliest examples (some 25 years ago), this failure was triggered by a technician putting incorrect terminating resistors on the end of a data bus. Because of the impedance mismatch between the characteristic impedance of the data bus and resistance of the terminating resistors, signals on the data bus were reflected off of the resistors. These reflections caused a standing wave on the data bus. Two of the four GPCs happen to be connected to the data bus at nodes of the standing wave and the other two GPCs were connected to the data bus at anti-nodes of the standing wave (see Figure 8). Because of this, two of the GPCs disagreed with the other two GPCs. It was lucky that this irreconcilable 2:2 disagreement happened in the lab.

A more recent example of this problem came closer to causing a disaster. At 12:12 GMT 13 May 2008, a space shuttle was loading its hypergolic fuel for mission Space Transportation System (STS)-124 when a 3-1 disagreement occurred among its GPCs (GPC 4 disagreed with the other GPCs). Three seconds later, the split became 2-1-1 (GPC 2 now disagreed with GPC 4 and the other two GPCs). This required that the launch be stopped. During the subsequent troubleshooting, the remaining two GPCs disagreed (1-1-1-1 split). See the reports given in [20] and [21]. This was a

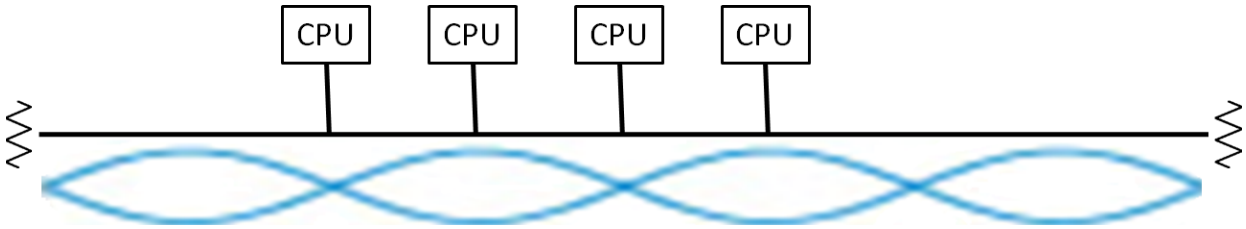


Figure 8. Standing wave caused by incorrect terminating resistors

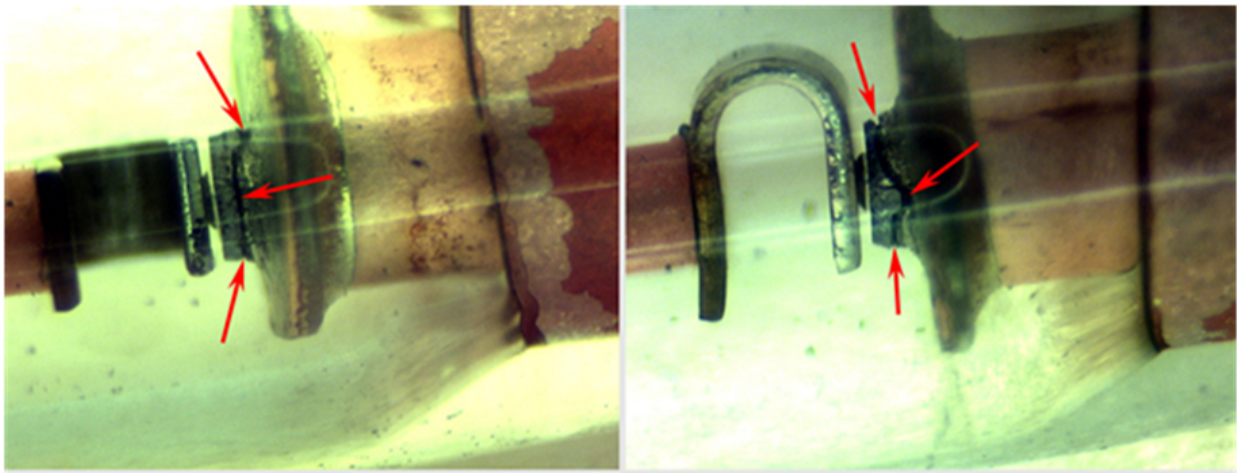


Figure 9. The diode crack

complete system disagreement. However, none of the GPCs was faulty. The fault was in the FA[2] 2 MDM. This fault was a crack in a diode. The photomicrographs in Figure 9 show two views of this diode, rotated 90 degrees. The dark wavy line pointed to by the red arrows is the crack. The current flow through diode is normally left to right through the material shown in these pictures. This means that the crack was perpendicular to the normal current flow and completely through the current path. As a crack opened up, it changed the diode into another component – a capacitor. This transformation is illustrated in Figure 10.

The normal signals that should appear on the data bus between the MDM and the GPCs are shown in Figure 11. The signals that were produced due to the diode failure are shown in Figure 12. Because some of the bits in the signal are smaller than they should have been, some of the GPCs' receivers could not see these bits. The ability to see these bits depends on the sensitivity of the

[2]FA nomenclature = Flight critical data bus category, Aft location

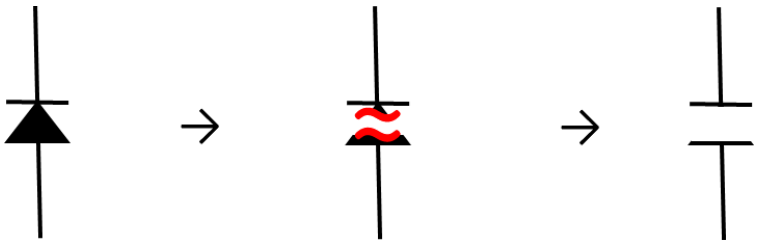


Figure 10. The transformation of a diode into a capacitor

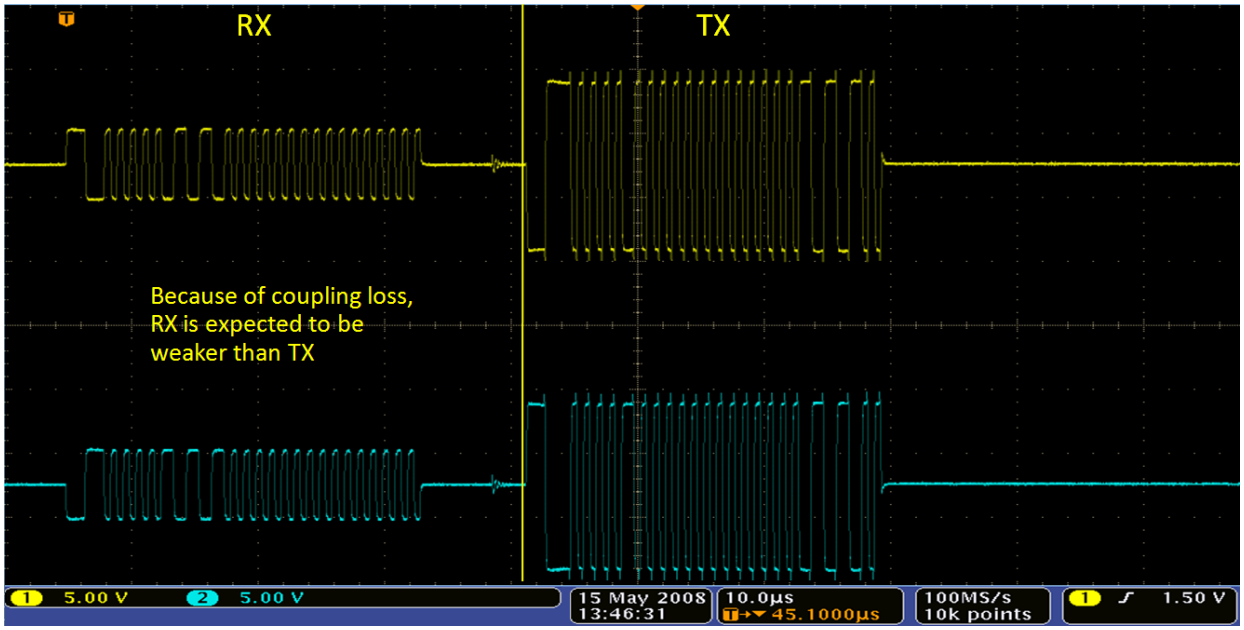


Figure 11. Normal bus signals

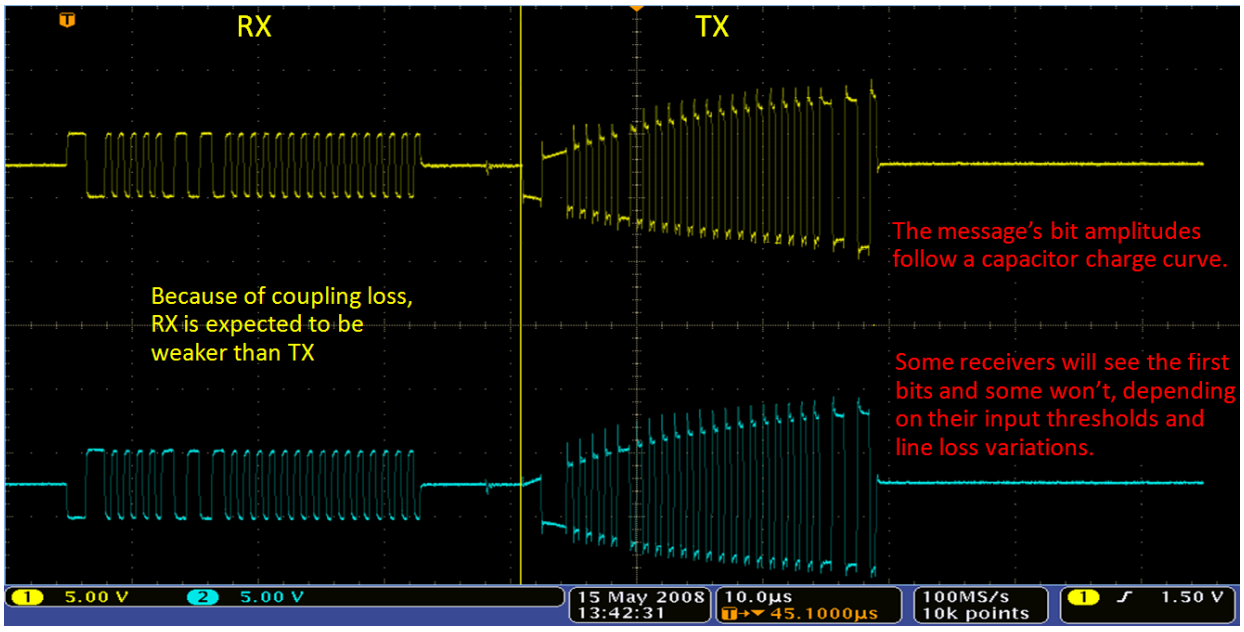


Figure 12. Bad bus signals caused by diode failure

receiver, which is a function of manufacturing variances, temperature, and its power supply voltage. From the symptoms, it is apparent that the receiver in GPC 4 was the least sensitive and saw the errors before the other three GPCs. This caused GPC 4 to disagree with the other three. Then, as the crack in the diode widened, the bits became shorter to the point where GPC 2 could no longer see these bits; which caused it to disagree with the other GPCs. At this point, the set of messages that was received correctly by GPC 4 was different from the set of messages that was correctly received by GPC 2 which was different again from the set of messages that was correctly received by GPC 1 and GPC 3. This process continued until GPC 1 and GPC 3 also disagreed with all the other GPCs.

2.4.2 Time-triggered Protocol (TTP/C)

A databus known as TTP/C was developed for the needs of the emerging automotive "by-wire" industry [4]. Its goals were to provide replica determinism while living within the cost constraints of the automotive marketplace. Because of its low cost, it has also found applications within the aerospace market. TTP/C is a Time Division Multiple Access (TDMA)-based serial communications protocol that provides synchronization in deterministic message communication over dual redundant physical media. TTP/C also 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. 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 a Byzantine failure. The Fault Injection Techniques in the Time-Triggered Architecture (FIT) project confirmed the possibility of this vulnerability by observing actual occurrences of such failures [22].

As part of the FIT project, a first generation time-triggered communication controller (TTP-C1) was radiated with heavy ions. [23]. 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 within the TTP-C1 Integrated Circuit (IC). ICs with improved design are now available for TTP/C.

During the many thousands of fault injection runs, several system failures due to Byzantine faults were recorded [23]. The dominant Byzantine failure mode observed was due to marginal transmission timing. Corruptions in the time-base registers, within the integrated IC that had been irradiated, 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 ICs of the system having slightly fast clocks; ICs with slightly slower clocks rejected the message. Even though such a timing failure would have been tolerated by TTP/C's Byzantine-tolerant clock synchronization algorithm [24], 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 ICs that had different perceptions of this transmission's validity from communicating with each other. Therefore, following such a faulty transmission, the system is partitioned into two sets or cliques — one clique containing the ICs that accepted the erroneous transmission, the other clique containing the ICs 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 IC prior to its next scheduled message. ICs that find themselves in a minority clique (i.e. unable to receive messages from the majority of active ICs) are expected to cease operation before transmitting. However, if the faulty IC 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 IC 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 IC 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 [25]. Subsequent generations of the TTP/C controller have incorporated parity and other mechanisms to reduce the influence of random upsets (e.g. ROM based microcode execution). SOS faults in TTP/C can be mitigated with a central guardian. This guardian assumes fail-detectable behavior and does not violate end-to-end CRC arguments, which has to be shown by exhaustive testing.

2.4.3 Mid-Value Select

Mid-Value select is a well-known method for masking the propagation of failures. It has properties that are similar to an M-out-of-N voter [26], but does not require any of its inputs to be bit-for-bit identical. Many mid-value select implementations in actual fielded systems are merged with other fault tolerance mechanisms such as reasonableness checks or other fault detection mechanisms that are then used to block some inputs to the mid-value selection if they are known to be bad via these other fault detection mechanisms. This blocking of inputs makes these types of mid-value selection mechanisms a form of hybrid n-Modular Redundant (nMR). Hybrid nMR systems can change the "M" and/or "N" in the M-out-of-N calculations by using reconfiguration. The objective is to be able to tolerate more faults than can be tolerated by using nMR alone (details described in example below). To make things even more complex, these additional fault detection mechanisms sometimes use a previous output of the mid-value select in comparison with its inputs to determine if they are faulty and/or use a previous output of the mid-value select as the replacement value for a detected faulty input.

One of the most common ways of blocking inputs to a mid-value selector is to override the value of a detected faulty input with the value that is the midpoint of the reasonable range of input values. This is illustrated on the left side of Figure 13, which was taken from a paper by Stephen Osder [27] and modified slightly. The "voter" in his figure is actually a mid-value selector, which due to its similarity with a bit-for-bit M-out-of-N voter can be called a voter. While Osder uses this example to show how this widely used mid-value selection design can fail to meet its design purpose for a non-replicated mid-value selector, we can use the same design to show how disagreement can arrive in replicated mid-value selectors.

In this example, zero input volts (ground) is assumed to be the middle of the valid range. The switches in this figure are driven by the fault detection logic and force faulty inputs to ground. The fault detection mechanism used in this example is one that is commonly used. It compares the previous output of the mid-value selection with each of the inputs. If an input value is more than a certain epsilon away from this last value, it gets switched to zero. The rationale for this design is that the mid-value selector could only tolerate one failure (worst case) if the switches were not included. With the switches (and assuming good enough failure detection circuitry that controls the switches), this design hopes to tolerate an additional failure using the following argument — When the first failure is detected, it is clamped to zero. If a second failure occurs that is further away from zero than the good value, the good value is selected by the mid-value selector. If the bad value is between the good value and ground, the worst value that the mid-value selection can output is ground, which is the midpoint of the reasonable input range. Sometimes this is sufficient. When it is not, a common variation of this idea is to use the previous output of the mid-value

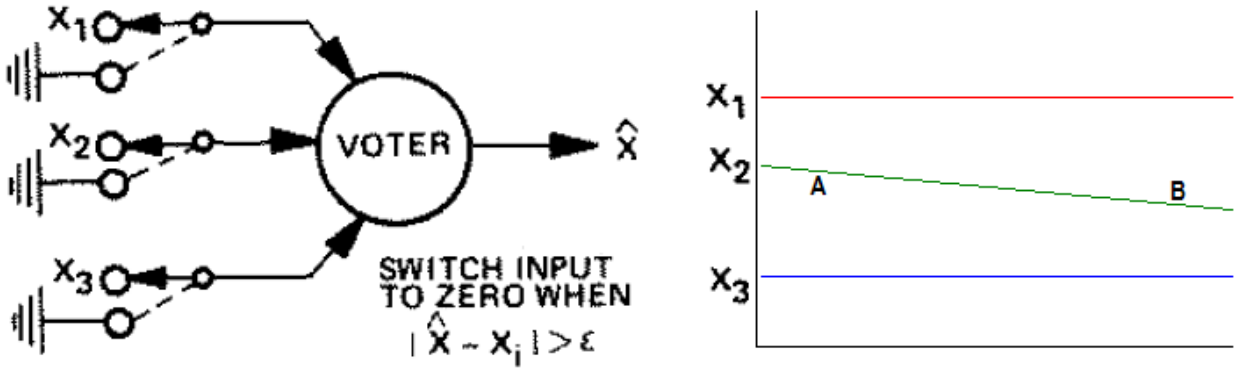


Figure 13. Mid-value select failure example

selection instead of the midpoint of the reasonable input range (ground in this example).

Mid-value select is used most advantageously where bit-for-bit M-out-of-N voters cannot be used due to the system's inability to ensure that the inputs are bit-for-bit identical. Most often this is due to the inputs being asynchronous with respect to each other. However, it is this very asynchrony coupled with these hybrid nMR additions to mid-value select that can still lead to system disagreement.

Going back to the Osder example, but using replicated mid-value selectors, the following scenario, as depicted in the simple plot on the right side of Figure 13, is possible — Two of the three signals are on either side of the signal that is currently being selected as the mid-value and are nearly epsilon away from this mid-value. Given that this middle input value is sampled asynchronously and is varying to some degree, one of the mid-value selectors could sample it when it was closer to the more positive of the other two inputs (i.e., X_2 is sampled near point A) and another mid-value selector could sample it when it was closer to the more negative of the two other signals (i.e., X_2 is sampled near point B). The former mid-value selector will then block the more negative of the other two inputs and the latter mid-value select will block the more positive of the other two inputs. We now have a disagreement between these two mid-value selectors. Even more perversely, a third mid-value selector could sample the middle input when it's exactly between the two other imports and not see either one of them as being an epsilon away. Thus, we get a three-way split: one blocking the positive input, one blocking the negative input, and one not blocking any inputs. The conditions for sustaining a three-way split are highly unlikely. However, a two-way split would be persistent. In this persistent state, the replicated mid-value selectors may initially select the same input (e.g. X_2 from the right side of Figure 13) but eventually select different inputs. For example, if X_2 continues on its downward slope and eventually becomes lower than X_3, any replicated mid-value selector that has blocked X_1 will select X_3 as the mid-value; while, any replicated mid-value selector that has blocked X_3 will select (the grounded) X_2 as the mid-value. Thus, this condition creates a system disagreement.

2.5 Level of Agreement Required

The level of agreement required among replicas (or conversely, the maximum amount of disagreement that can be allowed) depends on system requirements, both external and derived.

For flight control systems, typical external requirements include limiting actuator force fight and minimizing divergence among different information outputs (e.g. differences between pilot and copilot displays). The limits of force fight can be required to minimize wear on actuator components

and/or to minimize wasted energy. Note that a force fight can occur within an actuator or well "downstream" of the actuator, e.g. aero surfaces can cause a force fight in the air stream. Force fights caused by momentary disagreements or small amplitude disagreements seldom cause any problem. It is usually the accumulation of stress or wasted energy that requirements are put in place to avoid. And, this is why fault detection mechanisms for force fights typically include a time-amplitude integrator that is compared against some limit.

There are applications other then flight control systems that can have a much more stringent level of agreement required. For example, spacecraft systems include pyrotechnic initiators that have stringent requirements on timing between replicant signals and "escapes" cannot be allowed because an erroneous command to fire cannot be undone. In the area of automotive X-by-wire systems, requirements for reaction times may be much more stringent than aircraft flight control.

Typically, derived requirements are created to ensure that replicant outputs are close enough (in time and amplitude) that they do not exceed the capability of voters, compares, and selectors that are used for masking; or to prevent output "bumps" when switch-overs to backup replicants occur in active/standby systems. The latter could also be seen as an external requirement because the concern about "bumps" is effect on the system's external environment.

There are complexity trade-offs in these derived requirements. Typically, the better the system does at minimizing disagreement, the simpler the mechanisms need to be for comparison, selection, or voting. Simplicity affects not only size, weight, and power of an implemented system, but it also affects the ability to reason about the correctness of its design.

A common trade-off to be addressed in the design of fault-tolerant system is the degree of rigidity in the fault tolerance mechanism. As a design becomes more rigid in the enforcement of replica determinism, the less uncertainty "dirt" is passed downstream that may require complicated fault tolerance mechanisms to handle. On the other hand, enforcement of replica determinism could become too rigid – that is it becomes "brittle" where instead of bending it breaks. A typical brittle system is one that does an all-or-nothing exchange. The benefit of this rigidity is that all recipients of the exchange are guaranteed that if they receive any input (without being flagged as bad) all other recipients get exactly the same input. The cost of this rigidity is that even a minor transient can cause the loss of data that could have been useful. Possibly even more brittle are systems with a policy of "one strike and you're out" where a simple transient upset can cause a member of the replica set to be permanently voted out of membership.

3 Agreement in Asymmetric Benign Failure Systems

Algorithms for reaching agreement in the presence of Byzantine faults are well known. But, they are expensive to implement, particularly in their demand for communication bandwidth. An example of the large increase in bandwidth required to tolerate Byzantine faults using the well-known algorithms is shown in Figure 14. The numbers used in this figure assumes that digital signatures are not employed. With digital signatures, the number of "Generals" can be reduced from $3f + 1$ to $2f + 1$. However, the successful use of digital signatures requires that the signatures cannot be forged and there is no known means to prove that digital signatures cannot be forged, given the level of assurance required for highly dependable systems (if system dependability rests solely on the success of these signature schemes). The bandwidth is broken out into the amount of traffic that must be transmitted vs. received. This is because the sum of these two is a better indicator of how much work the system must do to handle this traffic rather than the amount of work that could be inferred from just looking at the traffic on the media. Most designers of highly dependable systems would be shocked to find that this required amount of work is nearly two orders of magnitude larger

than they would expect.

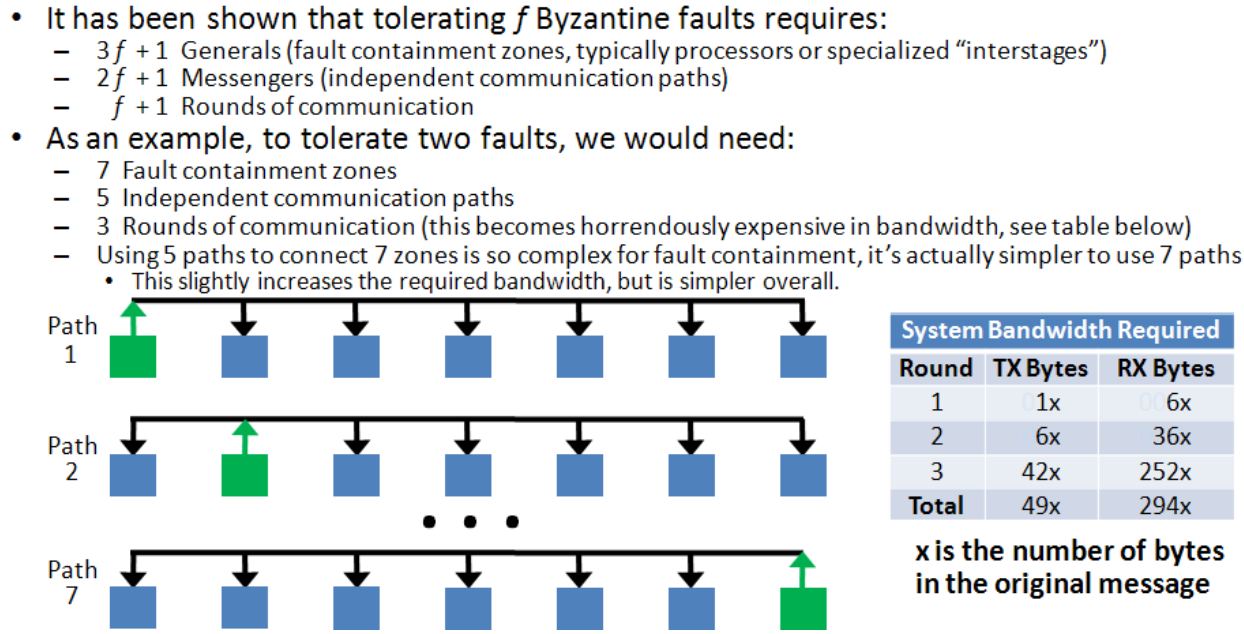


Figure 14. Bandwidth required for Byzantine agreement

Reaching agreement in the presence of Byzantine faults can be made simpler if some key components have their failure modes constrained to be (asymmetrically) benign. The definition of benign used here is taken from Schmid, Weiss, and Rushby [28]. This definition includes manifest faults (those that are detectable as obviously bad) and omission faults (failures to send any value). By extension, an asymmetric benign faulty device is defined as one that can send a correct value, an obviously bad value, or no value arbitrarily among differing observers.

While some designs naively assume that components naturally fail only benignly, this assumption is not supportable for highly dependable systems. Restricting failure behavior to be benign or asymmetrically benign must be enforced. One method for this enforcement is to replicate each of the key components into a subset, where each such subset provides the required constraint on failure behavior. The creation of these subsets forms a hierarchy for Byzantine agreement.

4 Hierarchical Byzantine Agreement

Nearly all of the Byzantine fault tolerant agreement strategies published in the literature assume a flat peer-to-peer communication model. This assumption does not necessarily mean that all the parties are connected with point-to-point links. But, it does mean that the algorithms/mechanisms used to tolerate Byzantine faults attempt to establish agreement among all of the participants as a whole set. A contrasting approach is to first reach agreement within small subsets, for example pairs, of the participants and then reach agreement among the subsets. One benefit of the latter approach is greatly reduced bandwidth. The following paragraphs describe Hierarchical Byzantine Agreement and how this bandwidth is saved.

To reach agreement in Byzantine fault scenarios, multiple rounds of communication exchange are required. According to the well-developed understanding of Byzantine fault tolerance, an additional round of exchange is needed for every additional Byzantine fault to be tolerated. In nearly all of the

published algorithms, each of the rounds of exchange requires retransmitting all the data received in the previous round. A reduced representation of a data message (e.g. a hash) could be sent out during all but the first round of exchanges. However, using a reduced representation would lead to reduced coverage for failures. The extreme case makes this point obvious—if the hash was simply parity, half of all failures would not be detected. Note that this hash should not be confused with the digital signatures referenced above. These two mechanisms serve totally different purposes. The hash would be used to reduce the size of messages in subsequent rounds of communication; whereas, digital signatures are used to reduce the number of fault containment zones required.

In hierarchical Byzantine agreement, each subset of receivers agree among themselves whether they have all received a message correctly. If they have received the message correctly, they need only send out a single status bit that indicates this during the next round of exchange. Thus, subsequent rounds of communication that would require entire messages to be exchanged in the well-known Byzantine algorithms are now reduced to a single status bit that indicates whether the entire message was received correctly or not. One could construct a hierarchical Byzantine agreement scheme that relaxes this all-or-nothing agreement, but that is outside the scope of this report.

Another variation of hierarchical Byzantine agreement does not explicitly exchange status data. Instead, the behavior of the subset as a whole is governed by the (implicitly) shared subset status. For example, if the members of a subset don't agree on the reception status of a message sent into the subset, the subset may not perform their normal function of relaying that message to other subsets. The next section of this report describes how TTEthernet uses this variation of hierarchical Byzantine agreement.

The first known use of hierarchical Byzantine agreement in a fielded system was the ARINC 659 (SAFEbus) [29]. In this case, the subsets were self-checking pairs. The Honeywell-designed SAFEbus interface hardware contains mechanisms that allow the halves of a pair to agree on whether they both received a message correctly or not, and also to automatically transmit a message containing the status of this agreement for other pairs to see. The agreement within a pair is achieved by employing a syndrome exchange between the halves of the pair. This exchange is transferred via local short-run wiring within the pair. Using this local wiring for the exchange eliminates the need for this agreement traffic to be carried on the SAFEbus itself.

The status exchange between pairs forms the upper layer of the hierarchical Byzantine agreement mechanism, which allows all the pairs within a SAFEbus system to reach agreement. In this case, the size of the status is a 32-bit word, the smallest message that SAFEbus can send. Because the SAFEbus self-checking pairs include self-checking drivers and paired media [30], its failure semantics are constrained to be the asymmetric benign fault class. As with all statements that limit faulty behavior to a particular class, there is a non-zero probability for occurrence of some faulty behavior outside this class [12]. Whenever such statements are made in support of arguing that a system meets dependability requirements, analysis has to be done to show that these probabilities are sufficiently small (i.e., the total of all probabilities for ignored failure modes added to other failure probabilities must not exceed system requirements)[3].

In the case of SAFEbus, a failure can "escape" if both halves of a self-checking pair fail in such a way as to produce identical outputs. Because the halves are independent, this escape would require two failures. It can be shown that the probability of two failures within one self-checking pair is well below the maximum allowed probability.

Thus, the SAFEbus hierarchical Byzantine agreement saves bandwidth costs in two ways. First, the communication between halves of a pair uses a very inexpensive short-run private communi-

[3] an essential argument that many architecture analyses omit

cation path that consumes no bandwidth on the main SAFEbus communication media. Second, the second round of exchange sends messages that consist of only a single word instead of fully replicating all of the data from the first round of exchange.

Another benefit of hierarchical Byzantine agreement is its ability to tolerate some multiple Byzantine faults while using less hardware than most approaches reported in the literature. In the case of SAFEbus, multiple Byzantine faults can be tolerated as long as no two faults are located within the same pair, produce identical symptoms, and occur at the same time. Given the fault zone independence between the two halves of a pair, the probability of such a dual failure occurring can be calculated and found to be much less than the maximum allowed by system requirements. A SAFEbus system can tolerate any number of other Byzantine faults within pairs[4] simply by adding another pair for each fault to be tolerated.

The two benefits of hierarchical Byzantine agreement augment each other to give a surprising amount of cost saving. As an example, compare SAFEbus against the two-fault scenario given in Figure 14. In this scenario, SAFEbus would require only three self-checking pairs, two rounds of exchange, transmit bandwidth of 4x+8 bytes, and receive bandwidth of 8x+16 bytes[5]. For this tremendous savings, SAFEbus gives up only the ability to tolerate two simultaneous identical Byzantine faults within a pair.

5 TTEthernet Protocol Agreement Mechanisms

5.1 TTEthernet Overview

TTEthernet provides time-triggered services that enable time-triggered communication over Ethernets that use active star (e.g., hub or switch) topologies. These time-triggered services establish and maintain a global time via a fault-tolerant, self-stabilizing synchronization strategy, which helps to establish temporal partitioning and ensures isolation of the synchronous time-critical data-flows from other asynchronous Ethernet data-flows. The main components of a TTEthernet system are switches and End Systems (ESs), as depicted in Figure 15.

5.1.1 TTEthernet Protocol Synchronization

TTEthernet uses a two-step approach to synchronization. In the first step, Synchronization Masters send protocol control frames to compression Masters. The Compression Masters then calculate a fault-tolerant average synchronization value from the relative arrival times of these protocol control frames and send out a new protocol control frame back to the synchronization masters and to other synchronization clients.

The Synchronization Masters and Compression Masters can be assigned to devices in any manner that meets the requirements for the system. The typical assignment is for (some subset of) the end systems to host the Synchronization Masters and for the switches to host the Compression Masters.

End systems exchange application data with each other by transmitting standard Ethernet frames. The points in time when end systems dispatch these frames can be coupled to the synchronized time. The transfer of these frames is then called time-triggered transfer, because the trigger

[4]The SAFEbus drivers and media are not similarly expandable to tolerate additional faults. However, their greater simplicity reduces their probability of generating Byzantine faults. If greater media fault tolerance were ever needed, the ARINC 659 document could be changed to allow additional lanes of media. This would be a simple change to the interface hardware.

[5]These numbers are all a multiple of four because SAFEbus uses quadruply redundant media.

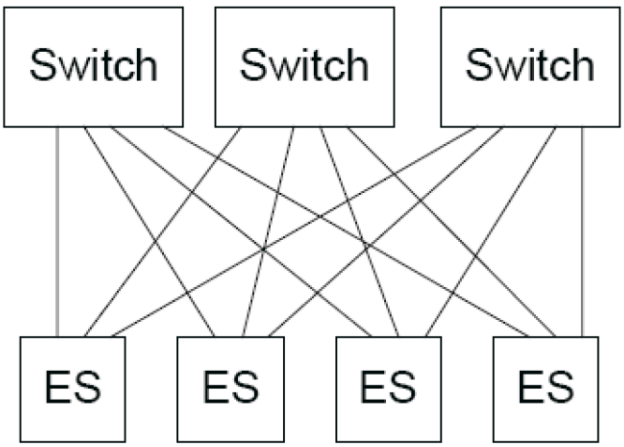


Figure 15. Typical TTEthernet System

for frame dispatch is derived from time. TTEthernet formally defines the relationship between the synchronized time and the time-triggered transfer.

TTEthernet supports the design of communication systems with mixed time-criticality in which several applications of mixed time-criticality share a single physical network. In particular, an Ethernet network can be used to transfer frames in a time-triggered mode (synchronous communication) and non-time-triggered modes (asynchronous communication as for example Ethernet frames transmitted according to the best-effort strategy). The TTEthernet synchronization strategy inherently compensates for latency and jitter resulting from this integration and ensures high-quality synchronization despite increased network latency and jitter. Synchronized time provides the foundation for partitioning and isolation of critical applications from the less critical or non-critical ones.

Offline scheduling tools can be used to guarantee that time-triggered transfers are conflict-free (i.e., no two time-triggered frames compete for transmission). Frames for time-triggered communication are not priority-based amongst themselves. Time-triggered media access is driven only by time progression and frame schedule. This prevents network traffic congestion for time-triggered frames and enables communication with fixed latency in Ethernet networks, unaffected by any asynchronous Ethernet traffic load.

TTEthernet provides two classes of service that are not time-triggered (TT). These classes are Rate Constrained (RC) and Best Effort (BE). Both of these classes run at lower priority than the TT traffic.

RC traffic establishes periodic communication with maximum bandwidth use to ensure bounded latency in complex networks. Successive transfers of RC frames belonging to the same rate-constrained data-flow are guaranteed to be offset by a minimum duration as specified offline. This is in contrast to TT communication, which in addition to the minimum duration also specifies a maximum duration for this interval. The RC communications paradigm is specified in ARINC 664 standard part 7 [31]. A system designer may decide to use RC transfers when determinism and real-time operating requirements are less strict than those that drive the use of TT communication. RC transfers guarantee sufficient bandwidth allocation for each transmission, with defined limits for delays and temporal deviations. In contrast to TT transfers, RC transfers are not dispatched with respect to a system-wide synchronized time base. Hence, different communication controllers may dispatch RC-transferred frames at the same point in time. Consequently, the RC-transferred frames may queue up in the network switches, leading to increased transmission jitter and requiring increased buffer space. Given the known transmission rate of the RC transfers and the network

switch controls, it is possible to avoid frame loss by calculating the transmission latency offline.

BE transfers implement the classic Ethernet approach. There is no guarantee if and when these frames will be transmitted, what delays may occur, or if BE-transferred frames will arrive at the recipient location. BE transfers use the remaining bandwidth of the network and have lower priority than TT and RC transfers.

5.1.2 TTEthernet Fault Tolerance

TTEthernet networks can be set up with redundant end systems and switches. TTEthernet allows the integration of guardians in switches and end systems. Guardians check if the communication on the network works in compliance with the predefined parameters. If faulty components disrupt communications, the guardian disconnects the network media segment or port. The typical Guardian function is implemented by using a self-checking pair or Command/Monitor (COM/MON).

TTEthernet can tolerate asymmetric benign faults. When configured to a multi-master mode, TTEthernet tolerates an asymmetric benign faulty communication path and even an asymmetric benign faulty end system at the same point in time. This failure mode means that each faulty device can arbitrarily drop frames on any of its incoming communication links and on any of its outgoing communication links with potential inconsistent dropping behavior for each frame.

TTEthernet can tolerate end system failures because their faults are contained at the switches. The switches in TTEthernet can be configured to execute a central bus guardian function. The central bus guardian function guarantees that even if a set of end systems becomes faulty, the system-wide impact of these faulty end systems is masked. TTEthernet switches establish fault containment boundaries.

TTEthernet also provides self-stabilization properties, i.e., the synchronization will be re-established even after transient upsets in a multitude of devices in the distributed computer system. TTEthernet stabilizes from an arbitrary system state to a synchronized system state. This self-stabilizing property becomes more and more important with decreasing feature sizes of computer chips and, therefore, resulting increase in transient upsets.

5.2 Asymmetric Benign Enforcement in TTEthernet

TTEthernet can constrain the failure behavior of its key components to be within the asymmetric benign fault model through the use self-checking pairs for its switches, and possibly its end systems as well. However, the first variant of TTEthernet to be developed (TT-GbE, which uses the IEEE 802.3 1000BASE-CX physical layer) does not use self-checking pair drivers or media. It was felt that the failure mechanisms of wire were well enough understood that the use of 8B/10B encoding plus the standard Ethernet Cyclic Redundancy Check (CRC) provided sufficient detection coverage for errors that occurred on the media. Because the drivers and media are not self-checking pairs, paired signals cannot be brought out from switches or end systems. Thus, even though switches and end systems may have self-checking pair electronics, their behavior is more aligned with that of COM/MON architectures (albeit, a very high coverage COM/MON).

The lack of self-checking pair drivers meant that a new mechanism for self-checking pair fault containment had to be invented. Single-string drivers are out of the question because they were not simple enough to reason about any possible constraint on their failure behavior. A mechanism was created (and patented) that has one half of the self-checking pair driving the media and the other half being able to "crowbar" any output that it disagrees with. This action would convert any faulty frame into a manifest failure by causing all receivers to see an illegal symbol before the end of the frame. A conceptual diagram of the self-checking pair is shown in Figure 16. In

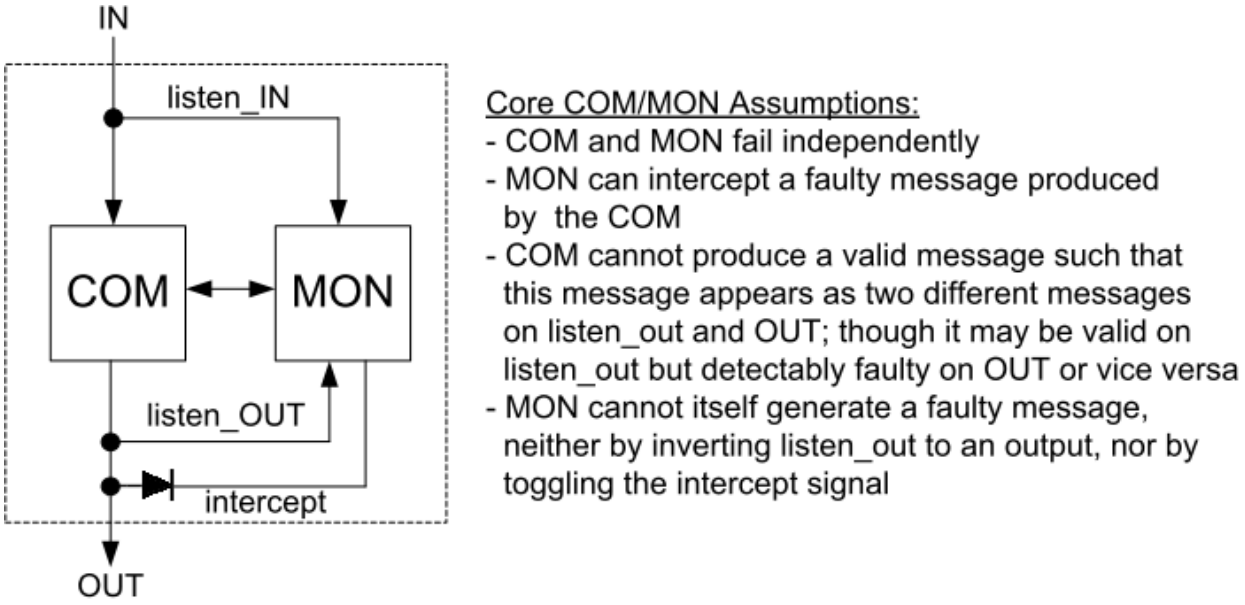


Figure 16. Conceptual Diagram of a TTEthernet Self-Checking Pair

this simplified diagram, the "crowbar" is the diode that the Monitor (MON) can use to cause the output signal to be shorted to ground if it sees that listen_OUT is faulty. When the output is not faulty, the diode is reverse biased and has negligible effect on the output signal. Note that there is a direct private communication link between Command (COM) and MON. This link is used for the local exchange as part of the first level in a hierarchical Byzantine agreement structure. Thus, it serves a purpose similar to the syndrome exchange of SAFEbus. This exchange protects the pair from being split (each half having different internal state) due to a Byzantine failure outside of its fault zone.

5.3 TTEthernet Startup and Synchronization Agreement

With this implementation of TTEthernet being constrained to only asymmetric benign faults, relatively simple startup and synchronization protocols can be used with high confidence. Startup consists of any one of a designated class of nodes sending out a unique coldstart frame. Given that switches can only be benignly faulty, this coldstart frame cannot be successfully forged in a way that can adversely affect the protocol. All good end systems that hear a coldstart frame acknowledge that frame and the time line that enforces the time-triggered traffic class begins.

During execution of the TTEthernet timeline, periodic synchronization events are scheduled. In each synchronization event, Synchronization Masters transmit protocol control frames at the time that each of these end systems thinks it is correct to do so. The self-checking pair switches contain the Compression Master function that acts as a majority voter for the synchronization frames. Given that faulty switches are only asymmetrically benign, the only failure behavior they exhibit is a failure to deliver the synchronization frames to all end systems. That is, there are no possible failures affecting integrity, only availability. The required availability is provided by the use of multiple switches. Each end system is directly connected to a number of switches that is sufficient to meet the system's availability requirements.

The TTEthernet standard [1] establishes the algorithms and the quorum criteria required to meet the system fault tolerance requirements for the synchronization protocol.

6 TTEthernet Application Agreement Mechanisms

Protocol agreement is insufficient to maintain overall system agreement; application agreement is also needed. This is particularly true for systems with a communication network that allows asymmetric failures, even benign ones. The omission or the rejection of a manifestly faulty message can cause the state of replicas to diverge. Of course, any application-level agreement will have to be built on top of whatever basic services the communication system provides.

The reasons that application agreement is needed and the degree of divergence allowed is given in section 2.5. To maintain application agreement, some form of state exchange is required.

The design of a particular application agreement mechanism is influenced by the following factors.

- divergence amplitude limits
- divergence recovery time
- bandwidth required
- network dependability
- whether the network is synchronous or asynchronous

6.1 Application-Level Hierarchical Agreement

With TTEthernet providing an asymmetric benign fault model communication network, clients of this network are guaranteed to receive good data or no data. What remains to be resolved is whether all good replicants in a set got the data or not. Using hierarchical Byzantine agreement as described above, applications on a TTEthernet can resolve this question efficiently by sending only a minimum size frame. However, this only works if all the components, including the end systems, enforce a benign fault model. The status returned from any end system that does not enforce a benign fault model cannot be trusted. For these simplex nodes, a full data exchange is required.

6.1.1 Single Source

Single-source state exchange mechanisms such as Leader/Follower and Master/Shadow are particularly well-suited for systems that are constrained to only benign faults. For this to be true, though, hosts as well as the network must be constrained to the benign failure class. When this is achieved, the Leader or Master can broadcast the definitive applications state without broadcasting any errors (i.e., all state that is transferred will be correct, but delivery is not guaranteed).

In the Leader/Follower case, a leader is elected from a set of candidates and holds tenure until it fails. During its tenure, the Leader is the definitive source of state and broadcasts that state with sufficient frequency to meet system requirements. Detection of the Leader's failure is simplified in time-triggered networks. All the Leader needs to do is to transmit a periodic heartbeat. When any of the nodes that are capable of voting for a new leader detect that the heartbeat has failed, they cast a vote for a new leader. If the threshold for electing a new leader is based on the total number of eligible voters rather than the total number of voters participating in any particular election, the voting mechanism itself can mask potential problems with inconsistent observations of the current leader's failure. That is, if only a minority of the eligible voters observes that the heartbeat has failed, they cannot cast enough votes to elect a new leader. Some bandwidth can be saved if the state transfer acts as the heartbeat.

In the Master/Shadow case, the current Master and all shadows form a priority ordered set. The highest priority fault-free participant in that set is the current Master. To save bandwidth in ARINC 659, there are Master/Shadow slots in the schedule that allow up to four nodes to use the same bandwidth. Using a mini-slotting protocol within those slots, the highest priority participant is allowed the first chance to transmit. If it doesn't begin to transmit within it mini-slot, the next higher priority participant attempts to transmit, ... and so on through the fourth priority. For state exchanges, all the participants would send the required state in that slot using the same format regardless of the source. Thus, any receiver needing this state can use the data transferred in this slot without needing to know the source. This can accommodate participants with omission failures, but not manifest failures. A similar thing can be done for TTEthernet, with the additional benefit that these mechanisms on TTEthernet can also tolerate manifest failures.

The way that this can be done for TTEthernet is to have the potential Leaders or Masters (collectively call them contenders) share the same Bandwidth Allocation Gap (BAG) resources. If the current Leader or Master is alive, it consumes the allocation. This consumption can be used to hold off the other contenders. For the Leader/Follower case, the consumption is the heartbeat that is used to hold off the vote for a new Leader. For the Master/Shadow case, the contenders are scheduled to have contiguous transmission windows with the temporal ordering of these windows being the priority for which contender takes over if the current Master fails to consume their shared BAG allocation.

6.1.2 Peer Exchange

While single-source mechanisms usually make the most sense for asymmetric benign systems, there may be situations where they do not. The most likely encountered such situation is when the hosts cannot be constrained to the asymmetric benign model. When single-source mechanisms can't be used, some form of peer exchange is needed. These exchanges will have to follow the more classical Byzantine exchange algorithms. But, it does help that the underlying network is asymmetric benign (at least in the switches). An efficient hardware-assisted exchange mechanism that exploits the asymmetric benign failure property of TTEthernet and hierarchical agreement is given in section 6.3 below.

6.2 Application Startup

Once a system's network is up and working correctly, applications need to do their own startup procedure. Typically, applications have both a "cold start" and "warm start" to establish a valid initial state. Cold start is used when a system is in a safe state. Systems typically exchange a number of independent sources of data that, in the aggregate, indicate the system is in a safe state. For example, a commercial airliner could use a combination of weight on wheels, air speed below some minimum, and door open. A similar rationale applies to warm start. However, warm start procedures tend to be more ad hoc and application-specific than cold start. Regardless of the mechanism used at the application level, some form of agreement is required via the network. Mechanisms for reaching agreement on the exchange of this application data can be built on the TTEthernet agreement, fault tolerance, and synchronization mechanisms described above.

6.3 Efficient Application Data Handling

To provide better throughput, application replication coordination and comparison may be implemented by a printed circuit board of special hardware, such as the self-checking pair host interface hardware used by SAFEbus [32]; alternatively, it may be performed by tasks distributed across

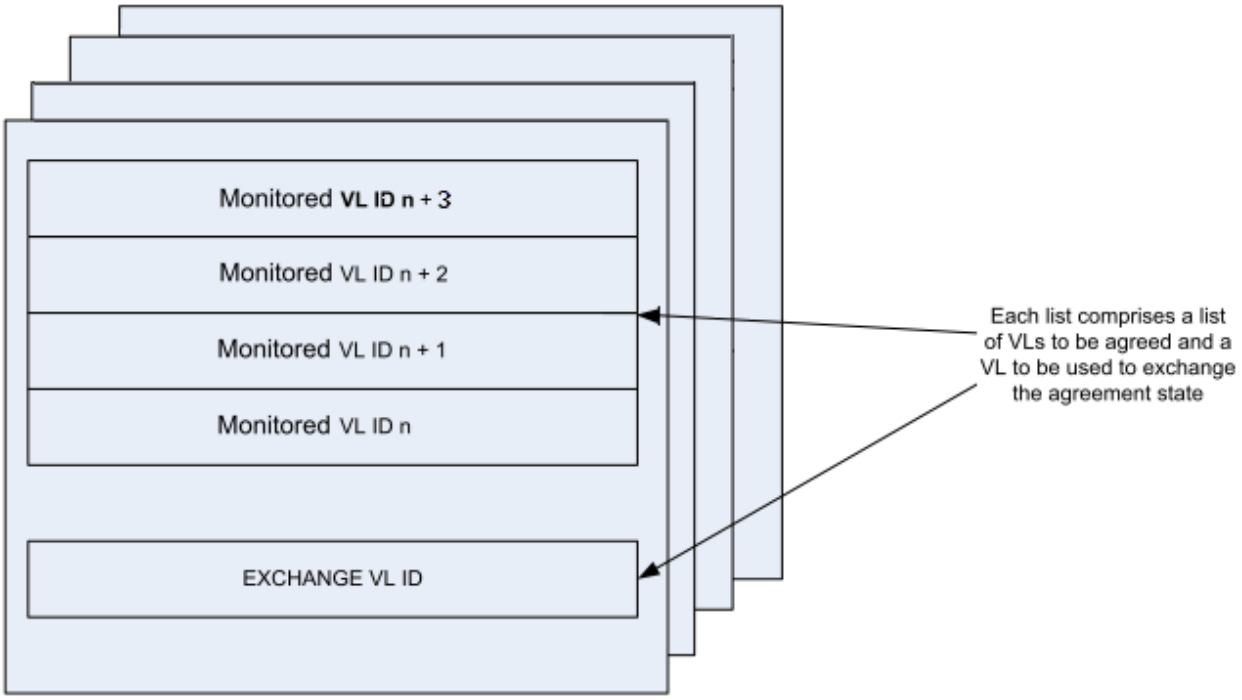


Figure 17. VL Agreement List - Maintained for Each Task

a network. In the latter case, i.e. where the replication is performed over a network, additional steps are required to ensure that the replicated computational task set achieves and maintains the required degree of state consistency necessary to produce identical outputs. This requires that the replicated tasks agree on initial state and all input data that is causal to internal state changes.

For networks without high-integrity, where value correctness cannot be guaranteed, the agreement process requires the retransmission and comparison of all values received by all consumers. Such exchanges can demand significant software and messaging overheads. For high-integrity networking technology, where value correctness can be guaranteed (for example, technologies such as self-checking TTEthernet, SAFEbus, or the BRAIN), the software and messaging overheads can be reduced by requiring agreement only on reception status as part of a hierarchical agreement structure. However, even with this reduction, agreement may still constitute a significant software overhead, if the agreement is to be performed in software and the agreement exchanges need to meet typical avionics real-time requirements. For this reason, it is advantageous for agreement exchanges to be implemented primarily in hardware, with minimal software involvement.

Hardware-assisted application data agreement can be done with an ingress (received frames) agreement scheme that utilizes the determinism of time-triggered frame exchange, although asynchronous variants may also be possible. For each replicated set of tasks, the network hardware within each node maintains a list of the Virtual Links (VLs), TTEthernet's frame identifiers, that require ingress agreement among the tasks. Each frame agreement list also has a dedicated VL or VLs to perform the ingress agreement exchange. See Figure 17.

As frames are received by an ES, they are checked against each of the agreement frame lists. If the frame is found on an agreement list, the receiving host adds the reception status to the exchange VL's frame buffer. As additional frames are received, each status is also written to the exchange buffer. The precise organization of the buffer is implementation specific. However, an example simple mapping is shown in Figure 18. In this example, the index of the frame ID in the

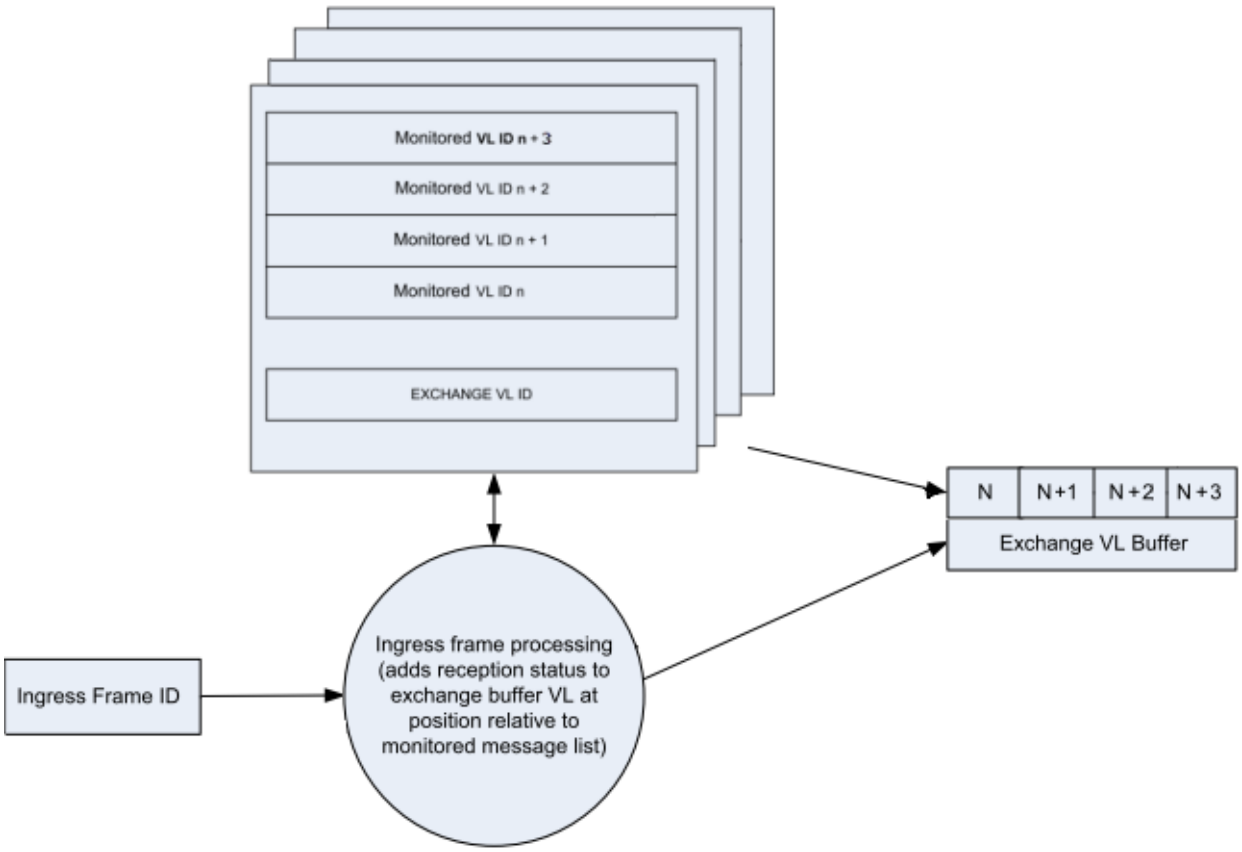


Figure 18. Ingress Frame Processing and Exchange Buffer Building

VL list is used as an index into the assigned exchange VL buffer space. The reception status word is written at the indexed location.

If the replicated system data flow is time-triggered, then in accordance with the time-triggered schedule, it can be determined what time all ingress frames should have arrived for the task set. Following this point, as shown in Figure 19, the content of the exchange buffer is transmitted. This frames is routed such that it will arrive at the other ESs that are replicating the associated task.

On reception of the exchange frame, each receiving node compares the status of the remote reception with its local reception status for each frame of the agreement list. If the status for both local and remote host indicate that the frame has been received OK, then the associated frame is marked with an agreed status. If either of the local or remote frame reception status indicate that the frame was not received, the frame is marked with non-agreed status. The agreed status is made available to all data consumers. This agreed status is then signaled as part of each frame's fresh data indication to the hosts. Using this additional status, the hosts are able to identify which data has been received by all of the replicates.

The time-triggered schedule is also used to clear frames status at a known point within the schedule to allow the frame status processing to resume in the next cycle. In practice, this may be implemented in conjunction with high-level buffer processing, such as the maintenance of ping-pong buffer schemes.

The above mechanism enables software to simply identify which data was received by all parties of the replicated task set. Using this information, the software can decide which data to use for replica-determinate calculations and, if necessary, substitute "safe-defaults" to mitigate missing

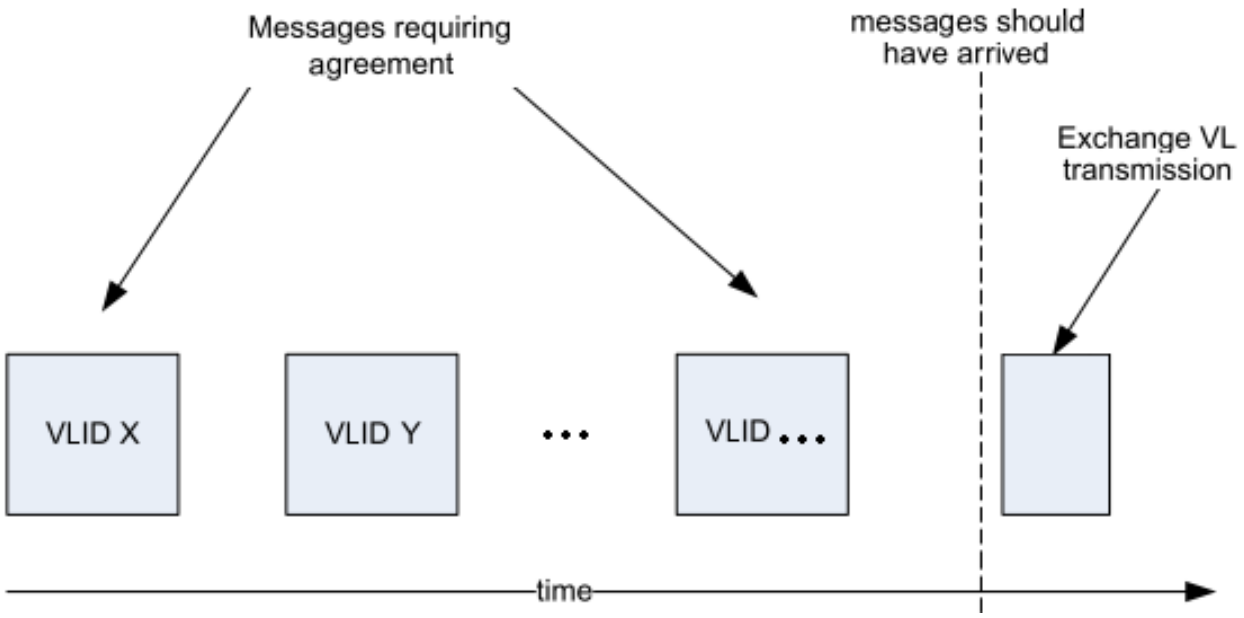


Figure 19. Scheduled Exchange Frame

data.

However, such decisions can still constitute some software overhead, since it requires separate reads and decision logic to be applied to each received frame. For this reason, the buffer status is efficiently summarized at the host interface as depicted in Figure 20. This is a single contiguous encoding for the summary status for all ingress frames required by the task set.

For each of the replicated tasks, a summary of the agreed state is presented in a packed data word or words. Using this summary information, it is possible for the replicated tasks to implement some of this default value selection logic as a table look-up operation (i.e., depending on what values are valid, the software may branch efficiently to alternative programmed logic that is mapped to the available data).

In IMA Systems, an active/shadow replication function is often utilized to optimize system performance. For example, in the AIMS of the B777, active and shadow nodes share SAFEbus messaging slots to conserve network bandwidth. In such configurations, the mechanisms presented here may be extended to optimize the buffer allocation associated with active shadow configurations. Nodes receiving frames from active/shadow pairs may use a shared buffer model where both active and shadow frame receptions write into a common buffer space. In such cases, the buffer agreement word would be an OR of the active/shadow frame receptions. Should the active and shadow frame receptions be temporally separated, the second reception may simply overwrite the first reception. In such a case, the exchange word will be simply scheduled to occur following the last scheduled transmission. It is assumed that other mechanisms are present to ensure that the outputs of active and shadow components are replica determinate and are bit-for-bit identical. The exchange mechanism presented here forms one potential agreement mechanism.

In some applications, it may be permissible to drop a cycle of processing, where reverting to a previous cycle's state vector may be preferable to loosing consistency between replicate task sets. For this reason, an exchange mechanism may be extended to include additional data (i.e., the result of the previous comparison steps, described above). This may be included into the above exchange frames (by adding a previous frame's status field) or by introducing additional exchange

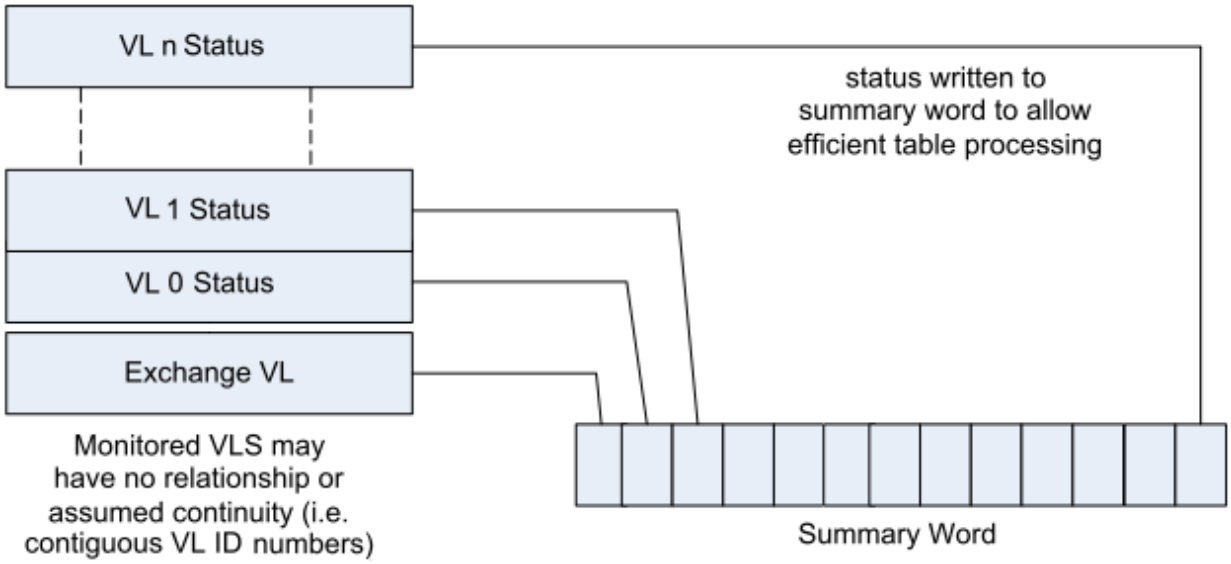


Figure 20. Summary Status Aggregation

frames. A frame counter may also be included to ensure consistency of the signaling. Each half of the pair can then compare the partner's reception status with the local reception status. On detecting a difference, each half may revert to a previous agreed state (i.e., the previous frame or a safe-configured, mode-specific, default state vector). Therefore, this mechanism may mitigate the scenario where one of the exchange frames is lost.

7 Conclusion

This report has discussed the challenges of maintaining application agreement and integration services.

A literature search was presented for previous work in the area of replica determinism. This literature search used both emerging Web search tools and traditional subject-area databases. These tools and sources included Google Ngram Viewer, DEVONagent Pro, TouchGraph, Compendex, Inspec, and NTIS.

Some sources of asymmetric and nondeterministic behavior were described and examples presented that show instances where system-level agreement was lost. These examples included: the complete loss of agreement among National Aeronautics and Space Administration (NASA)'s space shuttle processors for mission STS-124, a persistent loss of synchronization in a Time-triggered Protocol (TTP/C) system due to one of its components being given a high dose of radiation in an experimental setup, and a scenario where redundant copies of a common Mid-Value Select (MVS) design can lead to this agreement among these replicants.

The concept of hierarchical Byzantine agreement was introduced and its benefit of reducing the bandwidth needed for Byzantine agreement was explained.

The synchronization and fault tolerance features of TTEthernet were briefly reviewed and linked to the concepts of enforcing an asymmetric benign fault model and hierarchical Byzantine agreement.

Various methods of exploiting these features of TTEthernet are described for application start up and state agreement. The application-level mechanisms described include application level

hierarchical agreement, single source data exchange, peer data exchange, application startup, and efficient application data handling.

References

1. SAE AS-2 Embedded Computing Systems Committee: Time-Triggered Ethernet. SAE Standards N^o AS6802A, June 2011.

2. Laprie, J.: Dependable Computing and Fault Tolerance: Concepts and terminology. *Proc. 15th IEEE Int. Symp. on Fault-Tolerant Computing*, 1985.

3. Shannon, C. E.; and McCarthy, J., eds.: *Automata Studies*, Princeton University Press, Probabilistic logics and the synthesis of reliable organisms from unreliable components. 1956, pp. 43–98.

4. Kopetz, H.; and Grünsteidl, G.: TTP–A time triggered protocol for automotive applications. Inst. für Technische Informatik, Technische Universit, 1992.

5. Jackson, J.: Google: 129 Million Different Books Have Been Published. August 6 2010. URL `http://www.pcworld.com/article/202803/google_129_million_different_books_have_been_published.html`, accessed Jun 20,2012.

6. Google Books. URL `http://en.wikipedia.org/wiki/Google_Books`, accessed June 20, 2012.

7. NASDAQ Composite Index 1994-2008. 2012. URL `http://en.wikipedia.org/wiki/File:Nasdaq2.png`.

8. Lamport, L.: The Part-Time Parliament. Digital Equipment Corporation Systems Research Center, 1989.

9. Lamport, L.: The Part-Time Parliament. *ACM Transactions on Computer Systems*, vol. 16, no. 2, 1998, pp. 133–169.

10. Edwards, C., ed.: *Fault Tolerant Flight Control*, LNCIS 299, Springer-Verlag, 1 - Introduction. 2010, pp. 3–45.

11. Wenbing, Z.: Byzantine fault tolerance for nondeterministic applications. *2007 IEEE International Symposium on Dependable, Autonomic and Secure Computing, 108-15, 2007*, 2007.

12. Poledna, S.: Replica Determinism in Distributed Real-Time. *Real-Time Systems*, Kluwer Academic Publishers, vol. 6(3), May 1994, pp. 289–316.

13. Guerraoui, R.; and Schiper, A.: Consensus: The Big misunderstanding. *Proceedings of the 6th IEEE Computer Society Workshop on Future Trends in Distributed Computing Systems (FTDCS-6)*, 1997, pp. 183–188.

14. Cipra, B.: How number theory got the best of the Pentium chip. *Science*, vol. 267 (5195), 1995.

15. URL `http://aws.amazon.com/ec2/`, accessed June 20, 2012.

16. Patterson, D.: Why latency lags bandwidth, and what it means to computing. 2004. URL `http://www.ll.mit.edu/HPEC/agendas/proc04/invited/patterson_keynote.pdf`, accessed June 20, 2012.

17. Davidson, S. B.; Garcia-Molinq, H.; and Skeen, D.: Consistency in Partitioned Networks. *ACM Computing Surveys*, vol. 17, no. 3, 1985.

18. Poledna, S.: *Fault-Tolerant Real-Time Systems: The Problem of Replica Determinism*. Kluwer Academic Publishers, 1996.

19. Driscoll, K.; Hall, B.; Paulitsch, M.; Zumsteg, P.; and Sivencrona, H.: The Real Byzantine Generals. *Proceedings of the Digital Avionics Systems Conference*, 2004, pp. 61–71.

20. Bergin, C.: Faulty MDM Removed. May 18 2008. URL `http://www.nasaspaceflight.com/2008/05/sts-124-frr-debate-outstanding-issues-faulty-mdm-removed`.

21. Bergin, C.: STS-126: Super Smooth Endeavor Easing through the Countdown. NASA Spaceflight.com, November 13 2008. URL `http://www.nasaspaceflight.com/2008/11/sts-126-endeavour-easing-through-countdown`.

22. Gruenbacher, H.: Fault Injection for TTA. Deliverable 5.1-5.5 Combined Report IST 1999 10748, Carinthia Tech Institute, 2002. URL `http://www3.cti.ac.at/fit/`.

23. Sivencrona, H.; Johannessen, P.; Persson, M.; and Torin, J.: Heavy-ion Fault Injection in the Time-triggered Communication Protocol. *Proc. 1st Latin American Symposium on Dependable Computing LNCS 2847*, 2003, pp. pp. 69–80.

24. Pfeifer, H.; Schwier, D.; and von Henke, F. W.: Formal Verification for Time Triggered Clock Synchronization. *Proc. 7th IFIP International Working Conference on Dependable Computing for Critical Applications*, 1999.

25. Ademaj, A.: Slightly-Off-Specification Failures in the Time Triggered Architecture. *7th IEEE Int. Workshop on High Level Design Validation and Test*, 2002.

26. Osder, S.: Chronological Overview of Past Avionic Flight Control System Reliability in Military and Commercial Operations. *AGARD-AG-224*, P. R. Kurzhals, ed., NATO Research and Technology Organisation, vol. 224, Jan 1977, pp. 2–1–2–17. Available from NTIS HC A16/MF A01.

27. Osder, S. S.: Generic Faults and Architecture Design Considerations in Flight Critical Systems. *AIAA Journal Of Guidance*, vol. 6, no. 2, March–April 1983, pp. 65–71.

28. Schmid, U.; Weiss, B.; and Rushby, J.: Formally Verifed Byzantine Agreement in Presence of Link Faults. *Proceedings of the 22nd International Conference on Distributed Computing Systems*, IEEE, IEEE Computer Society Press, 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, 2002.

29. ARINC: *ARINC Specification 659: Backplane Data Bus*. Aeronautical Radio, Inc, Dec 1993.

30. Driscoll, K. R.; and Hoyme, K. P.: Apparatus and Method for Transmitting Information Between Dual Redundant Components Utilizing Four Signal Paths. United States Patent N^o 5386424, 1995.

31. ARINC: *ARINC Specification 664P7: Aircraft Data Network, Part 7, Avionics Full Duplex Switched Ethernet (AFDX) Network*. Aeronautical Radio, Inc, Jun 2005.

32. Bauer, G.; Bilic, K.; Driscoll, K.; Salloum, C. E.; Eles, P.; Elmenreich, W.; Goller, A.; Hall, B.; Kammerer, R.; Kantz, H.; Kopetz, H.; Obermaisser, R.; Paulitsch, M.; Pop, P.; Pop, T.; Scherrer, C.; Schmidt, E.; and Steiner, W.: *Time-Triggered Communication*. Embedded Systems Series, CRC Press, Taylor & Francis Group, 2011.

Appendix A

Acronyms and Initialisms

BAG Bandwidth Allocation Gap

BE Best Effort

BFT Byzantine Fault Tolerant

COM Command

COM/MON Command/Monitor

CRC Cyclic Redundancy Check

ES End System

FCS Flight Critical Systems

FDIR Fault Detection, Isolation, and Recovery

FIT Fault Injection Techniques in the Time-Triggered Architecture

GPC General Purpose Computer

HPC High Performance Computing

IC Integrated Circuit

ITU International Telecommunication Union

MDM Multiplexer Demultiplexer

MON Monitor

MVS Mid-Value Select

NASA National Aeronautics and Space Administration

nMR n-Modular Redundant

RC Rate Constrained

STS Space Transportation System

TDMA Time Division Multiple Access

TTP/C Time-triggered Protocol

TT time-triggered

VL Virtual Link

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
01- 02 - 2013	Contractor Report	03/2012 - 12/2012

4. TITLE AND SUBTITLE

Application Agreement and Integration Services

5a. CONTRACT NUMBER

NNL10AB32T

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Driscoll, Kevin R.; Hall, Brendan; Schweiker, Kevin

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

534723.02.02.07.30

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

NASA Langley Research Center
Hampton, Virginia 23681

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSOR/MONITOR'S ACRONYM(S)

NASA

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

NASA/CR-2013-217963

12. DISTRIBUTION/AVAILABILITY STATEMENT

Unclassified - Unlimited
Subject Category 62
Availability: NASA CASI (443) 757-5802

13. SUPPLEMENTARY NOTES

Langley Technical Monitor: Paul S. Miner

14. ABSTRACT

Application agreement and integration services are required by distributed, fault-tolerant, safety critical systems to assure required performance. An analysis of distributed and hierarchical agreement strategies are developed against the backdrop of observed agreement failures in fielded systems.

15. SUBJECT TERMS

Agreement strategies; Atomic broadcast; Fault tolerance; Replica determinism

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			STI Help Desk (email: help@sti.nasa.gov)
					19b. TELEPHONE NUMBER *(Include area code)*
U	U	U	UU	39	(443) 757-5802