Overcoming the Performance and Security Challenges of Building Highly-Distributed Fault-Tolerant Embedded Systems

by

Andrew Loveless

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in the University of Michigan
2023

Doctoral Committee:

Associate Professor Ronald Dreslinski, Co-Chair
Associate Professor Baris Kasikci, Co-Chair
Assistant Professor Jean-Baptiste Jeannin
Associate Professor Linh Thi Xuan Phan, University of Pennsylvania

Andrew Loveless
loveless@umich.edu
ORCID iD: 0000-0003-0627-5702

To my lovely partner Lisa, who has provided me with unrelenting support for the past 13 years.

ACKNOWLEDGMENTS

Completing a Ph.D. has been one of the most difficult challenges of my life, and it would not have been possible without the support of many generous people. In particular, I would like to thank the following individuals.

Ron, my co-advisor, and the first person to contact me from the University of Michigan. His kind and relaxed disposition was a major factor in me choosing to attend the university. While academia can be a pressure cooker, Ron was never a source of that pressure, and he always encouraged me to prioritize my own happiness and health. Ron liked to remind students that a Ph.D. is a training exercise, and that their best work will come after their Ph.D. This perspective helped me through multiple challenging times.

Baris, my other co-advisor, who shaped me more as a researcher than anyone during my Ph.D. In particular, Baris taught me the importance of thinking outside the box and pursuing only the most interesting research ideas. His feedback early in my Ph.D. led to me abandoning several research directions, which ultimately saved me time and led to more exciting work. Baris also taught me the craft of technical writing with more patience than I would expect from any advisor.

Linh, my closest external collaborator, who was a source of constant inspiration throughout my Ph.D. Before a few overnight editing sessions with Linh, I had no idea anyone could operate at such a high level for such long periods of time. I hope I picked up even a small part of her work ethic and attention to detail. Despite my attending a different university, Linh also acted as my unofficial third advisor. Her guidance was essential to making much of my research successful, and I hope we continue working together.

Thank you additionally to the other amazing researchers who influenced my research trajectory. These include Jean-Baptiste Jeannin, who exposed me to his work early in my Ph.D. and provided valuable feedback on this thesis, Chris Peikert, who helped me navigate my early work using error-correcting codes, Allen Clement, who suggested I target real-time systems venues, and Mosharaf Chowdhury, who suggested I investigate reducing latency in embedded systems. Thank you also to my labmates in EfesLab and CADRE Lab, who are some of the brightest people I have ever met. I would be fortunate to work with any of them again in the future.

Thank you also to my colleagues at NASA for all their support. Jim Ratliff and Adam Schlesinger for encouraging me to pursue a Ph.D., even though it meant leaving their projects. My

branch and division management, Bruce Manners, Steven Fredrickson, Paul Delaune, Susan Morgan, and Rafael Munoz for helping me receive fellowships that allowed me to continue working at NASA while in school. Brendan Luksik, Cody Hodges, and Paul Muri, who graciously helped take on my NASA tasks whenever I was overloaded. Brendan Hall, Todd Smithgall, Paul Miner, and Wilfredo Torres-Pomales for insightful conversations that contributed to my work. Also, thank you to NASA's Artemis Network Validation and Integration Lab, which let me use lab space and equipment to conduct many of my experiments.

Thank you to my parents, Tom and Kathy, and sisters, Amanda and Meredith, who have always been encouraging and eager to hear about my research. Thank you also to my partner Lisa, who knows more about my research than anyone else. She is my biggest supporter, and this dissertation would not be possible without her help.

Lastly, thank you to the National Science Foundation Graduate Research Fellowship Program (award DGE 1256260), which partly supported this work. Note that any views expressed in this dissertation are my own and not necessarily those of NASA or any funding agencies.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE

LIST OF TABLES

TABLE

ABSTRACT

Over the past few decades, embedded systems, like those in spacecraft and aircraft, have evolved into complex distributed systems with hundreds of nodes and dozens of network switches. With this shift comes new challenges. One challenge is *performance*. Embedded systems are often required to mask faults. Unfortunately, traditional fault masking approaches, like state machine replication, require nodes to coordinate their actions by exchanging messages over several communication rounds. This means that in modern systems, where these messages often need to traverse multiple switch hops and must compete with hundreds or thousands of other traffic flows, traditional fault masking protocols can have high worst-case latencies that make it difficult or impossible to meet deadlines. For a variety of embedded systems, missing deadlines can be just as dangerous as generating incorrect outputs — potentially even causing system failure. A second challenge is *security*. As embedded systems have grown, designers have looked for new ways to reduce size, weight, and power. One popular approach is to use mixed-criticality networks, which let systems share a single network between critical and non-critical devices. These networks are designed to ensure that non-critical devices, which often come from unsecured supply chains, have no way to disrupt the critical systems. However, the existence of shared network resources provides a potential means for attackers to bypass these isolation guarantees.

To overcome the performance challenge, I introduce two new Byzantine fault-tolerant (BFT) state machine replication (SMR) protocols that exploit emerging hardware trends in embedded systems. The first, IGOR, exploits the increasing prevalence of multi-core processors. Rather than requiring nodes to agree on a single set of redundant sensor data to execute on, as in traditional protocols, IGOR lets nodes execute on multiple sets of redundant sensor data simultaneously on different cores. A coordination protocol is used in the background to determine which execution will determine the system's final state, reducing the system's latency to the time needed for either execution or coordination — whichever takes longer. The second protocol, CROSSTALK, exploits an increasingly popular network topology, in which messages travel simultaneously through redundant planes of switches. By using novel algorithms to move sensor data back and forth between the redundant planes, CROSSTALK can ensure processing nodes maintain identical state without requiring any communication between the nodes, significantly reducing latency. Moreover, CROSSTALK can be used on even modest single-core embedded processors.

To illustrate the security challenge, I introduce PCSPOOF, a new cyberattack on a popular mixed-criticality network technology called Time-Triggered Ethernet (TTE). TTE is used in a variety of critical systems, including spacecraft, aircraft, and wind turbines. PCSPOOF shows that TTE's switch forwarding rules can allow a malicious non-critical device to infer secret information about the TTE network that can be used to construct fake TTE synchronization messages. By using a small amount of extra circuitry, the malicious device can inject the fake synchronization messages into the network, disrupting the operation of critical systems. Moreover, an attacker can exploit a flaw in the implementation of modern TTE devices to increase the rate of successful injections. PCSPOOF was disclosed to multiple impacted organizations in 2021, including several large spaceflight companies. The disclosure had significant real-world impacts, with multiple organizations acknowledging the attack and implementing defenses. PCSPOOF has also influenced changes to the standard for the TTE synchronization protocol (SAE AS6802).

CHAPTER 1

Introduction

1.1 Background and Trends

Embedded systems are rapidly growing in size, as well as becoming increasingly distributed. For example, the growing complexity of software systems [1, 2], as well as the need for increased onboard autonomy [3], has dramatically increased the number of microprocessors in spacecraft, aircraft, and automobiles [4, 5, 6]. In the past few decades, automobiles have grown from having 15 processors on average [7] to more than 100 [4]. In the same time frame, crewed spacecraft have gone from containing a handful of processors to over 50 [8, 5]. Emerging commercial aircraft are expected to contain hundreds of processors [6]. Similarly, embedded data networks are growing to accommodate the increasing processor counts. Modern commercial aircraft can contain as many as 8 network switches [2], with next-generation aircraft expected to contain more than 10 [6]. Modern spacecraft commonly contain anywhere from 6 to 12 switches [9].[1]

Meanwhile, the importance of fault tolerance in embedded systems is increasing. For example, the growing processor counts in embedded systems mean the probability of a processor fault occurring is increasing, which if not mitigated, could cause a system to fail [12]. Additionally, the desire for faster processing means smaller transistors are being used [13, 14], which are more susceptible to transient upsets from environmental factors like radiation and electromagnetic interference [13, 15]. Moreover, embedded systems are increasingly being asked to autonomously perform safety and mission critical functions that in the past would have been controlled by human operators [3, 16, 17, 18, 19]. For example, crewed spacecraft being developed for deep space missions, such as to asteroids and Mars, will have significant delays when communicating with Earth (e.g., 10–20 minutes [20]) — meaning they cannot be controlled and troubleshooted from Earth like most spacecraft today [3, 21]. Similarly, small modular reactors, which are being developed to reduce the costs and risks of nuclear power production, will be required to operate with near

[1] These switch counts do not account for onboard redundancy to tolerate failures, which often doubles or triples the actual number of switches in practice [10, 11].

autonomy to be financially viable [22]. Finally, autonomous automobiles [19] and aircraft [23] will be required to navigate safely without any operator control.

Unfortunately, traditional fault tolerance approaches used in embedded systems scale poorly to large networks. For example, it has been shown that in order for processors to reach agreement on some data in the presence of faults [24] — a common requirement of fault tolerance protocols in practice [25, 26] — the processors must exchange messages with each other over multiple communication rounds [27]. The number of required rounds grows as the number of faults to tolerate increases. This need for multiple communication rounds leads to large system response times [28] — particularly as the number of network hops between processors, as well as the amount of network traffic (which grows with the number of processors), increases. These large response times can make it difficult or impossible for systems to meet certain end-to-end deadlines (i.e., from sensors, to processors, to actuators) [28], which commonly must be on the order of tens of milliseconds to ensure adequate control stability [29, 26]. The problem is compounded by the need for systems to host an increasing number of fault-tolerant functions, which require software tasks be replicated on multiple processors and thus consumes more total processor time — possibly preventing some tasks from being schedulable [28].

Additionally, the growing size of embedded systems, as well as the increasing importance of fault tolerance (and thus need for replication [30]) in these systems, naturally increases size, weight, power, and cost (SWaP-C). In order to counteract this increase, a new trend has emerged in which, instead of using multiple data networks — each optimized for different purposes (e.g., deterministic command and control traffic, high-throughput video streaming), an embedded system will have a single network that is intended to meet the needs of all its subsystems [31, 32, 33, 34]. Such networks are called *mixed-criticality* networks because they allow both highly critical devices and less important commercial-off-the-shelf (COTS) devices to share the same switches [35]. For example, the flight computers and digital engine controllers in an airplane can connect to the same switches as the onboard Wi-Fi access point and passenger entertainment system [35].

The use of a mixed-criticality approach means the network is responsible for isolating critical traffic from non-critical COTS traffic [36]. In other words, even if a COTS device is faulty or compromised by an attacker, it should not be able to interfere with the correct or timely execution of a critical function [36]. This isolation is important, since COTS devices often come from unsecured supply chains and typically do not go through any detailed verification process that would detect tampering [37, 38]. Mixed-criticality networks are rigorously tested and sometimes even formally verified [39] to ensure they provide this isolation under all possible conditions.

1.2 Thesis Statement

At first glance, it appears like the need for coordination between redundant processors [27] means there is no way to avoid the poor performance of traditional fault-tolerant techniques when used in large distributed embedded systems. However, this is not the case. Rather, it is possible to take advantage of recent hardware trends in novel ways in order to counteract large message delays between processors and, as a result, significantly increase performance. Similarly, it appears like mixed-criticality networks are a safe and effective way to reduce the cost of constructing embedded systems. However, even the most rigorously verified network technologies are not infallible. The existence of shared resources means there can still be ways for an attacker to circumvent the isolation guarantees of the network in order to interfere with critical systems.

Thesis Statement

In this dissertation, I describe new ways to leverage multi-core processors and redundant switched network topologies in order to significantly increase the performance of fault-tolerant embedded systems. In addition, I show how the use of mixed-criticality network technologies can make systems susceptible to a new type of cyberattack from a malicious COTS device.

1.3 Dissertation Contributions

1.3.1 Leveraging Hardware Trends to Reduce Latency

The first part of this dissertation describes new ways to leverage recent hardware trends to increase the performance of fault-tolerant embedded systems. In particular, I focus on reducing the worst-case latency of Byzantine fault-tolerant (BFT) state machine replication (SMR). BFT SMR is a fault tolerance technique commonly used in critical systems like spacecraft [40, 41], airplanes [26, 42], wind turbines [43], and nuclear reactors [44, 22]. In general, it works by executing identical software simultaneously on multiple processors, or *replicas*. If some of the replicas become faulty — perhaps generating erroneous output — the actuators can perform a majority vote of the outputs of the replicas in order to determine the correct result.

BFT SMR suffers from high latencies due to the need for replicas to reach agreement on sensor data before executing. For example, the replicas may agree to execute on data d_1 from some sensor s_1, and that d_1 is the median value produced by three redundant sensors. Without agreement, non-faulty replicas could operate on different data and therefore generate different outputs — causing the majority vote at the actuators to fail. Agreement protocols take at least $f + 1$ communication rounds to tolerate f faults [27], meaning at least $f + 1$ communication delays are added to the

latency of every execution. Commonly, the agreement process alone can take up more than half the time needed to execute the overall BFT SMR protocol [28].

IGOR— Accelerating Byzantine Fault Tolerance with Eager Execution. To address the problem of high latency in BFT SMR, this dissertation introduces the concept of *eager execution*. Eager execution is a speculative computing technique that allows BFT SMR systems to overlay agreement on sensor data with execution, reducing the overall latency to the time taken by the agreement or execution process, whichever is longer. The main challenge of this approach is ensuring non-faulty replicas operate on the same sensor data. Eager execution overcomes this challenge by exploiting parallelism. Specifically, the approach leverages multi-core processors to allow replicas to execute on data from multiple redundant sensors simultaneously (rather than on data from just one agreed-upon sensor). Each execution is performed on a different core. While these executions are in progress, an agreement process is used to select which sensor's data (i.e., which execution) will determine the system's final state. This dissertation will show that, with this eager execution technique, it is possible to ensure the selected execution was always performed by a majority of non-faulty replicas. Moreover, the selected sensor data is guaranteed to either originate from a non-faulty sensor, or be bounded by data from non-faulty sensors.

The dissertation then describes IGOR, the first BFT SMR protocol to use eager execution. IGOR uses new agreement protocols that work in any network topology (e.g., point-to-point, multidrop bus, switched) and are optimized for the single or multi-fault cases. The standard (single-fault) version of IGOR takes the theoretical minimum number of communication rounds [27] to reach agreement and can use any existing Byzantine agreement protocol [45, 46, 47] as a primitive. Compared to the state-of-the art, it reduces latency by roughly 30% and improves schedulability by 1.88× on average. However, it can suffer from large latencies when tolerating multiple faults due to the need to split large messages into fragments. To address this, I introduce a multi-fault optimized version of IGOR that uses a binary reduction technique to convert agreement on arbitrary-length sensor data to agreement on a single bit. This multi-fault optimized protocol can reduce latency by as much as 1.75× and improves schedulability by 3.22× over the state of the art.

CROSSTALK— Making Low-Latency Fault Tolerance Cheap with Redundant Networks. The main shortcoming of IGOR is that, despite its latency and schedulability improvements, it requires execution on multiple cores. This means that it is not possible to use IGOR on low-powered single-core embedded processors [48]. Moreover, even if a multi-core processor is used, the need to perform eager executions on different cores means the capacity for the system to perform other computations is reduced. For example, a taskset that fully utilizes a single core on a non-fault-tolerant system might fully utilize three cores if all tasks were converted to use IGOR.

To overcome this shortcoming, this dissertation presents CROSSTALK, a new BFT SMR protocol that achieves comparable or lower latency than IGOR without requiring multi-core processing.

The key insight behind CROSSTALK is that the reason IGOR is potentially expensive is that it makes no assumptions about the network topology — instead treating the replicas as if they are connected by unicast channels. By making *slightly* stronger assumptions about the network topology, it is possible to have significantly lower overheads.

Specifically, CROSSTALK leverages the fact that modern real-time embedded systems are almost universally adopting *redundant switched networks* — i.e., switched networks where fault tolerance is provided by sending messages through redundant planes of switches [49, 11, 50, 51, 52, 53, 54, 55, 56, 57]. CROSSTALK exploits this topology by crossing messages between the planes using special algorithms to ensure no non-faulty replica can receive data from a sensor that other replicas did not also receive. In other words, CROSSTALK solves agreement entirely in the network without requiring communication between replicas. CROSSTALK also leverages special features of modern switches, such as the use of self-checking pairs to mask transmission errors, to avoid the need for voting logic and digital signatures that would otherwise require significant processing time. The result is that CROSSTALK can improve schedulability by 2.13–4.24× compared to IGOR, while still staying compatible with networks that systems often use in practice.

In presenting CROSSTALK, this dissertation also proves for the first time the impossibility of tolerating timing faults in synchronous agreement protocols that do not feature any communication between replicas. This proof is important not just for CROSSTALK, but because it shows that some recent results [58] on constructing BFT protocols for embedded systems can actually be incorrect if timing faults are possible.

1.3.2 Compromising Safety in Mixed-Criticality Networks

Like virtually all fault tolerance techniques used in safety and mission-critical real-time systems today [26, 59, 42, 60, 61, 62, 63, 64], IGOR and CROSSTALK make some basic assumptions about the behavior of non-faulty devices. These assumptions include that non-faulty processing nodes are synchronized to one another, and that messages sent over non-faulty network segments will be delivered successfully and by known deadlines.

The second part of this dissertation shifts gears and considers how a malicious actor might break these assumptions in mixed-criticality networks, causing even *non-faulty* processing nodes and network segments to become unpredictable (mimicking the behavior of faulty devices). It specifically focuses on *Time-Triggered Ethernet (TTE)*, a popular technology used in a variety of critical systems, including spacecraft [59, 52, 11], aircraft [65, 66, 67], and wind turbines [68]. The TTE protocol allows safety-critical devices, like flight computers, to communicate with low latency and jitter over multiple redundant planes. TTE's isolation guarantees also allow COTS devices to share the network while ensuring they cannot interfere with the TTE communication.

TTE is perhaps the most well-vetted mixed-criticality network technology in use today, and many of its sub-protocols have been formally verified [39].

PCSPOOF— Compromising the Safety of Time-Triggered Ethernet. PCSPOOF is the first cyberattack to break TTE's isolation guarantees, allowing a malicious COTS device connected to a TTE network — perhaps compromised by a supply chain attack — to cause resets of the TTE synchronization protocol [32]. Each reset can interfere with the timing of interrupts on critical processors connected to the network, as well as cause TTE messages to be delayed or dropped.

PCSPOOF works by exploiting a weakness in the TTE synchronization protocol, which is that a specific synchronization message, or *protocol control frame (PCF)*, from a trusted switch can cause a processor to detect a clique (a set of processors synchronized to each other, but not to the rest of the system) [69]. Upon detection of a clique, the processor briefly loses synchronization before resynchronizing to the network [32]. Normally, it is impossible for a COTS device to send such a message (it is dropped by the switch). However, by conducting electromagnetic interference (EMI) into the switch through an Ethernet cable, it is possible for a COTS device to send the message and trick the switch into forwarding it to other TTE devices. Moreover, only five circuit components are needed to generate the required EMI, meaning that the current process [37] used for verifying COTS devices for use in critical systems may not detect the malicious circuity.

This dissertation shows that PCSPOOF attacks can be repeated as often as every 10–15 seconds. Each attack can cause TTE devices to lose synchronization for up to a second and drop tens of TT messages — both of which can result in the failure of critical systems like spacecraft or aircraft [26, 29]. It also shows that, in a simulated spaceflight mission, PCSPOOF can cause uncontrolled maneuvers that threaten safety and mission success.

The development of PCSPOOF has had significant impact on real systems. In 2021, the attack was disclosed to multiple organizations, including NASA, ESA, Northrop Grumman, and Airbus, and several have implemented mitigations suggested in this dissertation. The TTE standard, SAE AS6802 [32], is also being updated to enable the use of larger TTE PCFs, which, once incorporated into next-generation devices, will disable the mechanism PCSPOOF uses to trick the switch into forwarding malicious PCFs.

1.4 Road Map

The remainder of this dissertation is structured as follows. First, Chapter 2 describes eager execution, a new technique for reducing the latency of BFT SMR in embedded systems, as well as IGOR, the first BFT SMR protocol to use this technique. Next, Chapter 3 describes CROSSTALK, a BFT SMR protocol that improves on the schedulability of IGOR by exploiting common aspects of switched networks used in modern embedded systems. Chapter 4 then describes PCSPOOF,

the first attack capable of breaking TTE's isolation guarantees. Finally, Chapter 5 concludes and discusses how future research could build on the work in this dissertation.

CHAPTER 2

IGOR: Accelerating Byzantine Fault Tolerance with Eager Execution

2.1 Introduction

Real-time control systems, such as those in spacecraft and aircraft, often use Byzantine fault-tolerant (BFT) state machine replication (SMR) to mask errors due to hardware faults and environmental factors, such as cosmic radiation [62, 63, 61, 70, 71, 64, 44, 42, 72, 14, 73, 74, 75, 40, 26, 76]. In these systems, the same function is performed simultaneously by multiple processors (i.e., *replicas*), each of which receives data from redundant sensors. To ensure that the replicas maintain the same internal state and produce the same output, care must be taken to ensure they operate on the same input data. This is typically accomplished by having the replicas run an *agreement protocol* on the data from each sensor [26, 77, 42]. The replicas then perform a *source selection* to choose a single "best" sensor value as the system input [71, 78, 26]. The source selection step is typically application-specific, and can be as simple as a mid-value selection [71, 26]. Finally, the replicas *execute* on the selected input value.

Unfortunately, this "agree-execute"[1] approach used in traditional BFT SMR systems has a significant drawback: *high latency*. Since any deterministic agreement protocol requires at least $f+1$ rounds of communication to tolerate f faulty replicas [27], requiring that the replicas reach agreement on inputs *before* executing adds at least $f+1$ rounds of communication latency to *every execution* they perform on the incoming data. As we show in §2.6, this additional delay can make it impossible to meet certain hard deadlines. Moreover, even when all deadlines can be met, the added latency results in an unavoidable reduction in real-time control performance and system stability, and thus increases the complexity of the control software to compensate for the extra delay [80].

[1] We borrow the term "agree-execute" from [79], but narrow its meaning to refer specifically to BFT systems that agree on inputs before executing.

A standard approach to addressing this challenge in non-real-time systems, such as data centers and blockchains [81, 82, 83, 84, 85, 86], is to adopt *speculative* execution, which forgoes agreement on the inputs altogether and *assumes* the replicas will end up in identical states. In the common case where replicas do not diverge, this approach completely avoids running an agreement protocol. However, when the states do diverge — which can be caused by faulty replicas — the system needs to rollback and repeat previous executions [81, 82, 83, 84, 85]. This outcome is not generally acceptable for real-time systems, since repeating executions can substantially increase worst-case response times, leading to deadline misses.

In this paper, we present IGOR, a novel speculative BFT SMR approach that leverages the increasing prevalence of multi-core processors in real-time embedded systems [14, 48] to enable speculation *without* rollback. The key idea behind IGOR is to *eagerly* execute on the data from redundant sensors simultaneously, without knowing which execution (and thus which sensor) will be used to determine the system's final state. While these executions are underway, the replicas reach agreement on which states to discard and which to keep. As soon as the executions and the agreement process are completed, IGOR can deliver results to the actuators. Thus, when the executions take longer than agreement (which is common in practice, as we show in §2.6), IGOR's end-to-end latency (from sensors to actuators) is the same as that of a non-replicated system — that is, IGOR achieves the *minimum possible latency*. In all other cases, IGOR's ability to overlay agreement and execution inevitably results in latency savings.

It is not sufficient to simply run an existing agreement protocol concurrently with the speculative executions, however. The reason is that, although the source selection algorithm will still select a single "best" sensor value, we cannot guarantee that the selected sensor value has been executed on by enough non-faulty replicas, and thus there may not be enough non-faulty replicas with the same state to out-vote the faulty replicas (c.f. §2.4.1). We solve this problem by introducing new agreement protocols that allow replicas to simultaneously reach agreement on both (1) the sensor values they received *and* (2) how many replicas received each of those sensor values. As such, the protocols are able to guarantee that a given sensor value cannot be selected unless there is a high enough number of replicas claiming to possess the value such that, even if some of those replicas are faulty, a sufficient number of replicas must have executed on the value successfully. Importantly, IGOR can provide this guarantee without sacrificing correctness or using more communication rounds than that of a traditional agreement protocol.

To evaluate IGOR's performance, we implemented a prototype of IGOR in NASA's Core Flight System (cFS) [87], an open-source general-purpose flight software framework used in a variety of real spacecraft, including the Lunar Reconnaissance Orbiter and Parker Solar Probe [88]. Our experimental evaluation shows that, for realistic system configurations, IGOR reduces latency by up to 1.75× and improves schedulability by 1.88–3.22× compared to the state of the art, and

that its latency closely matches the theoretical minimum (produced by a non-replicated system). We also used IGOR to execute simulated guidance, navigation, and control software from a real spacecraft (including multiple genuine flight software components); our evaluation shows that IGOR is able to meet deadlines that existing solutions cannot, leading to improved vehicle stability and performance.

In summary, we make the following contributions:

- IGOR: a speculative BFT SMR system with low latencies in both the presence and absence of faults (§2.4).
- A prototype of IGOR for NASA's cFS framework (§2.5).
- An experimental evaluation of IGOR, including benchmarks (§2.6.1) and schedulability analysis (§2.6.2).
- A case study of IGOR in a spaceflight application (§2.6.5).

2.2 Background and Challenges

In this section, we describe how BFT SMR systems are built today and why such a design results in poor performance.

2.2.1 Overview of a BFT SMR System

A typical BFT SMR system consists of multiple redundant processors (i.e., replicas) that communicate with a variety of redundant input and output devices [62, 63, 61, 64, 44, 42, 72, 70, 71, 14, 73, 74, 75, 40, 26, 76]. Depending on the application, these devices may include inertial measurement units, star trackers, remote interface units, and engine or thruster controllers. In modern systems, all these devices are connected to a single backbone network [71, 89, 59, 90]. For simplicity, we refer to all input devices as "sensors" and all output devices as "actuators". We refer to each input data item from a sensor as a "value".

To ensure non-faulty replicas never produce conflicting commands, it is necessary to ensure they maintain the same internal state, which in turn requires replicas to operate on the same inputs in the same order. Typically, the *ordering* of the inputs is known a priori (e.g., because system components are synchronized and tasks and traffic patterns are scheduled offline [91]). To ensure that the actual *content* of the inputs is identical, the replicas run an agreement protocol on all inputs before executing [77, 71].

Figure 2.1 illustrates this "agree-execute" approach: ❶ Each sensor sends its value to each of the replicas; since sensor 3 is faulty, some replicas receive different values from this sensor. ❷ The replicas then use an agreement protocol to agree on the value from each sensor, and ❸ perform

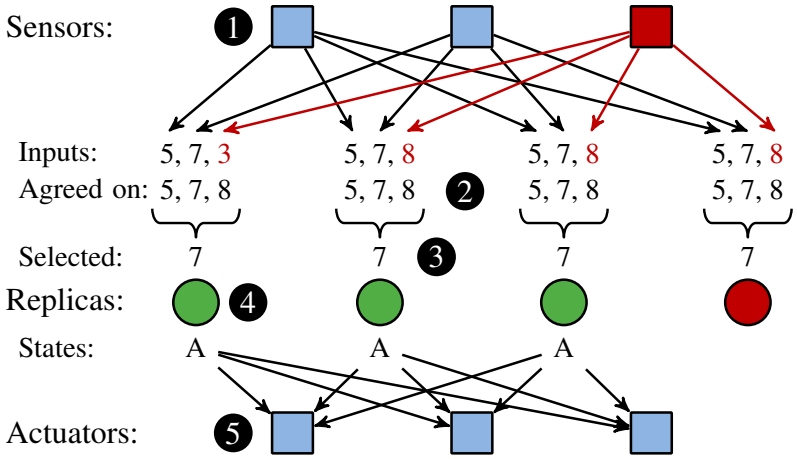

Figure 2.1: **Traditional BFT SMR** — Overview of a traditional agree-execute BFT SMR system. Sensor 3 and replica 4 are faulty.

source selection[2] to choose a single "best" sensor value to use as input for the computation. ❹ The replicas then perform the same deterministic computation on the selected input, and thus all non-faulty replicas end up with the same state. The replicas use the state to determine the system output, then send the output to the actuators.[3] ❺ The actuators use a majority vote to mask the output of a bad replica.

Notice that if the replicas *did not* run an agreement protocol before executing, then the source selection could result in different non-faulty replicas using different sensor values as their inputs, which would cause their internal states to diverge.

2.2.2 Drawbacks of the Agree-Execute Approach

Real-time systems require low latency for tight control loops and hazard response. Unfortunately, traditional BFT SMR systems suffer from high latency, as the replicas must reach agreement before executing and the latency of the agreement protocol is often significant.

The latency of an agreement protocol depends on two factors: the *number* of communication rounds in the protocol and the *length* of each round. A round must be large enough to ensure all messages sent at the start of a round are received by the end of that round [45, 46, 92]. Therefore, the round length must be at least as large as the worst-case traversal time (WCTT) of a message over the network.

In large systems like spacecraft and aircraft, which commonly have thousands of traffic flows and large networks spanning multiple switches [57, 93, 94], the message WCTT — and thus mini-

[2]The source selection process is typically application-specific; it could be as simple as a mid-value selection [26].
[3]A state change does not always require an output to be sent to the actuators.

mum round length — can be on the order of *several milliseconds* [93, 94, 95, 96, 57]. For example, in deterministic Ethernet (AFDX [31]) networks used in avionic systems, the WCTT can be up to 3–4 ms for individual Ethernet frames [57, 96]. Large messages must be fragmented into multiple frames sent *at least* 1 ms apart [31], and more commonly 4+ ms apart [57], which can result in message WCTTs that are multiple times that of a single frame [31]. We note that since agreement protocols often require replicas to send increasingly larger messages in each round [46, 45, 92, 97, 98], fragmentation is common, even if the data to agree on is small. For example, agreeing on 250 bytes in a typical 1 fault-tolerant BFT SMR system (4 replicas, where 1 replica may be faulty) requires replicas to broadcast 2250 bytes of data in the second round, which must be fragmented across at least 2 Ethernet frames. Agreeing on 1200 bytes would require 8 fragments.

Time-triggered networks [34, 99, 32, 100] can reduce frame WCTTs to hundreds of microseconds [89, 101]. However, if the frame WCTT is short, the round length is then dictated by other factors. For example, each round cannot be shorter than the period at which the network schedule lets devices access the network, which is often on the order of milliseconds to avoid delaying or dropping other traffic [102, 89]. Further, the round length cannot be shorter than what the *software* schedule allows. For example, replicas typically communicate through a dedicated I/O task or partition that executes with a fixed period [103, 104], which is often 10+ ms to avoid excessive processing overhead [105, 104]. Lastly, large messages still need to be fragmented across multiple time slots [95].

Since a single round can take several milliseconds, and any deterministic agreement protocol must take at least $f + 1$ rounds [27], simply having the replicas agree on sensor data can take upwards of 10 ms even for small values of f. Often, this agreement latency can be as much as, or exceed, the time needed for execution on the agreed upon data [106, 107], thus resulting in end-to-end latencies that are 1.5–2$\times$ higher than that of a non-replicated system (see §2.6).

The standard way to reduce the *average* latency of BFT SMR in non-real-time systems is to adopt a *speculative* approach [81, 82, 83, 84, 85] that avoids executing an agreement protocol in the (common) fault-free case. Unfortunately, using this strategy in a real-time system makes little sense, since the *maximum* latency, which occurs when replicas are faulty and requests need to be re-executed, can be *even higher* than in traditional BFT SMR systems.

2.3 Models

This section describes our system and fault models. Our models are consistent with real-time systems that rely on BFT in practice [62, 63, 26, 70, 59, 42].

2.3.1 System Model

We consider a distributed system of processors, sensors, and actuators (which we collectively refer to as *devices*) connected to a network. We assume the system is synchronous, i.e., there are known upper bounds on the time needed for processors to perform computations and for messages to traverse the network [108, 109, 46]. This is typically accomplished using specialized real-time operating systems [110, 111] and networking protocols [31, 32, 100, 42, 33, 112].

We assume all devices are synchronized, either via the network [32, 100, 33] or external timing equipment [113], and that the system progresses in a series of *rounds*. At the start of each round, devices send messages. At the end of each round, devices read messages and perform computations. Messages are received in the same round they are sent.

We assume replicas can communicate directly with each other, and with sensors and actuators, over the network. This assumption follows trends in distributed real-time control systems, which are becoming increasingly "flat", with a single backbone network connecting all system components [60, 71, 89, 59]. The network can use any physical topology, which may include switches [33, 32, 31], buses [100, 60], or point-to-point links [42, 62, 61]. Regardless of topology, we model *communication* between devices as point-to-point.

Lastly, we assume a device can identify the sender of any message it receives. This is a necessary assumption in any BFT system [46], since otherwise a faulty device could impersonate all the other devices [114]. The assumption is trivially satisfied in point-to-point networks. In other networks, it is typically satisfied using (1) static routing tables, where the devices connected to each switch port (for example) are known a priori, and switches discard messages that disagree with the table [31, 32, 33], or (2) a time-division multiple access scheme [60, 70, 59, 112], in which the sender is implied by the time a message is received.

2.3.2 Fault Model

We consider the classical Byzantine fault model, where, in a system with n replicas and m sensors, $f < n/3$ of the replicas [46] and $g < m/2$ of the sensors [115, 40] can be faulty. Faulty devices can exhibit any possible behavior, including pretending to have received messages that were never sent, dropping or failing to send messages, sending messages at the wrong time, or sending conflicting messages to different devices. For example, a faulty replica that is supposed to broadcast some value v to all replicas may instead send v to some replicas, v' to some other replicas, and no value to the rest of the replicas.

Byzantine behaviors can have multiple causes in practice, e.g., software errors [116], device wear-out [117], and bits flips from charged particles [118]. Broadcast networks can prevent some of these behaviors, since messages are sent only once by the device and replicated by the switch

or bus [31, 32]. However, Byzantine behaviors can still result from a faulty device that transmits different values on different redundant networks, with some receivers reading the message from network A first while others from network B first [70]. A faulty device can also transmit a marginal signal that is interpreted differently by different receivers receiving the same message [119].

In contrast to the replicas and sensors, we assume the network itself is *reliable*. This has two implications. First, the network cannot drop messages. This is typically accomplished using automatic retransmissions [33], or redundant network planes [31, 32, 100], in which messages are sent over multiple independent networks simultaneously. Second, the network cannot corrupt or create messages. This is typically accomplished by having network cards vote messages from redundant networks [71, 70], or by using specialized self-checking switches that are proven to fail silent with sufficiently high probability [120, 60, 59]. We note that by not worrying about network faults, we keep IGOR's design general and applicable to a broad range of architectures; there is no reason IGOR could not be analyzed in the context of network faults.

Finally, in safety-critical systems, where BFT is perhaps most used in practice [62, 63, 61, 70, 71, 64, 44, 42, 73, 74, 75, 40, 26], cryptographic methods like digital signatures are typically considered an insufficient means of constraining the behavior of faulty devices [121, 122, 123, 124]. We design IGOR to avoid cryptographic assumptions and to be correct in all possible executions.

2.4 Design

This section describes IGOR, a new speculative approach for building high-performance BFT SMR systems suitable for real-time applications. In general, IGOR has two main goals:

- **Low latency in all executions** — Real-time systems must meet deadlines in the worst-case scenario (i.e., under faults). There is no benefit to being fast in the absence of faults if the system is slow when faults do occur.
- **Robustness to faulty sensor data** — Traditional BFT SMR architectures use a source selection process to reject bad sensor data and select a single "best" input. Despite IGOR's speculative approach, it needs to retain this same robustness to faulty sensor data.

IGOR accomplishes these goals using a speculative approach, in which — rather than agreeing on sensor data before executing — replicas eagerly execute on sensor data while simultaneously agreeing on which data to use. The key idea is to keep replicas from wasting time communicating when they could be getting closer to delivering a result. Of course, IGOR's ability to reduce latency relies on the replicas' ability to perform extra computations. However, we believe this is a worthwhile trade-off in critical low-latency applications (e.g., flight control) that could not meet

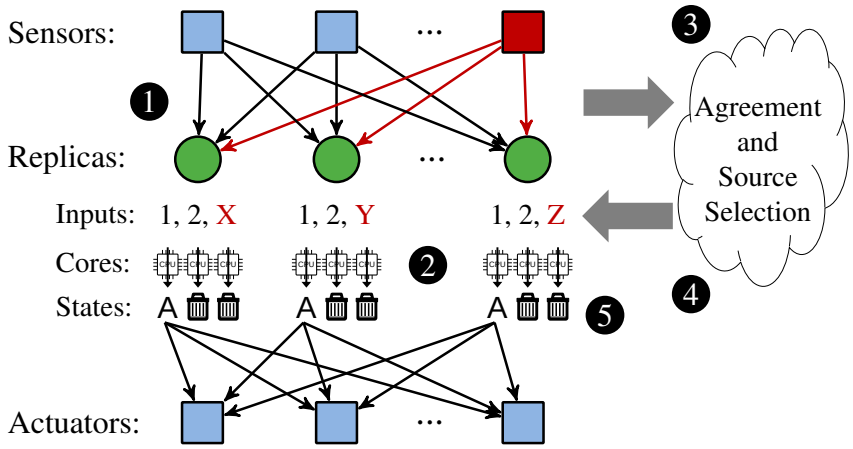

Figure 2.2: **Overview of IGOR** — High-level design of IGOR showing eager speculative executions on data from different sensors. Sensor 3 is faulty.

deadlines otherwise. It also lets IGOR take advantage of emerging aerospace processors, which can feature 4, 8, or more cores [14, 48].

Figure 2.2 gives an overview of IGOR's design. ❶ Like in traditional BFT SMR systems, the replicas get a value from each sensor; however, they do not run an agreement protocol on the values directly. ❷ Instead, each replica delegates each sensor value to a different core and executes on each value simultaneously. Each execution produces a (potentially different) resulting state, which is stored temporarily. ❸ While these executions are in progress, the replicas use an agreement protocol to determine which replicas claim to possess the value from each sensor. As a result of the agreement process, the replicas end up with an identical set of "candidate" sensors — sensors whose value could be selected as the "trusted" system input. IGOR ensures that if a sensor is non-faulty, it is guaranteed to be in this candidate set. ❹ The replicas then perform a source selection process to determine which candidate sensor, and therefore which execution, will be used to determine the system's final state. This process is analogous to the source selection performed in an agree-execute system. ❺ Once the executions on the sensor values are complete, the replicas commit the state resulting from the chosen execution and discard other states. If an output is required, the replicas reference their (now final) state to determine the output and send it to the actuators. Like in agree-execute systems, the actuators use a majority vote to determine the system output.

IGOR also uses two key optimizations to make BFT SMR more efficient. First, it uses a *binary reduction* technique [125, 126, 127, 128] to reduce the problem of agreeing on arbitrarily large sensor data to that of agreeing on a small constant number of bits. This technique enables IGOR to achieve low latencies even when tolerating multiple faults, which would otherwise be impossible due to the added latency of fragmenting large messages (see §2.6.1). Second, IGOR exploits the

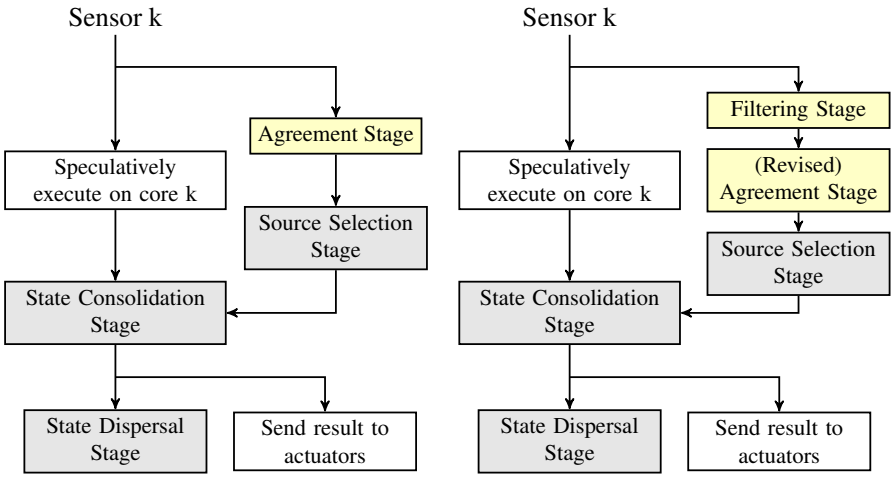

Figure 2.3: **Stages of IGOR** — Stages of IGOR optimized for the single (left) and multiple (right) fault-tolerant cases.

fact that, even though all non-faulty replicas must agree on the system state before the next iteration of the protocol, only a subset of those replicas are needed to deliver correct results to the actuators. Therefore, IGOR can produce an output before the agreement process is actually completed.

The following sections describe IGOR in detail. We start with a version of IGOR that is optimized for low latency in the single-fault case (§2.4.1). We then modify the protocol using a binary reduction technique to make a version of IGOR optimized for tolerating multiple faults (§2.4.2).

2.4.1 Optimizing Eager Execution for the Single-Fault Case

Figure 2.3 shows an outline of the protocol, which proceeds in multiple stages. The Agreement and Source Selection Stages are executed in parallel with the speculative executions. The State Dispersal Stage is executed concurrently with delivering results to the actuators. We now describe each stage in detail.

2.4.1.1 Agreement Stage

Each replica starts executing on the value from sensor k immediately after receiving it, with the execution assigned to core k. However, since sensor k may be faulty, and thus can send different values to different replicas, two non-faulty replicas may execute on different values on their k-th core. The goal of the Agreement Stage is to detect these kinds of inconsistencies and to ensure replicas agree on the content of the data.

Unfortunately, it is not sufficient for IGOR to simply place a traditional agreement protocol in parallel with the speculative executions. The reason is that, although traditional agreement protocols guarantee all non-faulty replicas end up with the same value, they make no guarantees about

how many replicas started with — and thus executed on — that value [45, 129, 130]. Even agreement protocols with stronger guarantees [131], such as ensuring that a value cannot be decided unless at least one non-faulty replica started with that value, are not sufficient. For example, consider a typical system with 4 replicas (1 faulty) and 3 sensors (1 faulty). The faulty sensor may send a value v to one replica and v' to the others, which the replicas start speculatively executing on. After running an agreement protocol, the replicas may decide on the value v from the faulty sensor, and that value may be selected by the source selection process as the system input. However, only one non-faulty replica actually executed on that value originally, and thus has the resulting state, which is not enough to out-vote a faulty replica when delivering results to the actuators.

IGOR's Agreement Stage fixes this problem by having replicas simultaneously agree on both the content of the sensor data and how many replicas received it. This is described in Protocol 2.1, which the replicas execute on each sensor value. It uses an existing Byzantine agreement (BA) protocol [46, 129, 130] as a primitive, which guarantees that, when a faulty replica broadcasts a value, all other non-faulty replicas receive the same value [46].

In Protocol 2.1, $\mathit{MyValue}_k$ is the value the replica received from sensor k. If no value was received, it is set to a predetermined default value. Values_k is an n-dimensional vector used to store values other replicas received from sensor k (recall that n is the number of replicas). All elements are initialized to $\perp$ to indicate the values are missing. $\mathit{Candidate}_k$ and $\mathit{MyAccept}_k$ are variables indicating whether sensor k's value is a candidate for source selection and whether the replica executed on that value originally, respectively. Both are initialized to `False`. Note that broadcasting includes logically sending to one's self.

Protocol 2.1: Agreement Stage

- Broadcast $\mathit{MyValue}_k$ to all replicas using a BA protocol
- $\mathit{Values}_k[j] \leftarrow$ the value broadcasted by replica j (if any)

  if $\geq n - f$ values in Values_k are the same **then**:

  $\mathit{Candidate}_k \leftarrow$ `True`

  Let v be the value with $\geq n - f$ matching copies

  if $\mathit{MyValue}_k = v$ **then**: $\mathit{MyAccept}_k \leftarrow$ `True`

  else: $\mathit{MyValue}_k \leftarrow v$

LEMMA 2.1. For any sensor k, $\mathit{Candidate}_k$ is the same for all non-faulty replicas. Further, if $\mathit{Candidate}_k$ is `True`, $\mathit{MyValue}_k$ is the same for all non-faulty replicas.

Proof. Each value in Values_k was broadcasted with a BA protocol; hence, it is the same for all non-faulty replicas. Thus, if one non-faulty replica sets $\mathit{Candidate}_k \leftarrow$ `True`, so do all non-faulty replicas. Otherwise, $\mathit{Candidate}_k$ is `False`.

We next prove the second part of the lemma. Suppose two non-faulty replicas R_i and R_j end up with different values for $\mathit{MyValue}_k$. Since $\mathit{Candidate}_k$ is `True`, the two replicas must have received different sets of $\geq n - f$ values that match the $\mathit{MyValue}_k$ they end up with. Since the values were broadcasted with a BA protocol, each replica must possess the same values in Values_k. However, any two sets of $\geq n - f$ values must overlap at $\geq n - 2f$ values, which is ≥ 1. Hence, R_i and R_j could not have received different sets of $\geq n - f$ matching values, which is a contradiction. □

2.4.1.2 Source Selection Stage

In the Source Selection Stage, the replicas decide which of the candidate sensors will determine the system's final state using an existing deterministic source selection algorithm. This process is analogous to the source selection process in traditional agree-execute systems, except that it happens in parallel with the executions.

Protocol 2.2 describes the Source Selection process. *SelectedSource* is a variable containing the ID of the selected sensor, which in turn determines the selected state.

Protocol 2.2: Source Selection Stage

- Let $\mathcal{C}$ be a set of tuples of the form $(k, \mathit{MyValue}_k)$, where $\mathit{Candidate}_k =$ `True` for each tuple
- Use the source selection algorithm to select the sensor with the "best" value in $\mathcal{C}$
  SelectedSource ← the ID of the selected sensor

LEMMA 2.2. All non-faulty replicas have the same set $\mathcal{C}$ and the same *SelectedSource*.

Proof. By Lemma 2.1, if one non-faulty replica includes sensor k in $\mathcal{C}$, then all non-faulty replicas do too. Also, by the same lemma, $\mathit{MyValue}_k$ is the same for any sensor k included in $\mathcal{C}$. Thus, set $\mathcal{C}$ is the same for all non-faulty replicas, and as long as the source selection algorithm is deterministic, all non-faulty replicas select the same sensor. □

LEMMA 2.3. If some sensor k is non-faulty and sends value v, then set $\mathcal{C}$ contains sensor k (and its value v).

Proof. Since sensor k is non-faulty, all non-faulty replicas set $\mathit{MyValue}_k \leftarrow v$ and broadcast v in the Agreement Stage. Since $\leq f$ replicas are faulty, each non-faulty replica receives $\geq n - f$ copies of v, sets $\mathit{Candidate}_k \leftarrow$ `True`, and keeps $\mathit{MyValue}_k$ as v. Thus, for all non-faulty replicas, set $\mathcal{C}$ contains (k, v). □

2.4.1.3 State Consolidation Stage

Once Source Selection is over, all non-faulty replicas know which sensor value was selected as the input. During State Consolidation, replicas that executed on that value commit the resulting state, and all replicas discard states resulting from the other speculative executions.

Protocol 2.3 describes the State Consolidation Stage. *SavedState* is a variable containing the current system state, which persists to the next iteration of the protocol. $\mathit{TempState}_k$ is the temporary state resulting from executing on sensor k's value.

Protocol 2.3: State Consolidation Stage

- **if** $\mathit{MyAccept}_{\mathit{SelectedSource}} = \texttt{True}$ **then**:
  $\mathit{SavedState} \leftarrow \mathit{TempState}_{\mathit{SelectedSource}}$
- Discard $\mathit{TempState}_k$ for all $k = 1, ..., m$

LEMMA 2.4. At least $n - 2f$ non-faulty replicas have $\mathit{MyAccept}_{\mathit{SelectedSource}} = \texttt{True}$ and have the same *SavedState*. At most f non-faulty replicas have $\mathit{MyAccept}_{\mathit{SelectedSource}} = \texttt{False}$ (i.e., do not have the state).

Proof. By Lemma 2.2, all non-faulty replicas set *SelectedSource* to the same sensor k from set $\mathcal{C}$. A sensor k is only in $\mathcal{C}$ if $\mathit{Candidate}_k$ = `True`, which means $\geq n-f$ replicas Byzantine broadcasted the same value v from that sensor to all replicas in the Agreement Stage. Of these replicas, $\leq f$ may be faulty. Thus, $\geq n - 2f$ non-faulty replicas started with $\mathit{MyValue}_k = v$, and thus set $\mathit{MyAccept}_k \leftarrow \texttt{True}$. Moreover, these $\geq n - 2f$ non-faulty replicas all speculatively executed on v (since they started with it). Since they all perform the same computation on v in the same state (shown later), they produce the same state $\mathit{TempState}_k$, which they then store in *SavedState*.

Now we prove the second part of the lemma. Above we said $\geq n - 2f$ non-faulty replicas possess the same *SavedState*. Since there are $\leq f$ faulty replicas and n total replicas, there are $\geq n - f$ non-faulty replicas, and thus at most $(n - f) - (n - 2f) = f$ non-faulty replicas do not have the state. □

2.4.1.4 State Dispersal Stage

At the end of the Stage Consolidation stage, at least $n-2f$ non-faulty replicas possess the final state and can deliver results to the actuators. However, up to f non-faulty replicas still do not possess the updated state. The purpose of the State Dispersal Stage is to provide the updated state to those replicas. Importantly, this state dispersal process happens simultaneously with sending results to the actuators, so does not contribute to the end-to-end latency. Moreover, it can be overlaid with

the process of receiving sensor data in the next iteration of the protocol, as long as the state is finished updating before the next speculative executions begin.

Protocol 2.4 describes the State Dispersal Stage. *States* is an n-dimensional vector used to store the *SavedState* from each replica. All elements are initialized to $\perp$.

Protocol 2.4: State Dispersal Stage

- **if** $MyAccept_{SelectedSource} =$ `True` **then**:
  Broadcast *SavedState* to all replicas
- **if** $MyAccept_{SelectedSource} =$ `False` **then**:
  States[j] $\leftarrow$ the state broadcasted by replica j (if any)
  SavedState $\leftarrow$ the majority of the non-$\perp$ states in *States*

The correctness of the protocol will be shown in Theorem 1.

2.4.1.5 Deliver Outputs

At the same time as the State Dispersal Stage, the replicas deliver outputs to the actuators. The protocol is shown in Protocol 2.5. The actuators use a majority vote of the outputs to resolve the final system output.

Protocol 2.5: Deliver Outputs

- **if** $MyAccept_{SelectedSource} =$ `True` **then**:
  Reference *SavedState* to determine the output
  Send the output to actuators

We now prove the correctness of the overall protocol.

THEOREM 2.1. Given there are $n > 3f$ replicas and $m > 2g$ sensors: (1) the system always operates on correct sensor data, (2) all non-faulty replicas obtain the same correct state, and (3) all non-faulty actuators obtain the correct system output.

Proof. We first consider condition (1). If the Source Selection Stage selects a value from a non-faulty sensor, (1) is trivially satisfied. Now we consider the case where the selected value comes from a faulty sensor.

By Lemmas 2.2 and 2.3, all non-faulty replicas are guaranteed to agree on $\mathcal{C}$, and $\mathcal{C}$ must include all values sent from non-faulty sensors. Since there are $m > 2g$ sensors, this means that $\mathcal{C}$ contains $\geq (2g+1) - g = g + 1$ non-faulty sensor values and up to g faulty sensor values. Thus, as long as a source selection algorithm that tolerates a minority of the sensors being faulty is used,

the input to the non-faulty replicas is guaranteed to be correct. For example, in the common case, where source selection is a mid-value selection [132, 26, 71], a faulty sensor value that is selected is guaranteed to be upper and lower bounded by at least one non-faulty sensor value on each side. In other words, the selected value must fall within the range of values from non-faulty sensors.

We now consider (2). Let $\mathcal{S}_{\text{happy}}$ be the set of non-faulty replicas for which $\mathit{MyAccept}_{\mathit{SelectedSource}}$ $=$ `True` at the end of the State Consolidation Stage, and $\mathcal{S}_{\text{sad}}$ be the set of non-faulty replicas $\notin$ $\mathcal{S}_{\text{happy}}$. By Lemma 2.4, $|\mathcal{S}_{\text{happy}}| \geq n - 2f$, and all replicas $\in \mathcal{S}_{\text{happy}}$ have the same *SavedState* s, which they obtained by executing on the (correct, as shown above) value from the selected sensor. In the State Dispersal Stage all replicas $\in \mathcal{S}_{\text{happy}}$ broadcast s to all replicas, and replicas $\in \mathcal{S}_{\text{sad}}$ do not broadcast their states. Thus, at the end of the State Dispersal Stage, all replicas $\in \mathcal{S}_{\text{sad}}$ possess $\geq n - 2f$ copies of s from non-faulty replicas, and at most f states $s' \neq s$ from faulty replicas. Since $n > 3f, n - 2f > f$. Thus s must be the majority of non-$\perp$ states held by replicas $\in \mathcal{S}_{\text{sad}}$.

The correctness of (3) follows from the logic in (2). All replicas $\in \mathcal{S}_{\text{happy}}$ have the same correct *SavedState*, as shown above. Thus, referencing *SavedState* produces the same correct output. Each replica $\in \mathcal{S}_{\text{happy}}$ sends the output to the actuators, and replicas $\in \mathcal{S}_{\text{sad}}$ do not send an output. Since $|\mathcal{S}_{\text{happy}}| \geq f + 1$, each actuator gets $\geq f + 1$ correct outputs and $\leq f$ outputs from faulty replicas. Thus the majority of the outputs received by the actuators must be correct. □

2.4.2 Scaling Eager Execution for the Multi-Fault Case

The protocol we described above is fast when tolerating a single fault. However, it requires replicas to broadcast full copies of the sensor data they receive using a Byzantine agreement protocol. As we show in §2.6.1, this results in high latency when tolerating multiple faults, since large messages sent in later rounds of the agreement protocol need to be fragmented into multiple frames.

The multi-fault version of IGOR fixes this problem using a *binary reduction* technique [125, 126, 127, 128]. The idea is to add an extra stage, which we call *Filtering*, before the Agreement Stage. The Filtering Stage ensures that no two non-faulty replicas can accept different data from the same sensor, thus reducing the agreement on each sensor value to agreement on a single bit (accepted value, or did not accept value). Since the size of the data to agree on is reduced, fragmentation (and the resulting latency) is reduced as well.

An outline of the revised protocol is shown in Figure 2.3 (right picture). The Filtering Stage, as a well as a new Agreement Stage, replace the Agreement Stage in the earlier protocol optimized for single faults. The other stages (State Dispersal, Consolidation, etc.) remain the same. Below, we describe the new Filtering and Agreement Stages in detail.

2.4.2.1 Filtering Stage

The protocol for the Filtering Stage is shown in Protocol 2.6, which the replicas execute on each sensor value. The output of the protocol is a single bit indicating whether the value is accepted (`True`) or not accepted (`False`).

MyValue$_k$ is the value the replica received from sensor k. If no value was received, it is set to $\perp$ (missing). *Values*$_k$ is an n-dimensional vector used to store values other replicas received from sensor k. All elements are initialized to $\perp$. *MyAccept*$_k$ is a variable indicating whether to accept the value from sensor k. It is initialized to `False`.

Protocol 2.6: Filtering Stage

- **if** *MyValue*$_k \neq \perp$ **then**:
  Send *MyValue*$_k$ to all replicas
- *Values*$_k$*[j]* $\leftarrow$ the value sent from replica j (if any)
  if $\geq n - f$ non-$\perp$ values in *Values*$_k$ match *MyValue*$_k$ **then**:
  MyAccept$_k \leftarrow$ `True`

After the Filtering Stage, we have the following guarantees:

LEMMA 2.5. If sensor k is non-faulty and sends v, all non-faulty replicas set *MyValue*$_k \leftarrow v$ and *MyAccept*$_k \leftarrow$ `True`.

Proof. Since sensor k is non-faulty, all non-faulty replicas receive v and store it in *MyValue*$_k$. Then, all non-faulty replicas forward v to all replicas. Since there are $\geq n - f$ non-faulty replicas, all non-faulty replicas receive $\geq n - f$ values that match *MyValue*$_k$ and set *MyAccept*$_k \leftarrow$ `True` □

LEMMA 2.6. *MyValue*$_k$ is the same for all non-faulty replicas that set *MyAccept*$_k \leftarrow$ `True`.

Proof. Suppose two non-faulty replicas R_i and R_j accept different values from sensor k. That means each replica received $\geq n - f$ values from distinct replicas that match its own value. Any two sets of $n - f$ replicas intersect at $\geq 2(n - f) - n = n - 2f$ replicas. That means $\geq n - 2f$ of the matching values for R_i and R_j came from the same replicas. Since $n > 3f$, at least one of those replicas must be non-faulty. A non-faulty replica sends the same values to all replicas. Thus, R_i and R_j accepted the same value, which is a contradiction. □

2.4.2.2 (Revised) Agreement Stage

At the end of the Filtering Stage, each replica chose to accept or reject the value from each sensor. In the Agreement Stage, the replicas use this information, along with values leftover from the

Filtering Stage, to agree on (1) which sensors are candidates for source selection and (2) what values those sensors sent.

The Agreement Stage is shown in Protocol 2.7. The replicas run a separate instance of the protocol for each sensor. The protocol uses any existing BA protocol as a primitive [129, 46, 130]. *Accepts*$_k$ is an n-dimensional vector used to store *MyAccept*$_k$ bits from other replicas. All elements are initialized to $\perp$. *Candidates*$_k$ is a variable indicating if sensor k is a candidate for source selection. It is initialized to `False`.

Protocol 2.7: (Revised) Agreement Stage

- Broadcast *MyAccept*$_k$ to all replicas using a BA protocol
- *Accepts*$_k$*[j]* $\leftarrow$ the bit broadcasted by replica j (if any)

  if count(`True`) in *Accepts*$_k$ $\geq n - f$ **then**:

  Candidate$_k$ $\leftarrow$ `True`

  if *MyAccept*$_k$ $=$ `False` **then**:

  For all j where *Accepts*$_k$*[j]* $=$ `False`, *Values*$_k$*[j]* $\leftarrow \perp$

  MyValue$_k$ $\leftarrow$ the most common non-$\perp$ value in *Values*$_k$

The rest of the stages are the same as in the single-fault optimized protocol. Therefore, to establish the correctness of the multi-fault protocol, we only need to re-prove a few of the lemmas from §2.4.1.

(REVISED) LEMMA 2.1. For any sensor k, *Candidate*$_k$ is the same for all non-faulty replicas. Further, if *Candidate*$_k$ is `True`, *MyValue*$_k$ is the same for all non-faulty replicas.

Proof. Each bit in *Accepts*$_k$ was broadcasted with a BA protocol, and thus is the same for all non-faulty replicas. Hence, either all non-faulty replicas set *Candidate*$_k$ $\leftarrow$ `True` or they all keep it `False`.

Now we prove the second part of the lemma. Let $\mathcal{S}_{\text{happy}}$ be the set of non-faulty replicas that set *MyAccept*$_k$ to `True` in the Filtering Stage. Let $\mathcal{S}_{\text{sad}}$ be the set of all non-faulty replicas $\notin \mathcal{S}_{\text{happy}}$. By Lemma 2.6, *MyValue*$_k$ is the same for all $R_i \in \mathcal{S}_{\text{happy}}$. Call this value v. Each $R_i \in \mathcal{S}_{\text{happy}}$ sent v to all replicas in the Filtering Stage and broadcasted `True` in the Agreement Stage. Since *Candidate*$_k$ is `True`, $|\mathcal{S}_{\text{happy}}| \geq (n - f) - f = n - 2f$. Thus, at the end of the stage, each $R_i \in \mathcal{S}_{\text{sad}}$ has $\geq n - 2f$ copies of v in *Values*$_k$. Also, since all $R_i \in \mathcal{S}_{\text{sad}}$ broadcasted `False`, no $R_i \in \mathcal{S}_{\text{sad}}$ possesses a non-$\perp$ value from an $R_j \in \mathcal{S}_{\text{sad}}$. Thus, for each $R_i \in \mathcal{S}_{\text{sad}}$, any $v' \neq \perp$ that is not v came from a faulty replica. Since $\leq f$ replicas are faulty, each $R_i \in \mathcal{S}_{\text{sad}}$ has $\leq f$ such v' values. Since $n > 3f$, we know $|\mathcal{S}_{\text{happy}}|$, which is at least $n - 2f$, is $> f$. Thus, for all $R_i \in \mathcal{S}_{\text{sad}}$, v is the most common non-$\perp$ value in *Values*$_k$. □

(REVISED) LEMMA 2.3. If some sensor k is non-faulty and sends value v, then set $\mathcal{C}$ contains sensor k (and its value v).

Proof. If sensor k is non-faulty and sends v, then by Lemma 2.5, all non-faulty replicas set $\mathit{MyValue}_k \leftarrow v$ and $\mathit{MyAccept}_k \leftarrow \texttt{True}$. All non-faulty replicas then broadcast `True` with a BA protocol, which each non-faulty replica stores in $\mathit{Accepts}_k$. As there are $\geq n - f$ non-faulty replicas, $\mathit{Accepts}_k$ contains $\geq n - f$ `True` bits for all non-faulty replicas. Thus, all non-faulty replicas set $\mathit{Candidate}_k \leftarrow \texttt{True}$, and $\mathcal{C}$ contains (k, v). □

(REVISED) LEMMA 2.4. At least $n - 2f$ non-faulty replicas have $\mathit{MyAccept}_{\mathit{SelectedSource}} = \texttt{True}$ and have the same *SavedState*. At most f non-faulty replicas have $\mathit{MyAccept}_{\mathit{SelectedSource}} = \texttt{False}$ (i.e., do not have the state).

Proof. By Lemma 2.2, all non-faulty replicas set *SelectedSource* to the same sensor k from $\mathcal{C}$. A sensor k is in $\mathcal{C}$ only if $\mathit{Candidate}_k = \texttt{True}$, which means $\geq n - f$ replicas claimed to set $\mathit{MyAccept}_k \leftarrow \texttt{True}$. Up to f of these replicas are faulty, so $\geq n - 2f$ non-faulty replicas actually did set $\mathit{MyAccept}_k \leftarrow \texttt{True}$. By Lemma 2.6, these $\geq n - 2f$ non-faulty replicas all had the same $\mathit{MyValue}_k$ at the start of the protocol, which they speculatively executed on. Since the replicas perform the same computation on the same value in the same state, they produce the same state $\mathit{TempState}_k$, which they store in *SavedState*.

We next prove the second part of the lemma. As discussed above, $\geq n - f$ replicas claimed to set $\mathit{MyAccept}_{\mathit{SelectedSource}} \leftarrow \texttt{True}$. In the worst case, all replicas which did not claim to set $\mathit{MyAccept}_{\mathit{SelectedSource}} \leftarrow \texttt{True}$ are non-faulty. Thus, at most f non-faulty replicas do not have the state. □

2.5 Prototype Implementation

To evaluate our solution, we built a prototype of IGOR in NASA's Core Flight System (cFS) [87], an open-source software framework used in a variety of real spacecraft [88] (and planned to be used in several future NASA missions [133]). Overall, our prototype consists of 5976 lines of C code.

Our prototype runs on a cluster of Raspberry Pi (RPi) 3B+ computers with 1.4 GHz ARM Cortex-A53 processors. We chose RPis as they use the same processor as NASA's upcoming High Performance Spaceflight Computing (HPSC) chiplet [14] (though the HPSC will have twice as many cores). Each RPi runs Raspbian 9.4 with kernel version 4.14.34 and the PREEMPT_RT patch, with dynamic clock scaling disabled to improve real-time performance. The RPis schedule tasks using the cFS scheduler [105], a cyclic executive that triggers tasks to run in predefined

time slots of a periodic schedule. The RPis' schedulers are synchronized to a common external timing circuit. To minimize timing variability, one core of each RPi is reserved for inter-replica communication, and the remaining three cores are used for task execution.

All RPis communicate through a gigabit Ethernet switch. We implemented a software layer in cFS to emulate the AFDX [31] network found in typical aircraft [50]. The worst-case Ethernet frame latencies measured in our prototype are a couple of milliseconds, which are similar to those observed in real AFDX networks [57, 96]. Messages that are sent to the same destination in the same round are batched whenever possible to reduce overhead.

For comparison, we also implemented two state-of-the-art systems, namely OM and TC. OM is an agree-execute system based on Lamport, Shostak, and Pease's Oral Messages agreement protocol [46, 92], which has the theoretical minimum number of rounds [27] and is used extensively in practice [62, 63, 61, 64, 42]. OM is also the basis for all existing Byzantine agreement protocols that meet this lower bound [45, 134, 135]. TC is an agree-execute system based on Turpin and Coan's reduction protocol [125], which uses a similar reduction approach to IGOR and takes fewer rounds than any existing protocol that uses a binary reduction [136, 128, 137, 126]. We used OM as the binary agreement primitive in TC, as well as in our IGOR prototype. Lastly, we also implemented a system without replication, NOREP, which provides the theoretical minimum latency.

2.6 Evaluation

To evaluate IGOR's performance and practical applicability, we conducted a series of experiments on our prototype. We had four key questions for our evaluation: (1) How effective is IGOR in reducing end-to-end latency? (2) How much can IGOR improve schedulability? (3) What are IGOR's computation and communication overheads? and (4) How well does IGOR perform in a real spaceflight application?

2.6.1 Latency

Experimental setup. For this experiment, we considered a range of workload parameters based on practical aircraft systems. Specifically, the sensor data size was distributed in the range {250, 750, 1250} bytes, and the task's worst-case execution time (WCET) was in {5, 10, 15, 20} milliseconds; these values were chosen to match those found in typical aircraft [107, 57]. (Note that the sensor data size is relatively large, since data delivered to the flight computers are often batched by downstream devices [59, 132].) We set the actuator data size, task's state size, and source selection time to be 500 bytes, 1500 bytes, and 1 ms, respectively, based on NASA's Orion Ascent Abort-2

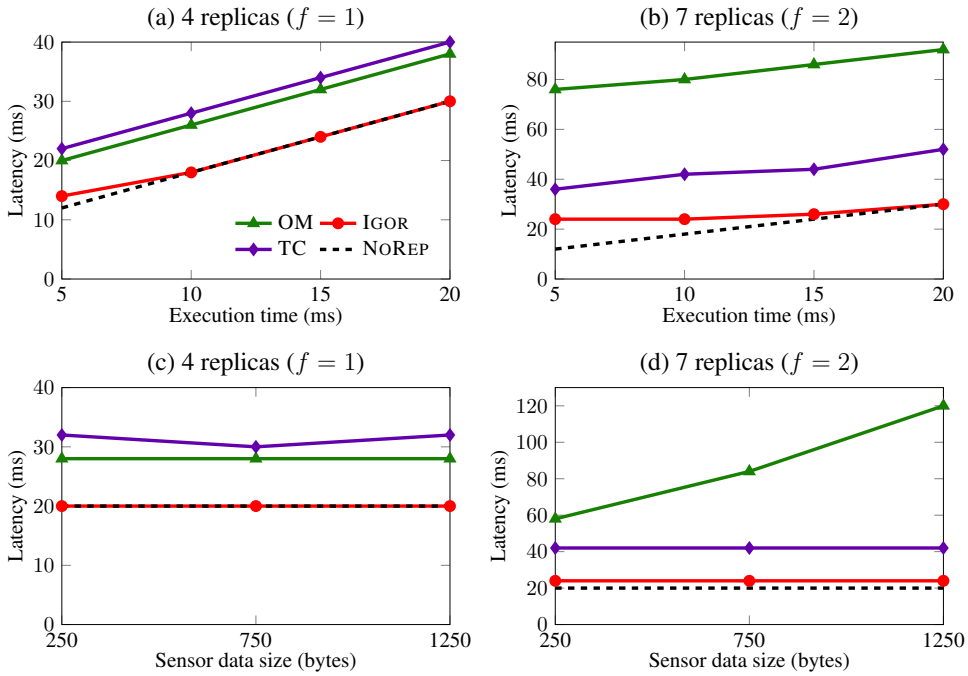

Figure 2.4: **IGOR's Latency** — End-to-end latencies of IGOR and other state-of-the-art protocols compared to a non-replicated system. Plots (a) and (b) use 750 bytes of sensor data with 3 sensors. Plots (c) and (d) use 12 ms execution times with 3 sensors.

system used in our case study (§2.6.5). The number of sensors was 3, the typical redundancy level found in aircraft and spacecraft [40]. The band allocation gap (BAG) – i.e., the minimum time between AFDX frames sent on the same virtual link – was 1 ms, which is the minimum allowed by the AFDX standard [31].

The workloads executed on the experimental platform described in §2.5. We used a 500 Hz cFS schedule (2 ms per slot), the highest rate that can be achieved with our timing circuit (and also the upper limit of what is typically seen in practice [95, 105, 138]). One of the RPis acted as the sensors and actuators, and the remaining RPis were used as replicas. We considered two fault settings: $f = 1$, with 4 RPis serving as replicas; and $f = 2$, with 7 RPis serving as replicas.

To determine the schedule table for each system (IGOR, OM, TC, NOREP), we first measured the maximum time required to execute each stage of the system (e.g., reading from sensors, agreement, and execution) over 100 iterations, then added 10% margin to each measured value. We then rounded each quantity up to the next time slot and scheduled the stages in sequence. Finally, we validated that the resulting schedule worked on our hardware cluster as expected (e.g., without causing any message drops or overrun of task's WCET). We measured the latency as the time between the instant the sensors were scheduled to transmit their data and the instant the actuators were scheduled to read outputs from the replicas.

Results. Figure 2.4 shows the results of all four systems under each fault setting, as we varied the

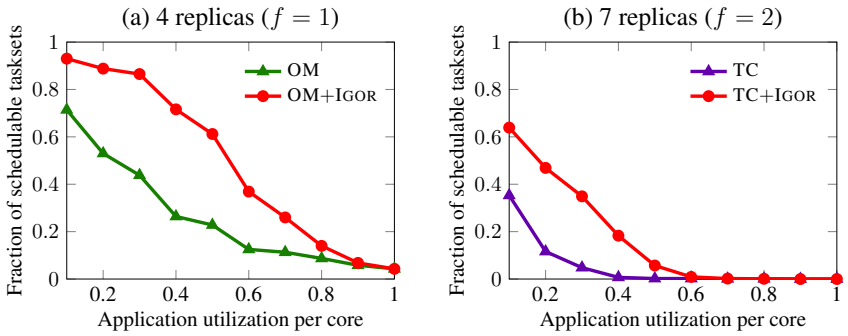

Figure 2.5: **IGOR's Schedulability** — Increase in schedulability when using IGOR. IGOR makes it possible to schedule significantly more tasksets than the state of the art.

task's execution time and sensor data size. As shown in the figure, IGOR achieves a latency very close to, or the same as, that of NOREP (the theoretical minimum) in all cases. For example, under the $f = 1$ setting, its latency is equal to NOREP's for all execution times of 10 ms and above. Moreover, IGOR's Filtering (§2.4.2) approach makes it especially fast when tolerating multiple faults: IGOR's latency *when tolerating 2 faults* is even lower than the latency of the best existing system (OM) *when tolerating 1 fault* for all execution times of 10 ms and above. The next fastest system (TC) has a 1.5–1.75× higher latency than IGOR's. Thus, IGOR not only delivers close-to-optimal latency but also substantially reduces latency compared to existing systems. Its benefits also increase with more faults.

2.6.2 Schedulability

Next, we evaluated whether IGOR could be used to improve the schedulability of a BFT SMR system.

Experimental setup. We considered application workloads consisting of independent constrained-deadline periodic BFT tasks, which are distributed over 3 cores on each replica (as on our RPis). We varied the workload utilization per core from 0.1 to 1, in steps of 0.1. For each utilization, we randomly generated 1000 tasksets. The tasks' WCETs were randomly selected from {5, 10, 15, 20} ms (the same range used in our latency experiments). The task periods were randomly selected from {200, 100, 50, 25} ms, which are commonly used in practice [104]. The tasks' deadlines (i.e., the maximum time allowed between reading a sensor input and producing a result) were uniformly distributed between their WCETs and periods. Notice that a task's deadline is also the maximum allowed time to finish all of the filtering, agreement, and execution stages.

We scheduled tasks using a common heuristic, where we organized tasks into rate groups and scheduled tasks with higher rates first [139]. Per cFS' design, tasks were scheduled without splitting into smaller sub-tasks. We used 2.5 ms slots to accommodate the tasks' periods. We deter-

mined the WCET for each protocol stage, such as filtering and agreement, from our earlier latency experiment.

We assumed sensor data is available whenever the BFT tasks expect it; in other words, the network imposes no additional constraints on the scheduling. As in our prototype, all inter-replica communication is handled by a separate core. Since we focused on the schedulability of execution tasks on the replicas, to simplify the analysis, we assumed that the source selection and the communication with sensors/actuators take negligible processor time; however, our results should apply to the general setting as well.

For each schedule, we determined whether the fastest existing BFT protocol (OM for $f = 1$, TC for $f > 1$) can feasibly schedule all BFT tasks. If not, we used IGOR instead for those tasks that were unschedulable (while still using the fastest existing protocol for tasks that were schedulable), and we checked whether the system would then become schedulable. For each task scheduled by IGOR, we replaced it with three speculative copies (assuming 3 redundant sensors). The IGOR tasks were scheduled in the same time slots on all 3 cores, at the start of their respective rate group. Besides meeting deadlines, IGOR tasks were also required to complete state dispersal by the end of their periods.

Results. Figure 2.5 shows our results for the two cases: when the best protocol is used alone, and when it is used in conjunction with IGOR. As expected, as the utilization increases, the fraction of schedulable tasksets also decreases for both cases. Notice that, in our workloads, tasks' deadlines can be much smaller than their periods, which explains the large drop in the number of schedulable tasksets at higher utilizations.

The results show that IGOR is able to substantially increase the number of schedulable tasksets, compared to using the best existing protocol alone: it is able to schedule 1.88× and 3.22× more tasksets under the single-fault and two-fault settings, respectively. This demonstrates that, even with the potential computation overhead for speculative execution, IGOR's efficiency in reducing overall latency also results in a substantial increase in the schedulability of the overall system.

2.6.3 Computation Overhead

Experimental setup. To evaluate IGOR's computation overhead, we repeated the same experiment as in §2.6.2. For each BFT taskset, we calculated the *remaining* available CPU capacity per core after having scheduled the taskset, and we report the average across all tasksets that were schedulable at each workload utilization. (Note that we excluded unschedulable tasksets, as we focus on hard real-time systems and hence a taskset is only accepted to run if it is schedulable.) As in our schedulability evaluation, our goal was to compare between (1) a system that used the fastest existing protocol on its own, and (2) a system that used the fastest existing protocol together with

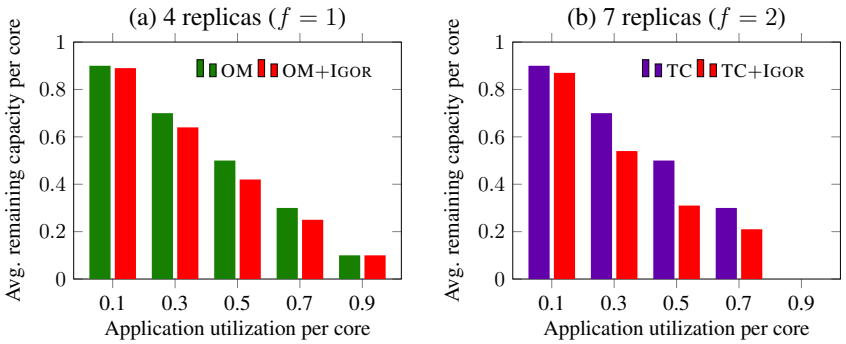

Figure 2.6: **IGOR's Computation Overhead** — Available compute capacity when using IGOR compared to the best existing protocol alone.

IGOR, where IGOR was used for the tasks that could not meet deadlines in (1).

Results. Figure 2.6 shows our results. In the figure, a higher remaining capacity corresponds to a higher resource use efficiency and thus a smaller overhead. The difference between the green/purple column (case 1) and the red column (case 2) represents the computation overhead added by IGOR.

When tolerating 1 fault, using IGOR results in only 1.1–20% reduction in remaining average capacity than a system that uses only OM. When tolerating 2 faults, using IGOR results in a slightly higher overhead, at 3.3–38% lower average remaining capacity. Note, however, that the computation costs in both fault settings are reasonably small compared to IGOR's substantial improvements in latency and schedulability reported in Figures 2.4 and 2.5.

2.6.4 Communication Cost

Experimental setup. Network bandwidth is often at a premium in real-time embedded systems. To evaluate IGOR's bandwidth usage, we sniffed all traffic entering the Ethernet switch and counted the total number of bytes (including Ethernet headers) over 100 iterations of each of the four systems (IGOR, OM, TC, and NOREP) on our prototype. Our AFDX layer uses Ethernet broadcast to emulate VL multicasting in AFDX networks, so each frame broadcasted by a given RPi was counted only once. We used 3 sensors in our tests, each generating 750 bytes (both numbers are the defaults in our latency evaluation).

Results. Figure 2.7 shows our results. In general, IGOR communicates roughly the same number of bytes as the existing state-of-the-art systems in both the single- and two-fault settings. The reason for IGOR's slightly higher communication cost is its need to distribute the state after each execution. IGOR's bandwidth usage could be reduced by dispersing only *deltas* from the previous state, or by not transferring parts of the state that are known not to change based on the sensor data being processed. Both of these techniques are commonly used in existing BFT protocols [79], and

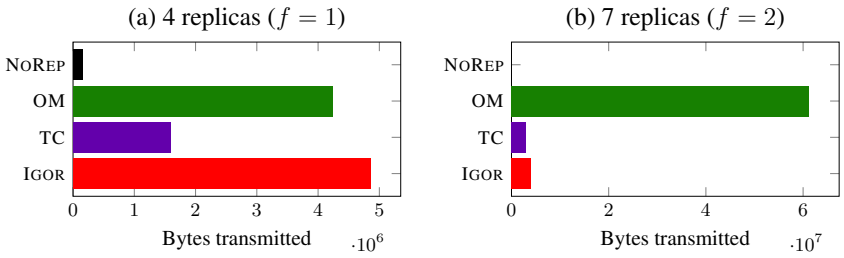

Figure 2.7: **IGOR's Communication Cost** — Total bytes transmitted in IGOR and other state-of-the-art protocols over 100 iterations.

Figure 2.8: **IGOR Spaceflight Simulation** — Orion AA-2 simulation moments after the crew module (left) separates from the launch abort system (right).

could be integrated into our prototype.

The reason for IGOR's high communication efficiency in the multi-fault case is its use of a Filtering Stage. When minimizing network bandwidth is a priority, the Filtering Stage could also be used in the single-fault case. However, this would result in a slightly higher latency (about 2 ms in our tests).

2.6.5 Case Study: Orion Ascent Abort-2

To evaluate how well IGOR performs in a real spaceflight application, we conducted a case study of NASA's Ascent Abort-2 (AA-2) flight test. The AA-2 test was performed in 2019 to exercise the launch abort system (LAS) for Orion, a spacecraft intended to take astronauts to lunar orbit. The purpose of the LAS is to pull the spacecraft away from the rocket if an emergency happens during ascent. In the AA-2 test, the LAS was intentionally activated, and it carried the spacecraft away and jettisoned the craft into the ocean.

We ported a simulation of the Orion guidance, navigation, and control software (GNC) to 4 RPis in our cluster. The software included multiple genuine Orion flight software components, including code for absolute navigation, optical navigation, propellant balancing, and abort functionality [95]. The software ran against a high-fidelity simulation, which models the vehicle's trajectory, as well as the sensors and actuators. Figure 2.8 shows a screenshot of the simulation.

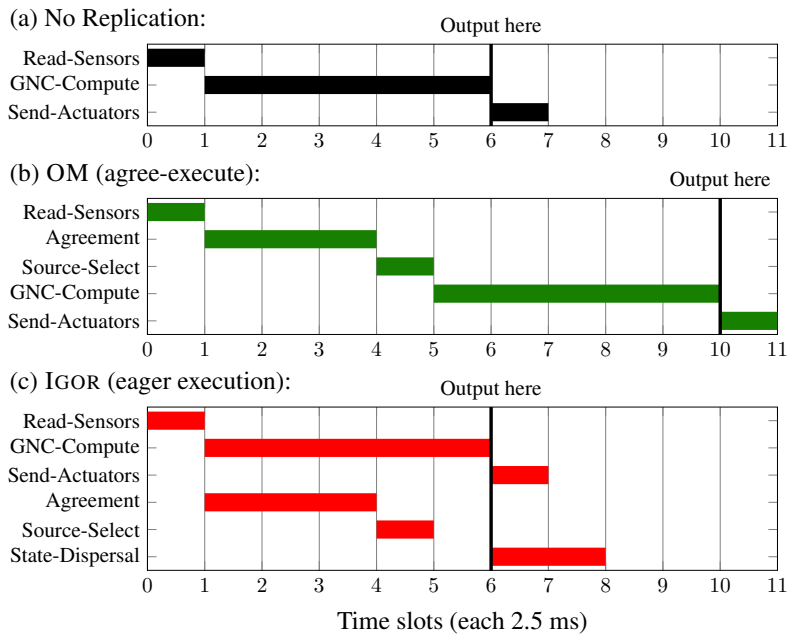

Figure 2.9: **Spacecraft Task Scheduling** — Time slot allocations for IGOR and OM when running the AA-2 flight software in the single-fault case.

The GNC software ran in a 40 Hz control loop. Every 25 ms, it read sensor data (e.g., from inertial measurement units, barometric altimeters), performed GNC computations, and commanded the actuators. We batched the sensor data into three redundant groups, each 772 bytes, to simulate batching from remote interface units or data concentrators. The GNC computations took roughly 9.6 ms and maintained 1304 bytes of internal state. The actuator data totaled 376 bytes.

We determined the schedule using the same process as in our latency evaluation (e.g., measured the worst-case execution time, added 10% margin), except that we used 2.5 ms time slots to accommodate the 40 Hz control loop. State consolidation was performed before sending to the actuators.

Our results are shown in Figure 2.9. As expected, IGOR adds no additional latency compared to a non-replicated system. In contrast, OM (the fastest existing protocol) adds 10 ms of latency. Therefore, in order to run at a 40 Hz rate, OM has to overlap sending to the actuators with reading from the sensors. This means that feedback from the output of a given GNC execution cannot be incorporated into the inputs of the next execution, which causes a noticeable reduction in vehicle stability. This was observed during our experiment, where the LAS tower rocked back and forth after igniting instead of traveling in a straight path. The results demonstrate that IGOR's ability to minimize latency not only improves schedulability but also enables much better control performance and system stability compared to state-of-the-art techniques.

2.7 Related Work

Speculation to avoid agreement. Several BFT SMR protocols designed for data centers use speculation to avoid the overhead of executing an agreement protocol [81, 85, 83, 84, 82]. In these systems, replicas execute client requests directly and are assumed to produce consistent states. If state divergence occurs, then the system rolls back to a previous state and repeats the execution. This approach makes sense in non-real-time systems that prioritize graceful executions over the worst case. However, it is a poor fit for real-time systems, which require low latency in all executions. IGOR's speculation is completely different in that it is *eager* rather than *predictive*. This way, IGOR gets the benefits of speculation in all executions, albeit at the expense of extra computation.

Overlapping agreement and execution. Several protocols use speculation as a way to overlap agreement with executing client requests [140, 141, 142, 143, 144]. In these systems, replicas execute while an agreement protocol is run in the background. If the agreement protocol detects inconsistency, the system rolls back and the computations are repeated. IGOR also uses speculation to overlay agreement and execution. However, since IGOR's speculation is not predictive, there is no need to rollback. Other predictive protocols avoid the need for rollback by generating an optimistic (possibly incorrect) result quickly, and a guaranteed correct result after some delay [145]. Unlike these systems, IGOR never exposes incorrect results to the actuators, and it uses speculation to reduce the latency of generating a guaranteed correct result (and not just an optimistic one).

Multi-core state machine replication. Several solutions increase the performance of SMR on multi-core processors by allowing replicas to execute independent requests in parallel [146, 147, 79, 148, 149, 150, 151]. This approach can greatly increase throughput when requests are mostly independent, which is important for web services with millions of clients. However, it does not significantly reduce the latency of executing individual requests. In general, IGOR is orthogonal to these works, and there is no reason IGOR could not be extended to also parallelize independent executions. Execute-Verify [79] systems allow multi-core replicas to execute dependent requests nondeterministically; however, they require replicas to run an agreement protocol afterwards to detect state divergence. In contrast, IGOR assumes deterministic execution, and parallelizes execution and agreement.

Byzantine extension protocols. Several Byzantine agreement protocols are built as extension protocols, i.e., they use techniques that reduce the problem of agreeing on arbitrarily large values to that of agreeing on a small number of bits [152, 153, 154, 47, 128, 127, 155, 156, 157, 158, 159, 160, 125, 126]. As a result, these protocols can often achieve low communication complexities when the value to agree on is sufficiently large. IGOR also uses a reduction-based technique, but for the purpose of reducing latency by preventing message fragmentation. To our knowledge, IGOR's Filtering and Agreement Stages take fewer rounds than any other reduction-based Byzan-

tine agreement protocol [125].

Non-equivocation. Several protocols use a combination of cryptography and trusted hardware to restrict a faulty device's ability to send conflicting information to other devices (i.e., equivocate), thus making agreement less expensive [158, 161, 162, 163, 164, 165, 166]. Other protocols prevent equivocation by having devices echo values they receive to one another, and make decisions based on a quorum of matching echoes [167, 168, 114, 169, 170]. IGOR also uses the idea of echoing values in its Filtering Stage to prevent faulty sensors from equivocating, without relying on cryptography or trusted hardware assumptions.

2.8 Conclusion

This paper presented IGOR, a new speculative BFT SMR approach that leverages multi-core processors to achieve low latency in both the presence and absence of faults. IGOR provides systems a means of meeting tight deadlines that would otherwise be impossible with classical BFT SMR approaches. Our experiments show that IGOR achieves up to 1.75× lower latency than the state of the art, often matching the latency of a non-replicated system, and improves the schedulability of BFT tasks by 1.88–3.22×. We show that IGOR has immediate benefits when used for a real space-flight application, and we believe it is broadly applicable to other BFT systems seeking improved real-time control performance.

CHAPTER 3

CROSSTALK: Making Low-Latency Fault Tolerance Cheap by Exploiting Redundant Networks

3.1 Introduction

Real-time control systems perform a variety of critical functions in the modern world, including flight control in spacecraft [28] and aircraft [26], and monitoring and reactor safing in nuclear plants [44]. A common way that these systems tolerate faults, such as memory corruptions in nodes [58] and stuck bits in network cards [123], is with Byzantine fault-tolerant (BFT) state machine replication (SMR) [28, 62, 63, 61, 73, 40, 26, 64]. In BFT SMR, the same function is performed simultaneously by multiple nodes (or *replicas*) and their outputs are compared. To ensure the non-faulty replicas have the same state (and can thus out-vote faulty replicas), it is necessary to ensure they all operate on identical inputs [26]. Traditionally, this is done by having the replicas run an *agreement protocol* on all sensor data they receive before each execution [28]. An example of a traditional BFT SMR protocol is shown in Figure 3.1a.

One problem with traditional BFT SMR protocols is that reaching agreement on sensor data is *slow*. It is well known that replicas must exchange messages for at least $f + 1$ communication rounds to reach agreement if f replicas may be faulty [27]. If the replicas are not co-located, as is typical in order to tolerate scenarios like fires or attacks that could disable all replicas [171, 172], these messages must traverse several network hops in each round, resulting in significant added latency. Moreover, additional latency comes from the need for tasks on the replicas to read, process, and send messages between rounds [28]. Overall, the latency of ensuring agreement can be a significant portion of the time needed to execute the overall BFT SMR protocol (e.g., 50% or more), making it difficult or even impossible to meet certain hard deadlines or successfully schedule BFT tasks [28].

A recent RTAS paper, IGOR [28], was designed to address this problem. Figure 3.1b shows an overview of IGOR. Instead of reaching agreement on sensor data *before* executing, replicas in IGOR reach agreement while execution on the sensor data is underway. Thus, in cases where

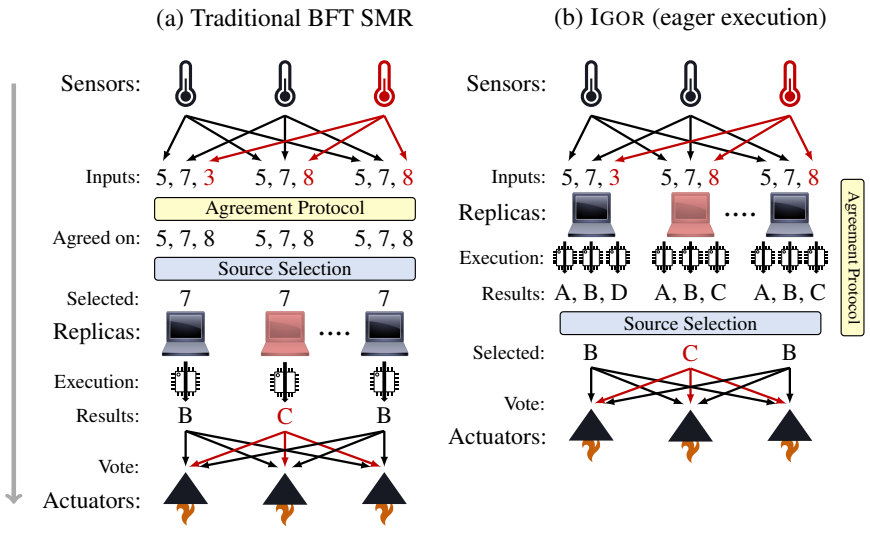

Figure 3.1: **Overview of Existing BFT SMR Protocols** — Example executions of a traditional BFT SMR protocol (left) and the state-of-the-art IGOR protocol (right). Sensor 3 and replica 2 are faulty. The network is not shown.

agreement takes less time than execution, IGOR's latency matches that of a non-BFT system (i.e., it is optimal). Unfortunately, while traditional protocols allow the replicas to down-select redundant sensor values to a *single* value before executing, IGOR requires replicas to simultaneously execute on data from *every* redundant sensor. This means that: (1) IGOR cannot be used on single-core processors (which are still common in practice [48]) and (2) IGOR's computation overhead is typically at least $3\times$ higher than traditional protocols. Moreover, IGOR requires $3f + 1$ replicas to tolerate f faults [28], while traditional protocols often only need $2f + 1$ [46].

We observe that the reason IGOR is so computationally expensive (and traditional protocols are so slow) is that, in order to be general, they make no assumptions about the network topology — instead treating the replicas as if they are interconnected by point-to-point channels [28]. As a result, they cannot exploit the redundancy and connectivity that often *already exist* in the network in order to increase performance.

We present CROSSTALK, a new BFT SMR protocol that minimizes latency (matching or beating IGOR) *without* requiring multi-core processors, *without* requiring any additional processing or replicas compared to traditional BFT protocols, and while remaining generally-applicable. The key insight is that embedded systems requiring BFT SMR are all moving towards or have already adopted the same general network architecture, which we refer to as a *redundant switched network* [173, 57, 174, 175]. In these networks, devices communicate through switches, and the whole network is replicated to form redundant *planes*. We observe that this design already provides enough redundancy and *nearly enough* connectivity to satisfy the "$f + 1$ round" [27] requirement

for agreement, merely as a consequence of the paths messages take through the network.

CROSSTALK works by taking these redundant switched networks that systems *already use in practice*, and adding a small number of new connections, or *cross-links*, between the planes. We show in §3.3.2 that by exploiting state-of-the-art techniques to restrict the behavior of faulty switches, it is possible to add these cross-links safely. With these cross-links in place, and by moving messages between the planes according to a specific algorithm, CROSSTALK can solve agreement *entirely in the network* as messages travel from the sensors to the replicas. In other words, agreement happens in a *single* round. Importantly, CROSSTALK does not require any non-standard capabilities in the switches, and thus can be used with existing network hardware and protocols.

One unique challenge of CROSSTALK's single-round design is tolerating timing faults, such as a sensor transmitting later than expected. In fact, we prove — to our knowledge, for the first time — that it is *impossible* to tolerate all timing faults with a single-round agreement protocol in synchronous systems. The main intuition is that, while the synchronous model assumes *bounds* on network latency, the *exact* latency is not known [176]. Thus, a message may arrive slightly before a deadline for some replicas (and thus be accepted) but after the deadline for others. In §3.4.4, we show how CROSSTALK can circumvent the impossibility result using network timestamps.

We developed a prototype of CROSSTALK for a NASA flight software framework [87] and evaluated it in a real avionics development lab. CROSSTALK achieved comparable or lower latency than the state of the art at a fraction of the computation cost, resulting in 2.13–4.24× better schedulability (§3.6.2). By requiring fewer replicas, CROSSTALK also reduced mass and cost (§3.6.3). We also executed simulated flight software from a real spaceflight mission and found that CROSSTALK met all deadlines while tolerating more faults than the state of the art and using nearly 3× less CPU time.

In summary, we make the following contributions:

- CROSSTALK: a novel BFT SMR protocol that exploits redundant switched network topologies that systems already possess to achieve low latencies without added computation costs (§3.4).
- A new proof on the impossibility of tolerating timing faults with single-round agreement protocols (§3.4.3) and a novel method for overcoming this result (§3.4.4).
- An experimental evaluation of CROSSTALK's performance and schedulability on a real avionics testbed (§3.6).
- A case study using CROSSTALK in a real spaceflight abort scenario (§3.6.5).

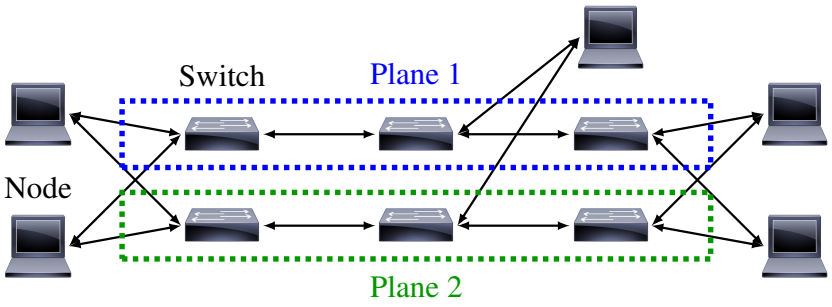

Figure 3.2: **Redundant Switched Network** — Example redundant switched network with two planes. Nodes here may be replicas, sensors, or actuators. Note that the topology *inside* a plane can take many different forms, including a star or ring.

3.2 Background: Redundant Switched Networks

Over the past decade, the need for higher bandwidths and increased scalability in embedded systems has driven designers to adopt switched networks. For example, aircraft have replaced ARINC 429 and 629 buses with Avionics Full-Duplex Switched Ethernet (AFDX) and Fibre Channel [53, 57]. Spacecraft have replaced MIL-STD-1553 buses with AFDX, Time-Triggered Ethernet, and SpaceFibre [28, 173, 53]. Trains and industrial control systems are replacing specialized fieldbuses with Time-Sensitive Networking (TSN) [174, 175].

Today, the most dominant strategy for tolerating switch faults in these systems is to replicate the entire network to form multiple *planes* (2–3 planes are typical) [11, 173, 57]. Each node is connected to all planes. To minimize latency, all planes are typically active simultaneously [31, 32, 175]. A node communicates by sending the same message over all planes, and the receiver accepts the first valid copy to arrive [31, 32]. Figure 3.2 depicts this *redundant switched network* topology.

Unfortunately, redundant switched networks are highly susceptible to Byzantine faults of nodes [71, 11]. For example, a faulty sensor can send different messages to different planes. Depending on which message happens to arrive first at each replica, different replicas may accept different messages [28]. Similarly, a faulty sensor may transmit two different messages on the same plane, and no messages on the other planes. If a switch in the plane is also faulty, it can fail to forward each message to different sets of replicas, causing disagreement.

As we will show, CROSSTALK exploits the redundant switched network topology systems already use to make masking these Byzantine faults fast and efficient.

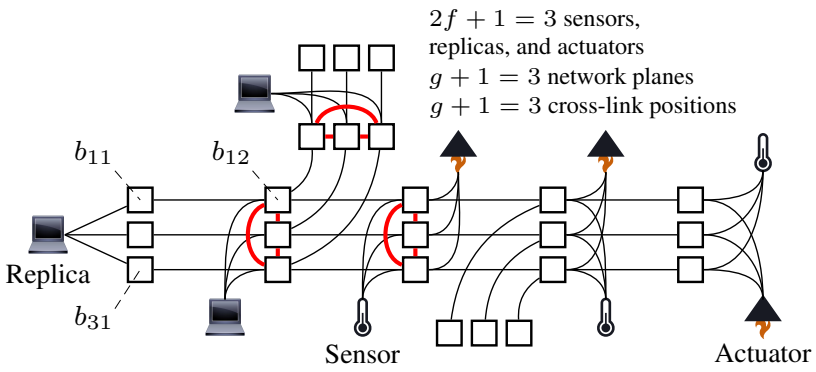

Figure 3.3: **CROSSTALK's System Model** — CROSSTALK's system model when $f = 1$ and $g = 2$. The squares are switches and the lines are links. The nodes can connect at any position on the network. The *only* thing we assume that systems in practice may not already contain are the cross-links (shown in red).

3.3 Models

3.3.1 System Model

Our system model is shown in Figure 3.3. The system consists of replicas, sensors, and actuators (collectively called *nodes*) connected to a redundant switched network, as described in §3.2. There are n network planes, and each plane has a unique identifier $\{1, \ldots, n\}$. We use b_{ik} to refer to the switch in plane i at position k. We assume the topologies of the planes are all identical, and that a given node connects to the planes at the same switch position on all planes (e.g., b_{1k} and b_{2k}). These assumptions are almost always satisfied in practice [57, 11]. CROSSTALK could also be extended to work in networks where these assumptions do not hold.

We assume the system is *synchronous*, meaning there are a priori known bounds on the time it takes nodes to execute tasks and for messages to traverse each switch. These assumptions are often satisfied using specialized real-time operating systems and deterministic switched networks [53, 57, 11]. As in past work [28, 58], we also assume nodes are synchronized within a bounded skew. A variety of fault-tolerant synchronization protocols could be used for this purpose [32, 177].

We assume messages take a pre-planned route from a sender, through the switches, and to one or more destination nodes. We refer to each such route as a *virtual channel*. Each message contains an identifier specifying its virtual channel. The switches are pre-programmed with a routing table indicating, for each message with a specific ID that arrives on a given ingress port, to which egress ports the message should be forwarded. If a message with ID x arrives on a switch port for which no route exists for ID x, the message is dropped. We use "sending on virtual channel x" to refer to transmitting a message containing the ID for virtual channel x, and "receiving on virtual channel

x" to refer to receiving a message with the ID for virtual channel x.

Cross-links between planes. The only somewhat non-standard assumption we make, which enables the CROSSTALK approach, is that switches at a subset C of switch positions in different planes are connected to one another. That is, for all $c \in C$, b_{ic} is connected to b_{jc} for any two planes i and j. We call these connections *cross-links* and the positions in C *cross-link positions*. We formalize the required number of cross-link positions in §3.3.2. The choice of C does not impact correctness, but does impact latency. We describe how to choose C to minimize latency in §3.4.

The required cross-links for CROSSTALK can be added to systems with low size, weight, and power impacts. The reason is that, as shown in Figure 3.3, cross-links are only added between redundant versions of the *same* switch (i.e., switches at the same position in different planes). Often these switches are co-located to reduce harness mass. As we will show in §3.6.3, CROSSTALK's cross-links add negligible mass and cost to the system. In fact, because CROSSTALK requires fewer replicas than the state-of-the art (IGOR [28]), CROSSTALK actually *reduces* mass and cost overall (§3.6.3).

3.3.2 Fault Model

We assume a *Byzantine* fault model for the nodes, where in a system with $2f + 1$ sensors, $2f + 1$ replicas, and $2f + 1$ actuators, up to f sensors, f replicas, and f actuators can deviate arbitrarily from the protocol ($\geq 2f + 1$ redundancy is needed to solve BFT SMR [28, 46]). This includes corrupting messages, sending different messages to each plane, and sending messages out of order [46, 28]. *Initially,* we assume faults do not occur in the time domain — i.e., messages from faulty nodes are generated in an a priori known bounded time [178]. This assumption is common in past work [58]. In §3.4.3, we relax this assumption to allow nodes to fail arbitrarily in the time domain.

The only restriction is that a babbling node cannot consume all the network bandwidth in order to prevent messages sent by other nodes from being delivered. This is almost always prevented in practice by using a bandwidth allocation system monitored and enforced by the switches [31, 33].

We consider a more restricted fault model for the switches, where in a system with $g+1$ network planes and at least $g + 1$ cross-link positions (i.e., $\geq (g + 1)^2$ switches have cross-links), up to g switches can be *omission* faulty [32]. This means faulty switches can fail to forward any messages they receive to any receivers, potentially resulting in some receivers getting a message and others not [32]. However, faulty switches cannot undetectably corrupt messages, create messages, or delay messages an unbounded amount of time [178, 32].

This is the standard switch fault model used in a variety of real BFT embedded systems in

practice [32, 177, 33]. In discussions with avionic systems designers, we found there are several reasons this is the case:

1. Switches are often implemented as self-checking pairs, with each switch containing two processors. The switch cannot send a message unless both processors produce the same message at the same time [71].
2. System designers use safety-layer protocols with features like layered CRCs, sequence number checking, and timestamp checking to minimize the probability of faulty switches undetectably altering or replaying messages sent by non-faulty nodes [179, 180].
3. Other critical parts of the system break if faulty switches can exhibit Byzantine behavior. For example, we are not aware of any BFT time synchronization protocol used in practice that tolerates more severe switch faults than omission [32, 177].

It is the omissive behavior of modern switches that makes adding cross-links between planes in CROSSTALK safe. Even if a switch is faulty, it has no way to consume excessive bandwidth or pass incorrect messages on the other planes.

Finally, we note that unlike CROSSTALK, most prior work on BFT SMR (including IGOR [28]) does not consider switch faults [46, 58]. Thus, CROSSTALK's fault model is strictly more general.

3.4 Design

This section describes CROSSTALK, a new low-cost BFT SMR protocol. CROSSTALK has two main goals: (1) *Low latency* — CROSSTALK aims to be fast in both the presence and absence of faults, and (2) *Low computation overhead* — CROSSTALK aims to be usable on single-core processors and to have minimal processing overhead. In contrast, the state of the art in low-latency BFT protocols has high overheads and requires multi-core processors [28].

The key idea of CROSSTALK, which allows it to achieve these goals, is to solve agreement *indirectly* as a consequence of routing messages between network planes. An overview is shown in Figure 3.4. ❶ First, each redundant sensor sends a message with its value to each plane. Rather than traveling directly to the replicas, the message is first routed through $g+1$ switches with cross-links (i.e., switches in C). Note that two sensors at different positions in the network may send their messages through the cross-link switches in different orders to minimize latency. Moreover, if $|C| > g+1$, each sensor's messages may travel through a *different* set of $g+1$ cross-link switches (i.e., C can be different from each sensor's perspective). ❷ As a message travels through a cross-link switch on a given plane k, the switch forwards a copy of the message to the "next" plane $k+1$. Each copy continues to traverse the cross-link switches, which in turn create copies and forward them to the next plane (until no such plane exists). ❸ After traversing $g+1$ cross-link positions,

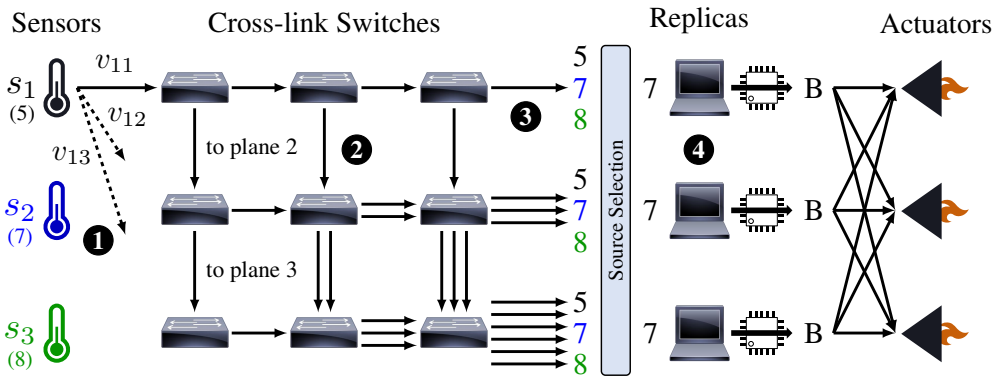

Figure 3.4: **Overview of CROSSTALK** — Example execution of CROSSTALK ($f = 1, g = 2$). We only show the flow of traffic sent by one sensor to one plane (v_{11}). However, the sensor also sends traffic to the other planes (v_{12} and v_{13}). Similar traffic flows exist for each sensor. For clarity we do not depict faults. Note that there may be any number of switches upstream or downstream of the cross-link switches, which are not shown.

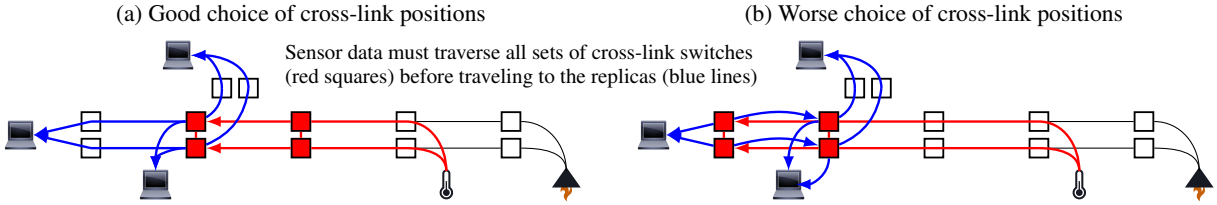

Figure 3.5: **Choosing Cross-link Positions** — Example of strategically choosing cross-link positions (red switches) to minimize latency in a system with 3 replicas and 2 planes. Only showing one sensor for clarity. Note that in (a), traffic does not deviate from the shortest path on its way from the sensor to the replicas.

each message is routed to the replicas. The protocol ensures that any message that arrives at one replica (potentially from a faulty plane), must arrive at all replicas. ❹ The rest of the protocol continues as in a traditional BFT SMR system (Figure 3.1a), with replicas using source selection to pick a sensor value, executing on that sensor value, and sending results to the actuators.

CROSSTALK reduces latency in two ways. First, the agreement process happens at *network speed*, with no need for any communication between replicas. That is, the time to reach agreement is bounded only by how fast the switches can forward messages to the replicas. Second, the cross-link positions can be chosen strategically. Latency is minimized when messages are not forced to go "out of their way" in order to traverse the cross-link switches before proceeding to the replicas. An example is shown in Figure 3.5.

CROSSTALK's approach also results in significantly lower computation overhead than the state of the art. Most importantly, since the replicas are not involved in agreement, the agreement process

is *agnostic to the number of replicas*. Thus, unlike other systems with perfect correctness (i.e., not relying on cryptography [28]), CROSSTALK requires only $2f + 1$ replicas (enough to mask output errors) instead of $3f + 1$ [46]. Moreover, unlike IGOR, CROSSTALK does not require replicas to perform any redundant computations on different cores.

Below we describe CROSSTALK in more detail. We focus on the rules for routing sensor data to the replicas and the agreement process, since these aspects are unique to CROSSTALK.

3.4.1 Routing Sensor Data to Replicas

This section describes the virtual channels CROSSTALK uses to route sensor data to the replicas. Data from each sensor is routed using n different virtual channels — one corresponding to each plane. Let s_i be a sensor that sends data to the replicas. Let m_{ij} be a message s_i sends on plane j. Let v_{ij} be the virtual channel used to carry m_{ij}.

The routing rules for v_{ij} are shown in Protocol 3.1. Let P be a list of all the plane identifiers ($\{1, \ldots, n\}$), with ID j appearing first. The remaining order does not impact correctness, but as discussed next, can be adjusted to balance the traffic load across planes. Also, assume that C is sorted such that latency is minimized as m_{ij} travels through the switches in C to the replicas (as shown in Figure 3.5). Let $P[1]$ and $C[1]$ refer to the first items in P and C respectively.

Protocol 3.1: Routing Rules for Sensor Data

- m_{ij} is routed from s_i along plane j to switch b_{jc}, where $c \leftarrow C[1]$
- **for each** $x \leftarrow 1, \ldots, n$ **do:** *# for all planes*
  - Let $p \leftarrow P[x]$
  - **for each** $y \leftarrow 1, \ldots, g + 1$ **do:** *# for all cross-link positions*
    - Let $c \leftarrow C[y]$
    - **if** $x \neq n$ **then:** *# route to next plane*
      - b_{pc} routes m_{ij} to b_{qc}, where $q \leftarrow P[x + 1]$
    - **if** $y \neq g + 1$ **then:** *# route to next cross-link position*
      - b_{pc} routes m_{ij} to b_{pd}, where $d \leftarrow C[y + 1]$
- **for each** $x \leftarrow 1, \ldots, n$ **do:** *# for all planes*
  - Let $p \leftarrow P[x]$ and $c \leftarrow C[g + 1]$ *# last cross-link position in plane*
  - m_{ij} is routed from b_{pc} along plane p to all replicas

Figure 3.4 shows the results of routing some m_{ij} (m_{11} in the example) through v_{ij} (v_{11}) in a simple network. As shown, more network traffic is generated on planes at higher indexes in P. Thus, to balance the traffic load across the planes, P should be chosen so the same plane does not

appear at the same index in P for multiple virtual channels originating at the same sensor. For example, P might contain $\{1, 2, 3\}$ for v_{i1}, but $\{2, 3, 1\}$ and $\{3, 1, 2\}$ for v_{i2} and v_{i3} respectively.

As a result of routing sensor data according to Protocol 3.1, we have the following guarantees.

LEMMA 3.1. If a non-faulty replica r receives a message m on v_{ij}, then all non-faulty replicas receive m on v_{ij} (agreement). Also, if s_i and some plane j are non-faulty, and s_i sends m, then all non-faulty replicas receive m on v_{ij} (validity).

Proof. First, consider agreement. Since switches are limited to omission, a faulty switch cannot create or alter messages. Also, switches drop messages that arrive on ports where no routing rules exist. Thus, m must have traversed a legal route in v_{ij}. Since $|C| \geq g + 1$, there must be at least one $c \in C$ for which b_{pc} is non-faulty for all $p \in \{1, \ldots, n\}$ (i.e., all cross-link switches at position c are non-faulty). According to Protocol 3.1, m's route to r must have entered such a b_{pc} from some plane $p = P[x]$. At that point, m is forwarded through all planes $P[x], ..., P[n]$. Also, to get to plane $P[x]$, m must have traversed planes $P[1], ..., P[x]$. Thus m traverses all planes $P[1], ..., P[n]$. Since there are $g + 1$ planes, at least one plane k is non-faulty and forwards m through any remaining cross-over switches on plane k, and then to all replicas. Thus, all non-faulty replicas get m on v_{ij}.

Next, consider validity. Since s_i is non-faulty, it sends m to all planes (i.e., on $v_{i1}, ..., v_{in}$). Since plane j is non-faulty, m travels through each b_{jc}, where $c \in C$, and then to all replicas. Thus, all non-faulty replicas receive m on v_{ij}. □

We note that other routing rules could be used with CROSSTALK besides those in Protocol 3.1. For example, a system could tolerate more scenarios with *more* than g faulty switches if each cross-link switch forwarded messages to *all* planes and not just the *next* plane. However, such approaches have much higher communication costs, and still cannot tolerate all $g+1$ faulty switch scenarios (which is impossible with only $g + 1$ planes, since all planes could be blocked).

3.4.2 Agreement and BFT SMR Protocols

With the virtual channels for routing sensor data to the replicas defined, we can now describe a simple protocol that ensures replicas agree on data from a potentially faulty sensor s_i. The protocol starts at a synchronized time t', at which s_i transmits and the replicas start gathering messages. It ends at a predetermined time $t = t' + \Delta$, which is chosen to accommodate the known worst-case latency from s_i to all replicas. In general, the protocol works by having the replicas select among the messages received on each virtual channel using a simple priority-based scheme.

The protocol is shown in Protocol 3.2. Let M_i be an n-dimension vector used by the replicas to store messages from sensor s_i (i.e., store what s_i sent to each plane). All elements are initialized

to $\perp$ to indicate the messages are initially missing. Let m_i be the final message the replicas accept from s_i. It is initialized to a default value.

Protocol 3.2: Agreement Protocol

Sensor s_i (at time t'):

- Send message m on virtual channels $v_{i1}, ..., v_{in}$

Each replica (at time $t = t' + \Delta$):

- **for each** $j \leftarrow 1, \ldots, n$ **do:** *# for all planes*
  - Let $W_{ij} \leftarrow$ the messages received on v_{ij}
  - **if** W_{ij} contains only one unique message m **then:**
    - Set $M_i[j] \leftarrow m$ *# set value received from s_i on v_{ij}*
- **for each** $j \leftarrow 1, \ldots, n$ **do:** *# for all planes*
  - **if** $M_i[j] \neq \perp$ **then:** Set $m_i \leftarrow M_i[j]$ *# priority-based selection*

The agreement protocol provides the following guarantees.

THEOREM 3.1. All non-faulty replicas agree on m_i (agreement). If s_i is non-faulty and sends m, then m_i is m (validity).

Proof. First, consider agreement. All non-faulty replicas decide on m_i by applying the same priority-based selection to M_i. Thus, we simply need to prove all non-faulty replicas agree on M_i. We make the proof by contradiction. Say that at the end of the protocol, two non-faulty replicas r_1 and r_2 have different values $M_i[j]$ for some j. This means r_1 and r_2 have different sets W_{ij}. Thus, one replica (say r_1) received a message m on v_{ij} in time interval $[t', t' + \Delta]$ that the other (r_2) did not. Lemma 3.1 implies r_2 also receives m. Moreover, since Δ is chosen to accommodate the worst-case traversal time from the sensor to the replicas, sensor s_i is not subject to timing faults (we relax this assumption in §3.4.4), and faulty switches are omissive and thus cannot delay messages, m must arrive at r_2 in $[t', t' + \Delta]$. This is a contradiction. Thus, all non-faulty replicas agree on m_i.

Next, consider validity. Since at least one plane j is non-faulty, Lemma 3.1 implies all non-faulty replicas receive m on v_{ij}. Since switches are limited to omission, they cannot create or alter messages. Similarly, all switches drop messages that arrive on ports where no routing rules exist. Since s_i is non-faulty, it sent only m. Thus, non-faulty replicas can receive no message besides m on $v_{i1}, \ldots, v_{in}$. Thus, for all non-faulty replicas, $M_i[j] = m$ and M_i contains only m or $\perp$. Thus, all non-faulty replicas set $m_i \leftarrow m$. □

The rest of CROSSTALK proceeds identically to a traditional BFT SMR system (see Figure 3.1a). Specifically, as a result of running Protocol 3.2 for each sensor, all non-faulty replicas

possess an identical vector of sensor values (up to one per sensor). Next, the replicas use a source selection process to determine which of these values to use as the input to their execution. Source selection is application specific, but is often simply a mid-value selection [26]. It has been shown that, with $2f + 1$ sensors, the non-faulty replicas are guaranteed to select a value from a non-faulty sensor or a value bounded by values from non-faulty sensors [28].

Next, the replicas execute on the selected sensor value. If the same deterministic execution is performed on all replicas, all non-faulty replicas are guaranteed to produce the same output. Lastly, the replicas send their outputs to the actuators, with each output traveling from each replica to each actuator over each redundant plane. The actuators accept the first valid message that arrives from each replica. The actuators perform a majority vote of the accepted messages to decide which operation to perform. Since there are $2f + 1$ replicas, the voted output is guaranteed to be correct.

3.4.3 The Trouble with Timing Faults

In the previous section, we made the simplifying assumption that the sensors were not subject to timing faults — i.e., if a faulty sensor generates a message, it is guaranteed to do so within an a priori known bounded time. Having a bound allows designers to calculate a time t at which, if a replica is going to receive a message from a sensor, that message is guaranteed to have arrived. When combined with the design of the virtual channels, it ensures that all non-faulty replicas agree on the data from all redundant sensors at time t (see Protocol 3.2).

Traditional agreement protocols tolerate timing faults as a consequence of requiring multiple communication rounds between the replicas. For example, a faulty sensor could delay sending its message m until right before the deadline for the first round, t_1, resulting in some replicas getting m before t_1, and others not. However, in the next round, the replicas share the messages they received with each other. As a result, any replica that failed to receive m by t_1 will receive it by the deadline for the second round, t_2.

Unfortunately, tolerating timing faults in a protocol with a single communication round (from sensors to replicas) is not so easy. Since the replicas cannot communicate with each other, they have no way to differentiate between a message that arrived before the deadline for *some* replicas, and one that arrived before the deadline for *all* replicas.

The challenge of ensuring agreement in the presence of timing faults is not specific to CROSSTALK. In fact, it can be shown that timing faults make it impossible for *any* protocol with only one communication round to ensure agreement by some synchronized deadline t. The result holds even if the network is reliable and has the guarantees of a broadcast channel [108] (i.e., it is impossible for the sensor to send a message to some replicas and not to others). This detail is important, because it means some recent guidance on constructing BFT protocols using broadcast

channels [58] is actually *incorrect* if timing faults can occur.

PROPOSITION 3.1. In a synchronous system subject to timing faults, it is impossible for a protocol that uses only a single communication round to guarantee both the agreement and validity properties of Theorem 3.1 by an a priori known time t.

Proof. We construct the proof by contradiction. Assume that such a protocol exists and that all replicas are non-faulty. We will show that there exists a scenario in which agreement is violated. Consider three possible scenarios below.

Scenario 1: Sensor s_i is non-faulty and sends m on time. m arrives at all the replicas before time t. By the validity property of the protocol, all replicas must set $m_i \leftarrow m$.

Scenario 2: Sensor s_i is faulty and crashes. Thus, no message arrives at any replica before t. By the agreement property of the protocol, all replicas must set m_i to the same message (e.g., a predetermined default).

Scenario 3: Sensor s_i is faulty and sends m after some arbitrary delay. Now, say we split the replicas into two sets — Set 1 and Set 2. The network latency in a synchronous system is bounded, but *not exactly known* (the exact latency depends on network loading) [46]. Thus, even if Set 1 and Set 2 are perfectly synchronized, m may arrive at Set 1 and Set 2 at slightly different times — e.g., before t for Set 1 and after t for Set 2. Even if t is increased, a faulty sensor can always delay longer to cause the same scenario. Alternatively, it is well known that perfect synchronization in a distributed system is not achievable [181] — meaning local time t on Set 1 may occur slightly after local time t on Set 2 in wall-clock time. Thus, even if the latencies from s_i to all replicas were *exactly known and the same*, m may arrive before t for Set 1 and after t for Set 2.

From the perspective of the replicas in Set 1, this scenario is identical to Scenario 1 — thus, all replicas in Set 1 must decide on m. From the perspective of the replicas in Set 2, this scenario is identical to Scenario 2 — thus, all replicas in Set 2 must decide on a default value. m and the default value may not be the same. Hence, the agreement property of the protocol is violated. □

3.4.4 Overcoming the Impossibility Result

In order to circumvent the impossibility result in Proposition 3.1, we need a way to bound when sensors can send messages to some safe temporal window, such that no replica can receive a message too early or too late. A Byzantine faulty sensor may send messages at arbitrary times regardless of any rules we impose on it. Thus, enforcement of this window must be outside the control of the potentially faulty sensor.

One solution is to use a time-triggered network, like TTEthernet [32] or IEEE 802.1Qbv [34], in which the switches are tightly synchronized to the nodes, and the exact timing of all message transmissions is scheduled offline. This design allows the switches to enforce a window during

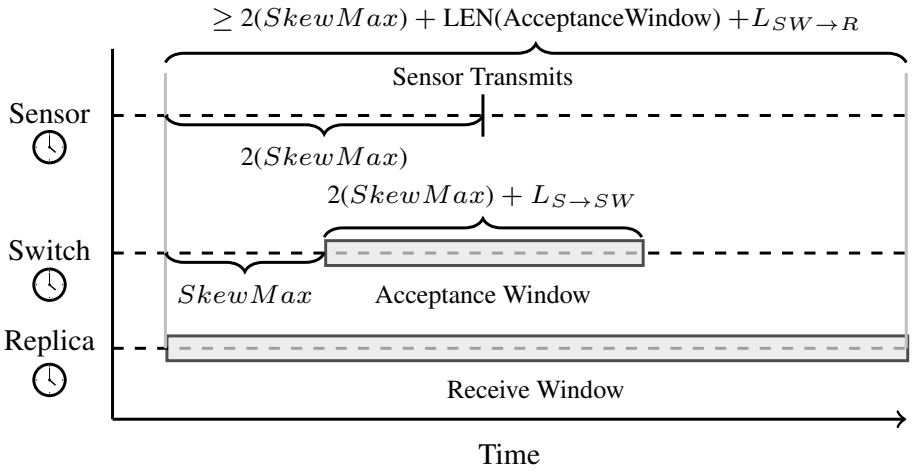

Figure 3.6: **Suitable Time Bounds** — Suitable time bounds for timestamping in CROSSTALK (shown with no skew between devices).

which a message from a node is allowed to arrive. Unfortunately, there are downsides to such networks, including the need for expensive specialized switches and network cards, as well as proprietary network scheduling tools [182].

A simpler approach is to use network timestamps. Let b_{jk} be the switch on plane j connected to sensor s_i. Then, in Protocol 3.1, have b_{jk} place the synchronized time that each message m arrives from s_i in m before forwarding m on v_{ij}. Since switches are constrained to omission (§3.3.2), meaning b_{jk} cannot put different timestamps in different outgoing copies of m, this scenario is indistinguishable from s_i sending messages to b_{jk} that *already* contain timestamps and b_{jk} not altering any messages. This means the agreement and validity conditions of Lemma 3.1 still hold. Moreover, since b_{jk} is limited to omission, the timestamp in m accurately reflects when m arrived at b_{jk}.

With this timestamping function in place, CROSSTALK can be altered to tolerate timing faults by slightly altering Protocol 3.2. Specifically, rather than letting $W_{ij} \leftarrow$ all messages received on v_{ij}, let W_{ij} only contain messages with timestamps that fall within a specific *acceptance window*.[1]

Figure 3.6 defines suitable bounds for the acceptance window (which are similar to [183]). In addition, it defines the window during which messages timestamped within the acceptance window may arrive at the replicas (i.e., the *receive window*). The duration of the agreement protocol (i.e., adjusted Protocol 3.2) is equal to the width of the receive window, which is $4(SkewMax) + L_{S \to SW} + L_{SW \to R}$, where $SkewMax$ is the maximum skew between any two synchronized devices, and $L_{S \to SW}$ and $L_{SW \to R}$ are the worst-case latencies from the sensor to switch b_{jk} and from b_{jk}

[1] We acknowledge that this technique requires replicas to have a priori knowledge of when sensors are scheduled to send messages. However, this knowledge is already available in synchronous flight software frameworks that we are familiar with (e.g., NASA's cFS [87]).

to the replicas respectively. In modern networks $SkewMax$ values as small as a microsecond are possible [184, 182], while worst-case message latencies (i.e., $L_{S\to SW} + L_{SW\to R}$) are typically several milliseconds [95, 57], Thus, the extra latency needed to make CROSSTALK tolerate timing faults is largely insignificant.

With these windows defined, we can prove the correctness of the revised CROSSTALK protocol.

LEMMA 3.2. If a non-faulty replica r receives a message m on v_{ij}, then if m was timestamped within the acceptance window, m must have arrived within r's receive window.

Proof. Assume the receive window shown in Figure 3.6 starts at time 0 for each device. Since the switches are limited to omission, and drop messages that arrive on ports where no routing rules exist, the timestamp in m must be the time m entered the network on plane j at some switch b_{jk} (the switch directly connected to the sensor on plane j). Since switches are not subject to timing faults, the latest time at which r receives m (on r's clock) is the time b_{jk} timestamped m according to r's clock (call this time t) $+L_{SW\to R}$. Say b_{jk} is skewed $SkewMax$ after r. In this case, t can be at most $4(SkewMax) + L_{S\to SW}$ at r — i.e., the end of the acceptance window in Figure 3.6, skewed right by $SkewMax$. Thus, m can arrive at r as late as $4(SkewMax) + L_{S\to SW} + L_{SW\to R}$ according to r's clock, which is at the end of r's receive window. Similarly, the earliest m can arrive at r is at t (if the network latency was zero). Say b_{jk} is skewed $SkewMax$ before r, in which case t can be as little as time 0 at r — i.e., the start of the acceptance window in Figure 3.6, skewed left by $SkewMax$. Thus, m can arrive at r as early as time 0 (per r's clock), which is in r's receive window. □

LEMMA 3.3. If s_i and some plane j are non-faulty, and s_i sends m, then all non-faulty replicas receive m on v_{ij} within their respective receive windows. Moreover, m is timestamped within the acceptance window.

Proof. Lemma 3.1 implies all non-faulty replicas receive m on v_{ij}. Since s_i is non-faulty, it transmitted at the scheduled time ("Sensor Transmits" in Figure 3.6). Thus, m arrived at b_{jk} as early as time $SkewMax$ on b_{jk}'s clock, and as late as $3(SkewMax) + L_{S\to SW}$ on b_{jk}'s clock. Thus m must contain a timestamp within the acceptance window. Thus, Lemma 3.2 implies m arrives at all non-faulty replicas within their respective receive windows. □

THEOREM 3.2. In the presence of timing faults, all non-faulty replicas agree on m_i (agreement). Also, if s_i is non-faulty and sends m, then m_i is m (validity).

Proof. First, consider agreement. Like in the original Theorem 3.1, we simply need to prove all non-faulty replicas agree on M_i. We will prove by contradiction. Suppose that at the end of the protocol, two non-faulty replicas r_1 and r_2 have different values $M_i[j]$ for some j. This means r_1

and r_2 have different sets W_{ij}. Thus, one replica (say r_1) accepted a message m on v_{ij} that r_2 did not. This means m contained a timestamp within the acceptance window and arrived within r_1's receive window. Lemma 3.1 implies r_2 receives the same m on v_{ij} as r_1 (with same timestamp). Lemma 3.2 implies m arrives within r_2's receive window. Since r_1 is non-faulty and accepted m, r_2 also accepts m, which is a contradiction.

Next, consider validity. Since at least one plane j is non-faulty, Lemma 3.3 implies all non-faulty replicas receive the same m on v_{ij} within their respective receive windows, and that m's timestamp is within the acceptance window. Thus, all non-faulty replicas accept m. Since switches are limited to omission, they cannot create or alter messages. Similarly, all switches drop messages that arrive on ports where no routing rules exist. Since s_i is non-faulty, it sent only m. Thus, non-faulty replicas receive no message besides m on v_{i1}, ..., v_{in}. Thus, for all non-faulty replicas, $M_i[j] = m$ and M_i contains only m or $\perp$. Thus, all non-faulty replicas set $m_i \leftarrow m$. □

3.5 Prototype Implementation

To evaluate our approach, we implemented a prototype of CROSSTALK in ∼8800 lines of C and Python code (including supporting tooling and scripts). The replicas were realized on a cluster of 5 embedded AsRock J3160 computers with 1.6 GHz quad-core processors. We selected this platform because its performance is comparable to state-of-the-art single board computers used in avionic systems [48]. The replicas communicated with two HP Z2 workstations with 3.2 GHz Intel Core i7-8700 processors — one representing a set of 3 redundant sensors, and the other a set of 3 redundant actuators. The computers all ran CentOS 7.9 with kernel 3.10.0-1160.53.1 and the PREEMPT_RT patch.

The replicas, sensors, and actuators communicated using real AFDX network cards manufactured by TTTech. We used NASA's TTX library [95] for interfacing to the cards. The cards each connected to 2–3 network planes (depending on the experiment), each consisting of a variable number of AFDX switches. Due to limited hardware availability (some experiments required 10+ switches), we also used a Dell Precision 7920 server with dual Intel Xeon 6242R processors to emulate switches in some experiments. The server was directly wired to the real AFDX switches, and was configured to execute the AFDX protocol and to mimic the delays of the real switches.

The replicas executed NASA's Core Flight System (cFS) [87], an open flight software framework used in real spacecraft. cFS tasks run in fixed time slots according to a cyclic executive. We synchronized the replicas using periodic interrupts from an external timing circuit. We used a 500 Hz interrupt rate, which matches real systems and past work [28].

We implemented two state-of-the-art protocols for comparison. The first, OM, is a traditional BFT SMR protocol based on Lamport's signed messages [46], which solves agreement in the theo-

retical minimum number of rounds. Like CROSSTALK, OM requires only $2f+1$ replicas to tolerate f faults. We used a simple signature scheme in OM based on modular inverse [46], which takes only around 0.01 ms to sign and verify a 200-byte message on our replicas. The second, IGOR [28], is a recent BFT SMR protocol that, to our knowledge, achieves lower worst-case latency than any existing protocol. IGOR has an optional "Filtering Stage", which is designed to reduce latency when message fragmentation is slow (e.g., there is 1+ ms between sending fragments). However, NASA's TTX library and our AFDX cards allow fragments to be sent back-to-back with no delay. Thus, we report IGOR's latency without the optional Filtering Stage, which minimizes IGOR's latency on our testbed (it is 10–20% lower than with Filtering). We used SM as the agreement primitive in IGOR. Unlike CROSSTALK, IGOR requires $3f + 1$ replicas to tolerate f faults.

Lastly, we compared CROSSTALK to NOREP, an SMR system with $2f + 1$ replicas, but no agreement protocol. Thus, NOREP does not tolerate Byzantine faults, and represents the theoretical minimum latency that can be achieved.

3.6 Evaluation

Our evaluation was designed to answer five key questions: (1) How does CROSSTALK's latency compare to the state of the art? (2) How much does CROSSTALK's low computation overhead improve schedulability? (3) What is the cost of CROSSTALK's cross-links? (4) What is CROSSTALK's communication overhead? and (5) How does CROSSTALK perform in a real spaceflight mission?

3.6.1 Latency

Experimental setup. For this experiment, we considered three representative network topologies shown in Figure 3.7. The first is an aircraft network from a recent ONERA paper [96]. The second and third are a space capsule network and space station network from a recent NASA presentation [9].

For each topology, we considered both $f = 1$ and $f = 2$ faulty replicas and $g = 1$ and $g = 2$ faulty switches. These values were chosen to reflect the requirements of real systems in practice [10, 133, 173]. For each combination of f and g, we executed three trials. In each trial, we randomly selected the positions of $2f + 1$ replicas, one redundant set of sensors, and one redundant set of actuators. As is typical in practice [171, 172], replicas were required to connect to different switches to reduce the chance of common-cause failures.

We developed tools based on Python's NetworkX [185] library to determine the rules for routing virtual channels between the nodes, as well as to select the cross-link positions for CROSSTALK.

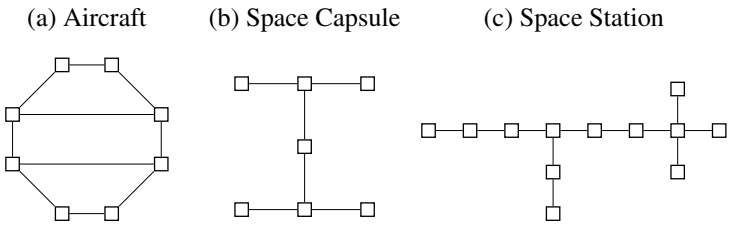

Figure 3.7: **Network Topologies** — Network topologies used in our evaluation. Each figure shows the switches in a single network plane.

The tools start by considering a single-plane version of the network, and use search to identify the shortest routes between all nodes that need to communicate, while meeting any protocol-specific constraints. For example, CROSSTALK requires that all routes from a given sensor to the replicas traverse a common set of $g + 1$ switches in the same order (as shown in Figure 3.4). These common switches were then used as the cross-link positions for that sensor, and the final multi-plane routes were determined by replicating the original single-plane routes on all planes, then extending them across planes at the cross-link positions. For NOREP, OM, and IGOR, the final routes were determined by simply using the original single-plane routes on each plane.

In each trial, the worst-case execution time (WCET) of the execution on the replicas was 10 ms, and the size of the sensor and actuator data was 200 bytes. Both were chosen to reflect typical values in commercial aircraft [107, 57, 96].

We instrumented the network so that each switch delayed each message approximately 1 ms when forwarding the message to a device on the same plane, which reflects typical delays in avionics networks due to network congestion [57, 96]. We configured the delay over cross-links to be 0.5 ms to reflect the fact that these links are significantly less congested (many systems do not currently use cross-links) [10].

We constructed the cFS task schedule by executing each segment of each protocol (e.g., a round of agreement) for 600 iterations and estimating each segment's WCET by adding 10% margin to the longest measurement. We then arranged the protocol segments in sequence, with each segment allocated enough time slots to accommodate the WCET. Lastly, we executed the resulting cFS schedule on our testbed, verified all deadlines were met, and recorded the end-to-end latencies of each protocol from sensors to actuators.

We note that the latencies of all the protocols are the same in both the presence and absence of faults. Thus, we did not explicitly inject faults when taking measurements. Moreover, the latencies for OM, IGOR, and NOREP are the same regardless of the number of planes, so we do not specify g in the results for those protocols.

Results. Our results are shown in Figure 3.8. In all cases, CROSSTALK's latency was compa-

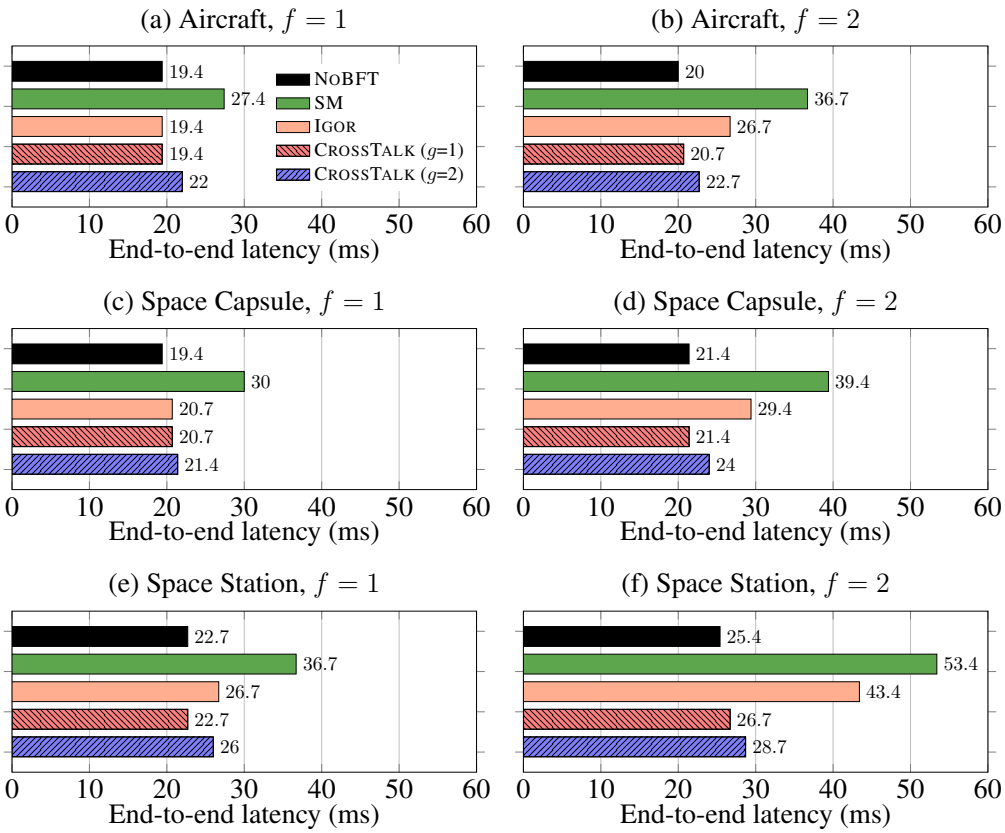

Figure 3.8: **CrossTalk's Latency** — Average worst-case latency of CrossTalk compared to state-of-the-art BFT protocols and NoRep.

rable to or better than the best existing protocol (Igor), meaning it can meet tighter deadlines. Importantly, CrossTalk achieves this latency *without* requiring computations on multiple cores (which, as we will see next, results in much higher schedulability).

Moreover, CrossTalk's latency savings increase as the number of faulty replicas, as well as the size of the network, increases. For example, in the space capsule, CrossTalk's latency is 1.2–1.4× lower than the best existing protocol when tolerating $f = 2$ faults. In the space station network, CrossTalk's savings increase to 1.5–1.6×. The reason is that, as f increases, the time needed for other BFT protocols to reach agreement increases due to the need for more communication rounds. Similarly, as the size of the network increases, the duration of each round increases.

We acknowledge that as the number of faulty switches (g) increases, the need for more cross-plane communication causes CrossTalk's improvement over Igor to decrease. However, unlike CrossTalk, Igor does not consider switch faults. Moreover, CrossTalk's latency is still 8.7% lower than Igor's on average when $g = 2$. Finally, we note that we are not aware of any system

required to tolerate more than 2 switch faults in practice.

3.6.2 Schedulability

Unlike IGOR, CROSSTALK can be realized on a single processor core. In this experiment, we evaluated how much this design improves schedulability.

Experimental setup. We considered workloads consisting of independent constrained-deadline BFT tasks executed on either a single-core, or distributed over three cores. In each case, we varied the application utilization per core from 0.1 to 1, and randomly generated 1000 tasksets per utilization. The task WCETs and periods were randomly chosen from $\{5, 10, 15, 20\}$ ms and $\{25, 50, 100, 200\}$ ms respectively, which are common values in practice and match prior work [28].

For each task, we randomly selected a topology and set of replicas, sensors, and actuators from §3.6.1. We used the data from §3.6.1 to specify the WCTT of messages and WCET of each protocol segment (e.g., a round of agreement). Each task was randomly assigned an end-to-end deadline within which data was required to leave the sensors, be processed on the replicas, and arrive at the actuators. The deadlines were uniformly distributed between the worst-case latency for NOREP (the best latency that can be achieved) and the task period.

For each utilization, we report the fraction of schedulable tasksets with each protocol. We scheduled tasks on the processor cores using the same heuristic as in IGOR [28], where tasks with smaller periods were scheduled first.

As in past work, we did not use IGOR for *every* BFT task when evaluating IGOR. Otherwise, IGOR would have very low schedulability due to requiring execution on all three cores. Instead, we used the strategy from the IGOR paper [28] in which we first attempted to use OM for all tasks. If any tasks were not schedulable with OM, we attempted to schedule only those tasks with IGOR. Thus, the system only incurs the overhead of IGOR when needed to meet deadlines.

Results. Our results are shown in Figure 3.9. As shown, we were able to schedule significantly more tasksets with CROSSTALK than with the best existing protocols. For example, in the single-core case, 2.37–3.12$\times$ more tasksets were schedulable with CROSSTALK than with OM. In the multi-core case, 2.13–4.24$\times$ more tasksets were schedulable with CROSSTALK than with SM+IGOR. Importantly, CROSSTALK also achieves very high schedulability at high utilizations. For example, with 3 cores in the $f = 1, g = 1$ case, we could schedule 88% of tasksets with CROSSTALK at a core utilization of 1.0, while we could only schedule 1% of tasksets with SM+IGOR.

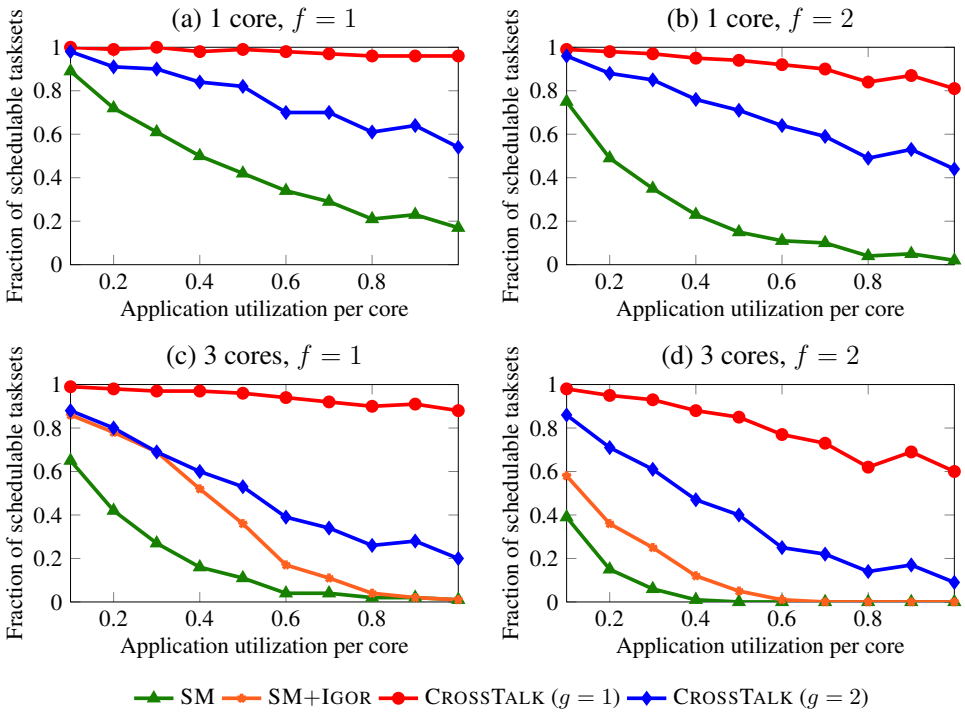

Figure 3.9: **CROSSTALK's Schedulability** — Schedulability improvement when using CROSSTALK. Note that IGOR cannot be used on single-core processors.

3.6.3 Cost Trade-off of Cross-Links

In the previous sections, we used our network scheduling tools to select $g + 1$ cross-link positions for each sensor, such that the length of the routes from the sensor to the replicas was minimized. Thus, since there may be *many* sets of sensors in a system, all for different purposes and at different positions, the system may contain *more* than $g + 1$ total cross-link positions. In this section, we evaluate the cost of this approach (e.g., in mass), as well as determine how much reducing the number of cross-links positions — which lowers costs but also forces some sensors to use "worse" cross-link positions — still allows us to retain CROSSTALK's latency benefits.

Experimental setup. We randomly generated 15 system configurations, each containing a topology from §3.6.1 and the positions of $2f + 1$ replicas, one set of redundant actuators, and 8 sets of redundant sensors (representing the many sensors in a real system). We varied the total number of cross-link positions allowed in the network from 2 to 7 (the number of switches in the smallest topology in Figure 3.7). For each BFT SMR protocol, system configuration, number of allowed cross-link positions, and combination of $f = \{1, 2\}$ and $g = \{1, 2\}$, we used our network tools (as described in §3.6.1) to find the exact cross-link positions and message routes that minimized the length of the longest route from the sensors to the actuators. Then, in each case, we calculated the

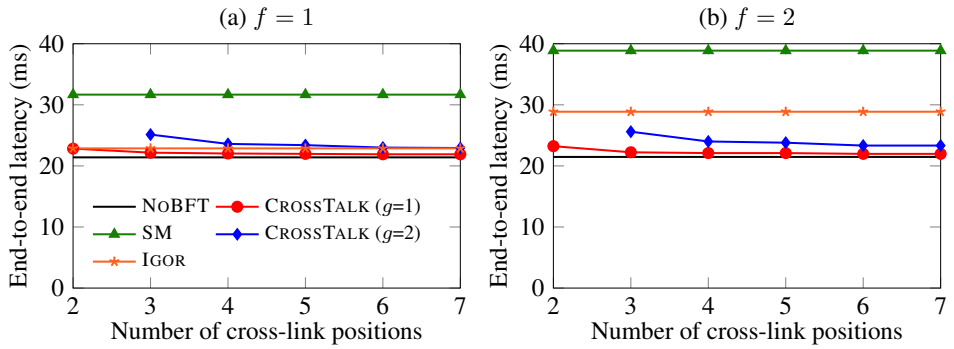

Figure 3.10: **Impact of Cross-links on Latency** — Impact of the total number of cross-link positions (the number of switches with cross-links per plane) on CrossTalk's system-wide worst-case latency. We only show up to 7 cross-link positions, since more cross-link positions result in no additional latency savings for CrossTalk. Note that when $g = 2$, CrossTalk requires $\geq g + 1 = 3$ cross-link positions.

protocol's worst-case latency using the switch delays and WCETs from 3.6.1. We report, for each number of cross-link positions, the average worst-case latency for each protocol across all sensors and configurations.

In addition, we calculated the mass of the replicas and cross-links needed for CrossTalk compared to OM and Igor for each combination of f, g, and the total number of cross-link positions in the system. We assumed each replica is 4.5 kg (based on a typical single board computer chassis [186]), each cross-link is 0.1 kg/m (a standard approximation in avionic systems [187]), and that each cross-link is 2 m long — which is more than the required network plane separation in safety-critical applications [188]. We then converted our mass results to monetary cost in a typical spaceflight application, assuming each replica costs $200K (the cost of a typical spaceflight computer [13]), each cross-link costs $150/m (the cost of space-qualified cables [189]), and that it costs $15K to launch 1 kg of mass to low Earth orbit (which is typical [190]).

Results. Figure 3.10 shows how the total number of cross-link positions in the system impacts CrossTalk's latency (and thus ability to meet deadlines). As shown, CrossTalk's latency is minimized with around 6 cross-link positions, with more cross-link positions yielding no additional benefit. This is because, in general, 6 cross-link positions is enough to allow each sensor's traffic to traverse the required $g + 1$ cross-link positions while still taking the shortest path to the replicas (as shown in Figure 3.5). Conversely, restricting the number of cross-link positions increases CrossTalk's latency — but only gradually. For example, with only 3 cross-link positions, CrossTalk's latency is still lower than Igor's in most cases. Moreover, we emphasize that when $f = 2$, CrossTalk's latency is always strictly smaller than Igor's for all numbers of cross-link positions.

Table 3.1: **Mass of CROSSTALK** — Mass of the replicas and cross-links for each protocol when configured to tolerate different numbers of faults. Results are in kilograms.

Faults		**SM**	**IGOR**	**CROSSTALK (# of cross-link positions)**					
f	g			2	3	4	5	6	7
1	1	13.5	18	13.9	14.1	14.3	14.5	14.7	14.9
1	2	13.5	18	N/A	15.3	15.9	16.5	17.1	17.7
2	1	22.5	31.5	22.9	23.1	23.3	23.5	23.7	23.9
2	2	22.5	31.5	N/A	24.3	24.9	25.5	26.1	26.7

Table 3.2: **Cost of CROSSTALK** — Cost of the replicas and cross-links for each protocol in a typical spaceflight application. Results are in hundreds of thousands of US dollars.

Faults		**SM**	**IGOR**	**CROSSTALK (# of cross-link positions)**					
f	g			2	3	4	5	6	7
1	1	8	10.7	8.1	8.1	8.2	8.2	8.2	8.3
1	2	8	10.7	N/A	8.3	8.4	8.5	8.6	8.7
2	1	13.4	18.7	13.4	13.5	13.5	13.5	13.6	13.6
2	2	13.4	18.7	N/A	13.7	13.8	13.9	14	14.1

Tables 3.1 and 3.2 show how CROSSTALK's mass and cost compare to IGOR when restricted to different numbers of cross-link positions. As shown, even with 7 cross-link positions (which is more than needed to minimize CROSSTALK's latency), CROSSTALK is *always* lighter and cheaper than IGOR. For example, when $f = 2$ and $g = 1$, CROSSTALK is 7.6 kg lighter and >\$500K cheaper. The reason is that CROSSTALK requires f fewer replicas than IGOR to tolerate f faults, and these replicas are much heavier and more costly than CROSSTALK's cross-links. Note than since IGOR is based on overlapping quorums of replicas, there is no way to reduce the number of replicas it requires.

We note that, while we assumed single and multi-core replicas have the same monetary cost, multi-core replicas may be more expensive in practice — in which case IGOR's costs would be even higher (i.e., CROSSTALK would have larger cost savings). Moreover, while we did not measure energy consumption, it is expected that CROSSTALK's need for fewer replicas, as well as only single-core replicas, would result in significant power savings over IGOR.

Lastly, we note that while CROSSTALK's mass and cost were slightly higher than OM, OM has significantly higher latency (up to 2× higher, see Figures 3.8, 3.10) and worse schedulability (Figure 3.9).

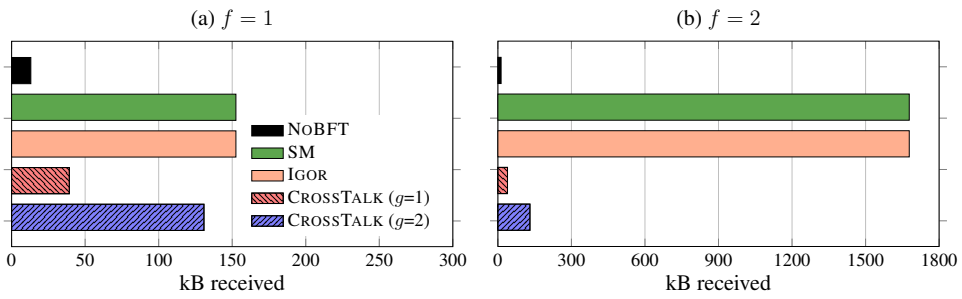

Figure 3.11: **CROSSTALK's Communication Overhead** — Number of kB received by each replica per plane per sensor in CROSSTALK compared to other protocols.

3.6.4 Communication Overhead

Experimental setup. One common concern regarding BFT SMR protocols is that they produce large amounts of network traffic. Since the switches in CROSSTALK create copies of messages in the network, one cannot evaluate CROSSTALK's communication overhead by measuring the amount of traffic *transmitted* by a node. Instead, we placed a network tap between one replica and one network plane, and used the tap to capture all traffic received by the replica. We captured traffic for 60 executions of each protocol in a representative space station network (repeating the experiment from §3.6.1). We report the amount of traffic received by the replica per sensor.

We note that, as described in §3.4.1, the virtual channels in CROSSTALK are balanced so that replicas receive the same amount of traffic from each plane. Similarly, the traffic in NOREP, OM, and IGOR is identical across all planes. Thus, the choice of plane to tap does not matter. Moreover, since we log traffic as it enters the replica and not elsewhere in the network, the choice of network topology and positions of the nodes (e.g., replicas) does not impact the results.

Results. Our results are shown in Figure 3.11. In general, CROSSTALK's communication overhead is lower than other state-of-the-art protocols. Additionally, its relative efficiency increases with f. For example, when $f = 2$, CROSSTALK's communication overhead is 12.83× lower than IGOR and OM. The reason is that CROSSTALK does not require any rounds of communication between replicas, while other protocols need at least $f + 1$ inter-replica rounds (with increasing message sizes). We acknowledge that using IGOR's optional Filtering Stage would decrease its overhead when $f = 2$, but at the expense of increased latency (as described in §3.5) due to needing an additional round. Moreover, we note that while IGOR requires 7 replicas to tolerate $f = 2$ faults, there are only 5 replicas in our testbed. Running IGOR with 7 replicas would result in higher communication overheads for IGOR and more relative savings for CROSSTALK.

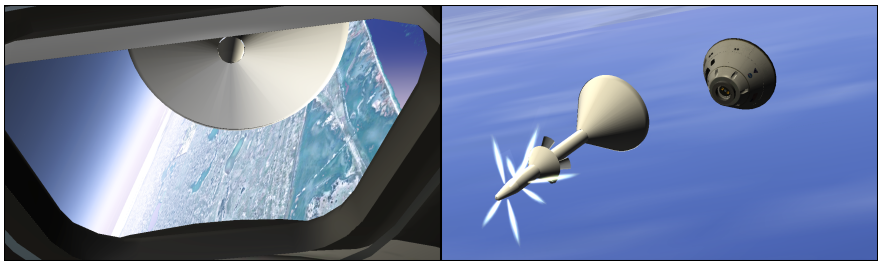

Figure 3.12: **CROSSTALK Spaceflight Simulation** — View from inside and outside the simulated capsule after being jettisoned from the rocket.

3.6.5 Case Study: Spaceflight Abort Test

Lastly, we wanted to determine how CROSSTALK performs in a concrete system. For this purpose, we used simulated flight software from a NASA abort test, which we obtained from the authors of IGOR [28]. The simulation, shown in Figure 3.12, is based on a real mission conducted in 2019. During the mission, a rocket carried an empty crew capsule to an altitude of 9.5 km, at which an abort was intentionally triggered and the capsule was safely ejected from the rocket. We executed the provided flight software on replicas in our cluster. The flight software communicated through our AFDX switches with a provided NASA simulation, which modeled the vehicle dynamics, sensors, and actuators. The network was instrumented to induce delays comparable to AFDX avionics networks in practice [57, 96], as in §3.6.1.

The main flight software control loop executed at 40 Hz — in each execution reading data from the sensors, performing computations, and sending results to the actuators. The end-to-end latency requirement from the sensors to the actuators was 25 ms. If this deadline was not met, the rocket deviated from its correct flight path and the abort failed.

We scheduled the flight software using the same approach as in §3.6.1 and executed both CROSSTALK and IGOR on our testbed. Both protocols met all deadlines when $f = 1$, resulting in correct behavior of the rocket and capsule. This includes CROSSTALK in the $g = 2$ (two faulty switches) case, which executed with approximately 4 ms of margin. However, while IGOR failed to meet deadlines when $f = 2$ by several milliseconds (resulting in the capsule ejecting at an incorrect orientation that compromised safety), CROSSTALK continued to meet deadlines when $f = 2$, even with $g = 2$ faulty switches. Thus, CROSSTALK's ability to reduce latency compared to IGOR directly resulted in improved fault resilience — while both protocols could be used, CROSSTALK was able to tolerate an additional faulty replica. Moreover, CROSSTALK did so while using nearly 3× less total CPU time.

3.7 Related Work

Making agreement fast. Several protocols use techniques to reduce the latency of reaching agreement. Some do this by allowing replicas to stop an agreement protocol early when no faults occur or all replicas initially agree [191, 192, 154, 47, 193, 194, 195, 196] — sometimes resulting in protocols that appear to solve agreement in a single round. However, these protocols provide no latency savings when faults occur. Other protocols use a speculative approach, in which replicas (1) execute on client requests optimistically while running an agreement protocol in the background [140, 141, 142, 143, 144], or (2) forgo agreement altogether and check for state divergence after the execution [81, 82, 83, 84, 85, 197]. In either case, if divergence is detected, the system is rolled back and the execution is repeated, resulting in high worst-case latencies. In contrast, CROSSTALK is designed to achieve low latency in both the presence and absence of faults. IGOR [28] has similar goals to CROSSTALK. However, CROSSTALK does not require replicas to perform multiple computations on different cores, resulting in increased schedulability.

Leveraging the network topology. Several protocols exploit network redundancy in their designs. One class of protocols, fault-tolerant routing protocols [198, 199, 200], are concerned with creating a reliable communication channel between a sender and receiver. However, importantly, these protocols cannot tolerate Byzantine end nodes. Another class of protocols aims to solve consensus in directed graphs [201, 202]. However, they are aimed at maximizing resilience in *arbitrary* network topologies. As a result, CROSSTALK is significantly more efficient when used in realistic embedded networks.

Other solutions leverage unique aspects of embedded networks to increase performance. One patent [203] inserts skew between redundant time-triggered messages to ensure agreement among receivers. Other protocols vote on message copies received on redundant planes to mask transmission faults [71, 173]. Importantly, unlike CROSSTALK, both approaches can only tolerate a single faulty device (switch or end node) at a time.

Agreement in the network. Several protocols modify the switches or use software-defined networks (SDNs) to execute an agreement protocol in the network or otherwise guarantee message ordering [204, 205, 206]. However, these protocols are all limited to tolerating crash faults of both the switches and end nodes. In contrast, CROSSTALK can tolerate omissive switches and Byzantine faulty end nodes, and can be used without requiring SDNs or switch modifications.

Impossibility results. Fischer and Lynch proved $f+1$ communication rounds are needed for agreement in synchronous systems with point-to-point channels [27]. Despite its "single round" design, CROSSTALK does not violate this result. Rather, the required rounds ($g + 1$ for CROSSTALK) take place in the network as messages are forwarded between planes. The FLP result [207] showed agreement is impossible in asynchronous systems subject to crash faults. We use similar intuition

in our impossibility proof. However, the proof is distinct. Specifically, while synchronous systems can tolerate timing faults with $f + 1$ rounds of exchange between replicas, as well as Byzantine faults in the *value* domain without requiring any rounds between replicas (given sufficient rounds occur in the network), we prove it is not possible for synchronous systems to tolerate timing faults without rounds between replicas (regardless of rounds in the network).

3.8 Conclusion

BFT SMR is essential for tolerating faults in critical embedded systems. However, existing BFT SMR protocols force designers to choose between high latencies and substantial computation overheads due to redundant computations. We presented CROSSTALK, a new BFT SMR protocol that leverages redundancy that already exists in embedded networks to minimize latency without requiring extra computation. Our evaluation showed that CROSSTALK improves system schedulability by 2.13–4.24×, and can increase the resilience and control performance of real systems. In the future, we believe more opportunities will exist for protocols like CROSSTALK to exploit emerging networking trends to increase the performance of real-time embedded systems.

CHAPTER 4

PCSPOOF: Compromising the Safety of Time-Triggered Ethernet

4.1 Introduction

Increasingly, embedded systems are using *mixed-criticality* network technologies that allow traffic with different timing and fault tolerance requirements to coexist in the same physical network [31, 32, 33, 34]. These technologies let designers reduce size, weight, power, and cost by sharing the same network between critical and non-critical parts of the system. For example, aircraft can share one network between vehicle control systems and passenger Wi-Fi and entertainment systems [208, 209]; spacecraft can share one network between life support systems and onboard experiments [11, 133]; and manufacturing plants can share one network between robot control systems and data collection systems [210].

One of the most successful mixed-criticality network technologies is *Time-Triggered Ethernet (TTE)* [32]. Today, TTE serves as the network backbone for several spacecraft, including NASA's Orion capsule [59], NASA's Lunar Gateway space station [11], and ESA's Ariane 6 launcher [52]. TTE is also widely used in aircraft [65, 66, 67], energy generation systems [39], and industrial control systems [211, 212], and is a leading contender to replace CAN bus and FlexRay as the standard network technology in future automobiles [213, 214].

TTE has several properties that make it attractive for safety and mission-critical applications. Most notably, TTE follows a *time-triggered (TT)* paradigm, in which devices are tightly synchronized, and they send messages and execute software according to a predetermined schedule. This TT approach reduces message latencies to hundreds of microseconds and jitter to near-zero [95, 101], making TTE appropriate for even the tightest control loops. TTE also provides fault tolerance by replicating the whole network to form multiple *planes*, and by forwarding messages over all planes simultaneously [215].

In addition, TTE enables mixed-criticality architectures by being 100% compatible with standard Ethernet [216]. This means that *non-critical* systems, which typically use standard Ethernet

hardware to lower costs [217], can send messages over the same cabling as the critical TTE devices. Unlike TT traffic, standard Ethernet traffic is forwarded on a *best-effort (BE)* basis, filling in space *around* the TT traffic [216]. Also, standard Ethernet traffic typically only travels over a single network plane, so does not have any fault tolerance guarantees [11].

A key aspect of TTE's mixed-criticality design is that the TT part of the system is *isolated* from the BE part. In other words, no matter how the BE devices behave, they should not be able to disrupt the synchronization between TTE devices or the timely/successful delivery of TT traffic [36]. This isolation is commonly used as a justification for several cost-cutting measures, including: (1) procuring BE devices from relatively untrusted (but low cost) suppliers [218, 219]; (2) relaxing security requirements for BE devices [220]; and (3) reducing the scope of analysis and certification of a system to focus solely on the TTE devices [221]. For example, on NASA spacecraft, onboard experiments are often provided by university research groups, are operated by the university students with minimal NASA involvement, and are not considered in safety reviews or the certification process of the overall vehicle [222, 223].

In this paper, we present PCSPOOF, a new attack that breaks TTE's isolation guarantees for the first time — allowing a single malicious BE device on a single plane to disrupt synchronization and communication between TTE devices on all planes. PCSPOOF is based on two key observations:

First, it is possible for a malicious BE device to *infer* private information about the TTE network that is needed to construct valid TTE synchronization messages, called *protocol control frames (PCFs)*. For example, an attacker can exploit the fact that (1) all PCFs in the network contain a common identifier, and that (2) BE devices are *not allowed* to send messages containing this identifier. Such messages are simply dropped by the switches. Therefore, by issuing phony ARP [224] requests to other BE devices (e.g., routers), tricking them into sending messages containing possible identifiers, then checking which of the messages are dropped, an attacker can quickly determine the *actual* identifier used in the PCFs (see §4.4.1.2).

Second, using a few extra circuit components, a malicious BE device can conduct electromagnetic interference (EMI) into a TTE switch and trick the switch into forwarding PCFs that the BE device is not allowed to send. In particular, by conducting EMI into the switch over an Ethernet cable, resulting in radiated EMI *inside the switch*, a BE device can cut the header off of a BE message in flight, revealing a malicious PCF in the message's payload (a type of packet-in-packet attack [225]). Since the EMI radiates from inside the switch, the attack cannot be prevented by conventional switch and cable shielding. Additionally, since the source of the radiated EMI (the port connected to the attacker) is close to the internal switch components (1–10 cm), the EMI requires relatively little power to be effective, and therefore can be generated with a small circuit (e.g., a 2.5 cm $\times$ 2.5 cm square, see §4.4.2.5). As we show in §4.3, such a circuit could reasonably be hidden in a BE device and integrated into a TTE system without detection.

Finally, our work reveals a flaw in modern TTE devices that makes them especially susceptible to PCSPOOF's EMI injection. In particular, while modern devices verify the *contents* of the preamble that precedes each message, they do not verify the preamble *length*. An attacker can exploit this by sending very large BE messages, which are more likely to reveal PCFs when corrupted by EMI, without the PCFs being rejected by downstream TTE devices (see §4.4.2.2).

We evaluated PCSPOOF on a real TTE testbed. Our results show that PCSPOOF can successfully inject a malicious PCF in 10–20 s. A single injection can cause TTE devices to lose synchronization for up to a second and fail to transmit tens of TT messages — both of which can cause the failure of critical systems [29, 26]. Moreover, in the worst case, PCSPOOF causes these outcomes simultaneously for *all TTE devices in the network* (see §4.6.2). We also evaluated PCSPOOF on an avionics testbed for a real spaceflight mission; our results show that PCSPOOF can threaten mission success and safety from a single BE device, such as those used in an onboard research experiment developed by a university.

In summary, we make the following contributions:

- PCSPOOF: the first attack to break TTE's isolation guarantees; PCSPOOF can disrupt critical TT systems from a single malicious BE device (§4.4).
- An extensive study of the susceptibility of TTE hardware to PCSPOOF, which reveals a security flaw in the implementation of modern devices (§4.4.2.2).
- A detailed experimental evaluation of PCSPOOF on a real TTE testbed that assesses the probability and impact of successful attacks (§4.6).
- A case study demonstrating the effect of PCSPOOF on a simulated spaceflight mission (§4.6.4).
- A detailed description of methods to make TTE systems more resilient to PCSPOOF (§4.7).

4.1.1 Responsible Disclosure

We disclosed our attack to several organizations using TTE for critical applications, including NASA, ESA, Northrop Grumman Space Systems, and Airbus Defense and Space. All organizations acknowledged the seriousness of the attack and several are implementing mitigations we suggest in this paper. Our work is also making NASA reconsider the way that onboard experiments and commercial-off-the-shelf devices are verified to be safe.

We also disclosed our attack to TTTech Computertechnik AG, the leading provider of TTE equipment and chip-IP. TTTech acknowledged the attack and is working on hardware, configuration, and tooling updates to mitigate it.

Moreover, the SAE AS-2D2 committee is working to mitigate our attack by revising the TTE standard (SAE AS6802) to allow PCFs up to 1518 bytes (the max Ethernet frame size). The use of

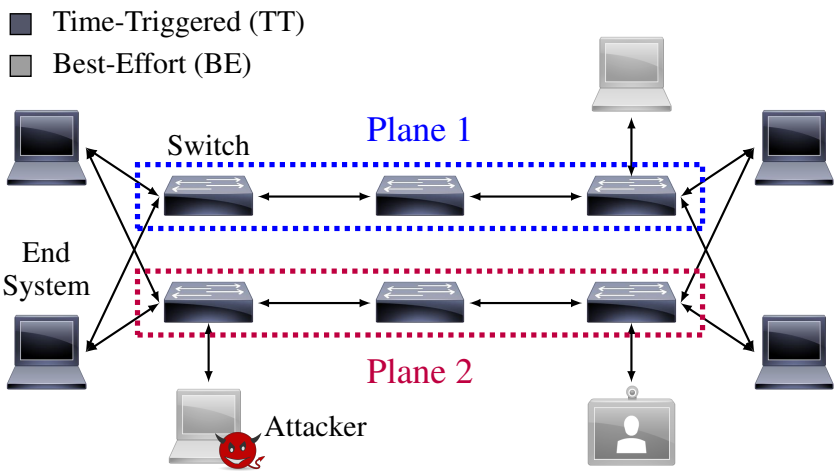

Figure 4.1: **Example TTE Network** — Example of a typical fault-tolerant TTE network with redundant planes. The attacker controls a single BE device on a single plane.

max-sized PCFs would prevent the PCF injection method we use (cutting the header off a frame in flight) from producing a PCF that is accepted by TTE devices.

4.2 Background

In this section, we describe Time-Triggered Ethernet and the synchronization protocol it is based on.

4.2.1 Time-Triggered Ethernet (TTE)

TTE networks contain two types of devices, *switches* and *end systems*, where each end system consists of a host processor (which runs user software) and a TTE network interface card (NIC) [70]. Like in standard Ethernet, the switches forward messages, or *frames*, between the end systems. For redundancy, the entire network is replicated, creating multiple paths between each end system [215]. We refer to each redundant network as a *plane*. End systems send frames simultaneously through all planes, and receivers accept the first frame that arrives. This approach allows the system to continue operating even after multiple failures. An example of a typical TTE network is shown in Figure 4.1.

TTE networks utilize a *time-triggered* design, in which all TTE devices are tightly synchronized, and the behavior of the network is determined by a global schedule [32]. The schedule is built offline and loaded onto each TTE device before the system is deployed. The schedule specifies when TT frames are forwarded and expected to arrive. In addition, it specifies the timing of interrupts that software running on the end systems use to coordinate their actions [11].

This design reduces network latency and jitter to a minimum, resulting in very predictable system performance [95, 101].

Additionally, TTE networks are compatible with standard Ethernet [216]. This allows designers to use (inexpensive) standard Ethernet hardware for devices without strict timing or fault tolerance requirements, like passenger entertainment systems in airplanes or monitoring systems in power plants [226, 227]. These devices can plug directly into TTE switch ports and treat the TTE network exactly like a standard Ethernet network. We refer to these standard Ethernet devices as *best-effort (BE)* devices, since the switches forward their traffic around the pre-scheduled TT traffic only as bandwidth allows.

TTE uses several mechanisms to *isolate* the TT traffic from the BE traffic, including not allowing BE traffic to be transferred in windows reserved for TT traffic, and storing TT and BE frames in separate switch buffers [216]. Together, these mechanisms aim to ensure malicious BE devices have no way to interfere with the TT part of the system [36].

On the surface, attacks that break TTE's isolation guarantees seem impossible. For example, since the TTE switches reserve bandwidth for TT traffic and store TT and BE traffic separately, flooding the network from a BE device cannot cause TT traffic to be delayed or dropped [228]. The switches will simply drop the excess BE traffic to allow TT traffic to flow. Also, a BE device cannot generate its own TT traffic, since the switches ignore any TT traffic that is not defined in the pre-loaded schedule [229]. Even in the extreme case where a malicious BE device somehow kills the switch it is connected to, the TTE devices will continue to operate without disruption over the redundant planes.

4.2.2 The TTE Synchronization Protocol

TTE networks rely on a synchronization protocol to enable communication between devices [32], and PCSPOOF works by *disrupting* this protocol. Below, we describe the synchronization protocol and why it is susceptible to PCSPOOF.

There are two main roles that TTE devices can take in the synchronization protocol: (1) *sync master* and (2) *compression master* [32]. Typically, a subset of the end systems act as sync masters (based on the required fault tolerance), and 1–2 switches per plane act as compression masters [215]. The remaining devices act as *sync clients*, which use the synchronized time base, but do not help maintain it [32].

In general, synchronization works by continuously exchanging special messages, called *protocol control frames (PCFs)*, between the devices [32]. This exchange is repeated at regular periods called *integration cycles* [32]. At the start of each integration cycle, each sync master sends a PCF with its local clock value to the compression masters. The compression masters average the

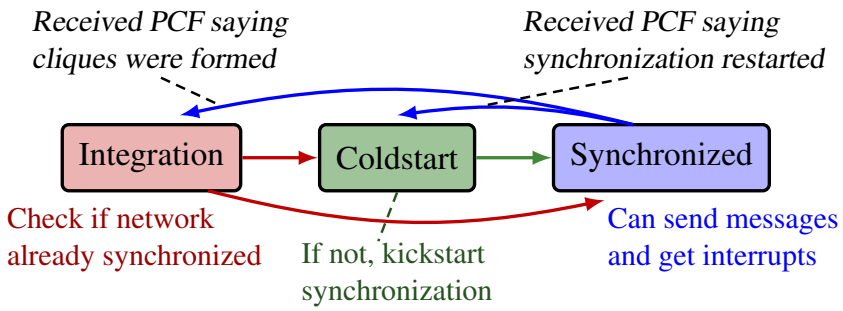

Figure 4.2: **Synchronization State Machine** — Simplified version of the state machine that sync masters execute in the TTE synchronization protocol.

received clock values, then send out the resulting clock value in a new PCF to the sync masters and clients, which use it to correct their local clocks.

In addition, receiving certain PCFs from a compression master can cause sync masters and clients *to lose synchronization*. Specifically, a special "coldstart acknowledgement" PCF tells a sync master that another sync master detected synchronization was lost and is reestablishing it [32]. Similarly, the contents of a normal (i.e., integration) PCF can tell a sync master/client that *cliques* — multiple groups not synchronized to each other — have formed [69]. In either case, the sync master/client briefly loses synchronization and attempts to resynchronize with the network. Figure 4.2 shows how these PCFs impact the sync master state machine.

Because a single PCF can knock devices out of synchronization, significant effort has been spent to ensure all PCFs generated by compression masters can be trusted. For example, the TTE standard requires each compression master to be a self-checking pair — i.e., it only produces a PCF if two independent processors agree on the contents [32].

In PCSPOOF, we exploit the trust the TTE protocol puts in compression masters. By injecting PCFs into the network that look like they came from real compression masters, an attacker can make sync masters/clients repeatedly lose synchronization. We note that because synchronization loss between non-faulty devices is so rare in practice (requiring multiple specific failures), systems are often not designed to tolerate it [70, 28, 230]. Also, even systems that do tolerate synchronization loss are not designed to tolerate the repeated synchronization loss caused by PCSPOOF (see §4.6.4).

4.3 Threat Model

We assume a standard multi-plane TTE network like those used today in spacecraft [59, 11, 52], aircraft [67], and energy generation systems [231] (see §4.2.1). The network includes both TTE and BE devices. For fault tolerance, the TTE end systems are connected to and communicate

over all redundant planes simultaneously. In contrast, BE devices typically do not have any fault tolerance requirements, so often connect to only a *single* switch in a *single* plane in order to save wiring mass and cost [11].

We assume the attacker has the ability to execute malicious software on a *single* BE device, including sending and receiving *standard* Ethernet messages. The connectivity of the attacker's BE device is shown in Figure 4.1. In addition, we assume the BE device includes additional circuit components that allow it to conduct electromagnetic interference (EMI) through its Ethernet cable and into the switch. As we show in §4.4.2.5, such a circuit can be constructed from as little as 5 circuit components, and can take up as little as 2.5 cm $\times$ 2.5 cm on a single-layer printed circuit board.

There are two realistic ways these assumptions can be satisfied in practice: (1) the BE device is supplied by a malicious third-party and integrated into the TTE network at *design time*, or (2) the BE device is connected to the TTE network *after* the network is deployed.

First, the system integrator could obtain the BE device from a malicious third-party and integrate it into the system at design time. In TTE networks, non-critical functions are commonly performed using commercial-off-the-shelf (COTS) devices to reduce costs [11]. This is true even in critical industries like spaceflight and aviation [37, 38]. Unlike critical TTE devices, which come from secure supply chains, COTS devices come from unsecured supply chains that are susceptible to tampering [37]. Also, the companies that design COTS devices are often relatively untrusted, and do not typically follow any formal development process to ensure safety and security (e.g., RTCA DO-254) [37]. In addition to COTS suppliers, BE devices in spaceflight commonly come from university research groups and laboratories [222]. In any of these organizations, a rogue employee, student, or team could alter the device with the malicious circuit and software [232, 233]. A simple *ticking timebomb* [232] trigger, which enables malicious behavior after a configurable amount of time, could be used to activate the circuit and software after the network is deployed — without requiring any input from the attacker.

Even in critical industries like spaceflight and aviation, such malicious hardware and software is not likely to be caught by the system integrator. The reason is that, besides through well-known means like causing an explosion or fire, there has been no known way for non-critical BE devices in a TTE network to disrupt the operation of critical TT devices. As a result, verification of these BE devices is limited to ensuring they do not contain dangerous substances, will survive the operating environment (e.g., vibration and thermal qualification), and perform their intended function [223, 37]. For example, explosives are detected by swabbing, and other dangerous materials are detected through outgassing tests [234]. However, no detailed analysis of the circuit components, circuit layout, or software is performed [223, 235, 236, 37]. The malicious circuit and software needed for PCSPOOF cannot be detected by such basic safety testing. Moreover, a ticking timebomb trigger

could simply delay activating the malicious circuit and software until after all functional testing is completed [232].

Second, instead of compromising the system at design time, an attacker could connect a malicious BE device to the network after the network is deployed. For example, TTE allows a factory to share switches between the assembly line and non-critical hardware, like laptops used for monitoring and analysis [212, 237]. If an employee could be tricked into plugging a malicious device (e.g., a USB to Ethernet dongle) into one of these switches, for example, through a supply chain attack (as above) or social engineering, they could inadvertently disrupt the control of critical plant processes and halt production of the entire facility. Alternatively, consider a future commercial airplane that shares a TTE network between the passenger cabin and vehicle control systems.[1] Modern airplanes contain exposed seat electronics boxes under the seats for connecting entertainment units to the passenger network [227]. If a passenger has knowledge of the connectors used, they could secretly disconnect one of these electronics boxes during a flight, plug in a malicious device [227], and interfere with the safe operation of the aircraft — even if the vehicle control data is all encrypted.

We stress that in all the above cases, an attacker with a connection to only a *single* network plane can disrupt TTE devices *throughout* the network and on *all* planes (see §4.6). We also note that the connection from the attacker to the TTE network could span more than a single Ethernet cable. The attack works even if the connection is made via a series of several cables, patch panels, and Ethernet jacks.

Lastly, we stress that, besides the single BE device, the attacker has no access to or knowledge of any part of the TTE network. In particular, the attacker has no information about the TT network schedule or the position of devices within the network. Additionally, the attacker cannot receive any TT messages, or access any telemetry or diagnostic information from the TTE switches or end systems.

4.4 Design

This section describes PCSPOOF, the first attack capable of disrupting critical TTE traffic flows and interrupts from a BE device. PCSPOOF achieves this goal by disrupting the TTE synchronization protocol [32]. Disrupting synchronization lets PCSPOOF potentially disrupt *any TT traffic flow*, without needing the attacker to know what traffic flows exist or what they are used for. It also makes PCSPOOF broadly applicable, since all TTE networks use the same synchronization protocol [32].

[1] While, to our knowledge, no commercial airplanes *currently* share switches between the passenger cabin and critical devices, device manufacturers have advocated that TTE's isolation guarantees would make such sharing safe, while reducing size, weight, power, and cost [208, 209].

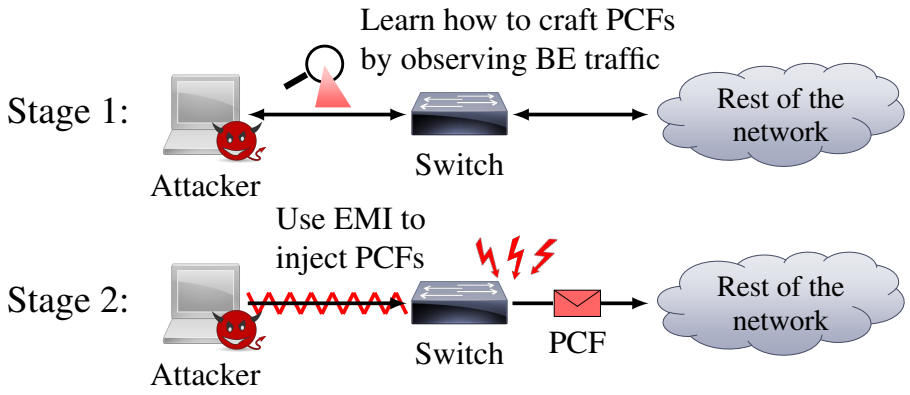

Figure 4.3: **Overview of PCSPOOF** — High-level design of PCSPOOF showing the two stages of the attack. First, the attacker learns how to construct malicious synchronization messages. Then, the attacker uses EMI to inject them into the network.

Of course, disrupting the TTE synchronization protocol from a BE device should be impossible. The protocol is formally verified to work correctly in spite of a malicious TTE end system and any number of malicious BE devices [39].

To overcome this challenge, we use two key observations: (1) an attacker can deduce secret information, known only to TTE devices, in order to create malicious protocol control frames (PCFs), and (2) an attacker can use EMI generated from a BE device to inject these malicious PCFs into the network and get them accepted by TTE devices.

Figure 4.3 gives an overview of PCSPOOF. The attack proceeds in two stages. In the first stage, the attacker learns *how* to craft authentic-looking PCFs, which requires two pieces of information. The first is the *critical traffic marker*, a special bit pattern found at the start of every PCF. The second is a *virtual link ID*, which identifies the switch that sends a given PCF. As we will show in §4.4.1, the first value is found *indirectly* by observing how the switches respond when forwarding different types of BE traffic. The second value is inferred using knowledge of common network scheduling practices and public hardware documentation.

In the second stage, the attacker injects malicious PCFs into the network. However, since switches block PCFs from BE devices, this requires somehow *bypassing* the switch. To accomplish this, we leverage the fact that, by conducting EMI through the Ethernet cable and into the switch, it is possible to induce link resets *in different switch ports*. These link resets can be used to "transform" BE traffic, which the attacker is allowed to send, into PCFs *as they leave the switch*. Since the transformation happens downstream of the switch logic, it cannot be prevented by extra switch error checking or self-checking pair processors [238].

Next, we describe in detail how to craft malicious PCFs (§4.4.1) and inject them into the network (§4.4.2).

4.4.1 Stage 1: Crafting Malicious PCFs

In the first stage of PCSPOOF, the attacker learns how to craft authentic-looking PCFs. Below, we describe the structure of a PCF, as well as how to obtain the information necessary to make injected PCFs look legitimate.

4.4.1.1 Anatomy of a PCF

In general, the structure of a PCF is the same as a standard minimum-sized IEEE 802.3 Ethernet frame [32, 239]. However, unlike standard Ethernet frames, PCFs are not forwarded according to a destination media access control (MAC) address. Instead, the first 6 bytes of the frame, which would normally contain the destination MAC address, are replaced with the following two fields:

- *Critical Traffic Marker* — A special value used to identify all PCFs and TT traffic in the network.
- *Virtual Link ID* — Identifies the source of the PCF. For our purposes, it typically identifies a switch.

In order for a PCF to be seen as legitimate, these fields must both match values specified in the network schedule loaded onto the TTE devices when the network was deployed.

In addition, PCFs contain the following fields. However, unlike the critical traffic marker and virtual link ID, it is easy for an attacker to pick suitable values for these fields. This is because either (1) the range of acceptable values is very small or (2) the fields are simply not checked by the TTE hardware.

- *Source MAC Address* — Identifies the source of a frame. Note, however, that since the virtual link ID *also* identifies the source, this field is not checked in practice. We tested a large array of modern and legacy TTE hardware (listed later in Table 4.1), and found that it is never used. For our purposes, it can be set to any value.
- *EtherType / Length* — Indicates that the frame is a PCF. It must be set to 0x891d [32].
- *Integration Cycle* — Tracks the current synchronization period. It must fall within a range defined in the schedule [32]. However, a value of 0x0 is always valid.
- *Membership New* — Identifies which sync masters contribute to the synchronized time base. When injecting integration PCFs, setting this to a high enough value tricks devices into detecting a clique [228]. In our tests, a value of 0x1 was always sufficient.
- *Sync Priority* — Must match the compression master priority in the network schedule. Most TTE networks use only one priority [240], so the value 0x1 is usually correct. Otherwise, the hardware limits the possible values to a small range (e.g., 0x1–0x3) [241, 242].

- *Sync Domain* — Identifies a specific set of synchronized devices in the network. Most networks have only one sync domain [215], so the value 0x0 is usually correct. Otherwise, the hardware limits the possible values to a small range (e.g., 0x0–0x7) [241].
- *Type* — Identifies the type of PCF. It must be set to 0x08 for a coldstart acknowledgement PCF or 0x02 for a normal integration PCF (see §4.2.2).
- *Transparent Clock* — Tracks delay in the switches. It must fall within a range determined by the hardware. In our tests, a value of 0x0 was always valid.

Since the critical traffic marker and virtual link ID are the only fields that are difficult for an attacker to select, we focus the rest of this section on how an attacker can determine them.

4.4.1.2 Finding the Critical Traffic Marker

Generating authentic-looking PCFs requires the attacker to find the critical traffic marker used in the network schedule. To accomplish this, they can take advantage of the following rules, which TTE switches use when determining how to forward frames [32].

- If the destination MAC address contains the critical traffic marker, the virtual link ID is valid, and the frame comes from a known TTE device, the frame is forwarded according to the TTE schedule.
- If the destination MAC address contains the critical traffic marker but the virtual link ID is invalid, or the frame comes from a BE device, the frame is dropped.
- If the destination MAC address does *not* contain the critical traffic marker, the frame is forwarded according to the rules of IEEE 802.3 (standard Ethernet) [239].

From these rules, we see that all frames sent by BE devices should be delivered (as bandwidth allows), *except* those containing the critical traffic marker. Thus, an attacker can *infer* the critical traffic marker by tricking other BE devices into sending the attacker frames containing *possible* critical traffic markers and checking which frames *do not arrive*. Below, we describe one method for accomplishing this by abusing the Address Resolution Protocol (ARP) [224], which is used by nearly all BE Ethernet devices.

To start, the attacker must find the IP address of another BE device in the network, which we refer to as the *target*. Any device can be used, such as a router used for passenger Wi-Fi in an airplane, or an inventory management computer in a factory. To get the target's IP address, the attacker sends Internet Control Message Protocol (ICMP) echo requests to all IP addresses in the subnetwork and sees who responds. The standard `fping` utility can do this out of the box, and the process takes tens of seconds even in large networks.

Next, the attacker cycles through a list of possible critical traffic markers. For each one, the attacker sends an ARP request to the target saying "Which MAC address goes with IP X? Tell MAC Y," where X is the IP address of the target, and Y is the MAC address containing the critical traffic marker to test. Upon receiving this message, the target replies to MAC address Y with the target's MAC address.

Assuming the attacker spoofs their source MAC address as Y in each ARP request, each reply for which Y *does not* contain the critical traffic marker is forwarded to the attacker. This is because, with each ARP request, the switch learns to associate MAC address Y with the attacker's port. Otherwise, the reply is dropped. Thus, the attacker can identify the critical traffic marker by sending an ARP request for each possible critical traffic marker and checking which request gets no reply. To handle the fact that BE messages can be dropped for reasons unrelated to the critical traffic marker (e.g., buffer overflows), the attacker repeats this process in phases; in each phase, only testing critical traffic markers for which no reply was received previously.

There are only around 1 billion possible critical traffic markers [31, 32], so brute forcing the critical traffic marker is fast in practice. We used a Raspberry Pi 4 to find the critical traffic marker in multiple representative spacecraft networks with real surrogate spaceflight hardware. It took only 6–7 hours on average when sending ARP requests at 100 Mbps, and 24 hours when sending requests at 25 Mbps.

We note that, since the critical traffic marker is part of the TTE schedule, it typically does not change over a system's lifetime [173]. The reason is that the schedule typically undergoes a thorough verification and validation process [243, 244, 245]. Changes to the schedule can require repeating this process, which is expensive and time consuming [173, 243]. This means the attacker does not need to determine the critical traffic marker all at once, or at the same time as they execute the rest of the attack.

Finally, we acknowledge that if all BE devices in the network were configured to use static MAC/IP mappings and drop ARP requests, the method described above would not work. However, when testing on real COTS devices used in flight (e.g., routers), we found that these devices respond to ARP requests. Also, our discussions with avionic designers have revealed industry is explicitly embracing ARP to avoid the complexity of managing static MAC/IP mappings.

However, we note that even if ARP is disabled on all BE devices, an attacker can still easily find the critical traffic marker by using two malicious devices. One device simply sends frames with every possible critical traffic marker to the other device. Since the switch does not know the identity of every BE device, it will flood each frame out of all ports. The second device then tracks which frames are not received. This method cannot be prevented in any TTE switch that has default routes enabled, and we have successfully tested it on a variety of real spaceflight switches.

4.4.1.3 Finding the Virtual Link ID

The last piece of information the attacker needs to generate PCFs is the virtual link ID corresponding to a *real* compression master (i.e., switch) that generates PCFs in the network. That way, once a PCF is injected on a given network segment, downstream TTE devices cannot tell that the injected PCF is illegitimate.

Theoretically, the virtual link ID could be any 16-bit number, so there are 65536 possibilities. However, there are two pieces of information an attacker can use to reduce the number of possible virtual link IDs to 2 or fewer.

First, even though there are *theoretically* 65536 possible virtual link IDs, existing switches do not support that many. Also, the number of IDs that switches *do* support is public information. For example, TTTech's Space ASIC, which is used in NASA's Gateway and ESA's Ariane 6 launcher, only supports 4096 virtual link IDs [246]. TTTech's aircraft switches are limited to the same number [242].

Second, existing TTE scheduling tools use *extremely* predictable rules for assigning virtual link IDs to PCFs. For example, the most popular scheduling tools assign virtual link IDs in *reverse order* from the maximum value supported by the hardware (i.e., 4096) [241]. Virtual link IDs for sync masters (i.e., end systems) are assigned first, with a different ID used for each of three PCF types [32]. Virtual link IDs for compression masters (i.e., switches) are assigned next, with each switch using the same ID for all PCF types.

As the number of sync masters and compression masters in a system is predictable, so is the virtual link ID the attacker needs. For example, existing switches support at most 8 sync masters [247]. Additionally, most TTE systems have one compression master per plane. We are not aware of any existing system with more than two compression masters per plane. Thus, in a large system like a spacecraft or aircraft, the virtual link ID needed by the attacker is likely $4095 - (8 \times 3) = 4071$, and more rarely 4070.

As we show in §4.6.1, PCF injection is so fast that, even if the attacker cannot determine the virtual link ID with certainty, they can simply try all possible values until injection succeeds. Also, we note that, like the critical traffic marker, virtual link IDs are part of the TTE schedule [241], so are unlikely to change once a system is deployed [173].

4.4.2 Stage 2: Injecting PCFs into the Network

Now that the attacker knows how to construct a PCF, they need a way to inject the PCF into the network. The attacker cannot simply send the PCF directly, since all PCFs sent from BE devices will be dropped by the switch. To overcome this challenge, PCSPOOF uses EMI to "transform" a BE frame, which the BE device is allowed to send, into a PCF.

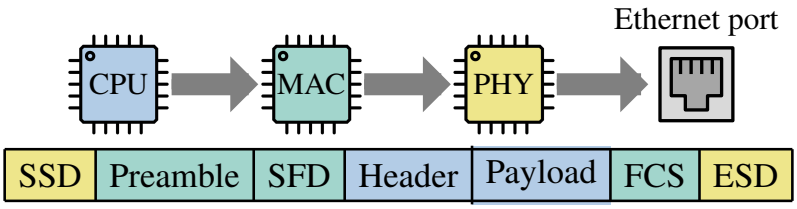

Figure 4.4: **Ethernet Encapsulation** — Ethernet frames generated by software on the host processor (CPU) are encapsulated by both MAC and PHY.

In order to perform this transformation, the attacker stores a PCF *inside* the payload of a benign BE frame. By carefully corrupting the BE frame in transit, it is possible to then trick the switch into sending the PCF. Attacks that use this general approach, hiding a malicious message inside a benign message, are called *packet-in-packet attacks* [225].

Below, we describe what makes Ethernet susceptible to packet-in-packet attacks, how TTE hardware can prevent these attacks, and how PCSPOOF defeats these defenses.

4.4.2.1 Packet-in-Packet Attacks on Ethernet

To understand why Ethernet is susceptible to packet-in-packet attacks, it is first necessary to understand how Ethernet frames are generated and interpreted by Ethernet devices.

Two types of integrated circuits are needed for a device to send and receive Ethernet frames — the *media access controller (MAC)* and the *physical layer transceiver (PHY)*. The MAC is responsible for assembling and validating Ethernet frames, and for passing them between the host processor and PHY. The PHY is responsible for translating these frames between bytes understood by the MAC, and special symbols used at the physical layer, and for writing and reading these symbols to and from the Ethernet wiring.

Figure 4.4 shows the path of a frame through the MAC and PHY. Each circuit adds additional information to the frame. Specifically, the MAC adds the *preamble* (7 bytes of 0x55), which allows a receiver to "lock on" to the incoming frame, as well as the *start frame delimiter (SFD)* (1 byte of 0xd5), which signals the start of the Ethernet header. The MAC also adds the frame check sequence (FCS) at the end of the frame. The PHY adds a *start-of-stream delimiter (SSD)* to signal the start of the transmission, as well as an *end-of-stream delimiter (ESD)* to signal the end of the transmission.

When receiving a frame, the PHY waits until it sees the SSD, at which point it tells the MAC the preamble is starting. The MAC then reads the preamble until it gets to the SFD byte, at which point it reads in the Ethernet frame. When the PHY receives the ESD symbol, it again signals the MAC, at which point the MAC knows the frame is complete. The last 4 bytes read by the MAC

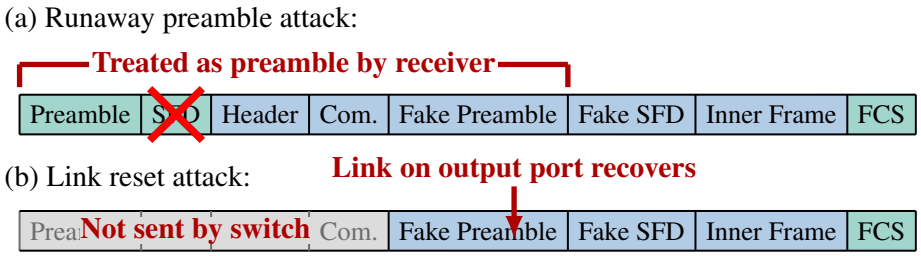

Figure 4.5: **Packet-in-Packet Attacks** — Two types of packet-in-packet attacks have been demonstrated on wired Ethernet.

are treated as the FCS.

This design has been shown to be susceptible to two types of packet-in-packet attacks, which we show in Figure 4.5. In the figure, assume an attacker wants to send a malicious frame (the "inner frame") past a switch to some receiver. However, the switch is configured to drop this frame.

The first type of packet-in-packet attack is a *runaway preamble* attack. Here an attacker exploits the fact that, if a frame's SFD byte is corrupted after the frame is forwarded by the switch, the receiver's MAC will treat this SFD byte (and any following bytes) as preamble [248]. Many MACs do not check that the preamble matches the expected pattern (all 0x55). Thus, by placing a *fake* SFD byte in the frame's payload, immediately before the inner frame, an attacker can trick a receiver into reading the inner frame [248].

The second type of packet-in-packet attack uses *link resets* [249]. In Ethernet, the PHY continuously checks for link pulses and idle symbols produced by the device on the other side of the cable [239]. If these indicators are disrupted, the link is "lost," and the PHY stops transmitting frames. However, the MAC is *not aware* of link status changes and will continue transmitting [249]. An attacker can exploit this by sending a frame (the same structure as above) through the switch while the link is down on the outgoing port. If the attacker is lucky, the link will recover in the middle of the fake preamble being transmitted by the MAC, resulting in only the inner frame actually being sent by the switch.

Note that for either approach to work, the FCS of the original frame must be made to match that of the inner frame. For this, an attacker can exploit the fact that, by adding a 4-byte FCS complement to a frame's payload, it is possible to force the frame's FCS to any value [248]. For more details on calculating and using the FCS complement, see past work on packet-in-packet attacks [248, 249].

Device	Preamble		
	Too Long	Too Short	Non-0x55
TTTech Dev. 1G SW	Y, $\leq$11 bytes	N	N
TTTech PMC 1G NIC	Y, $\leq$11 bytes	N	N
TTTech A664 Lab SW	Y, $\leq$1451 bytes	Y, $\geq$3 bytes	Y, 1st two bytes
TTTech OBC HiRel SW	Y, $\leq$1451 bytes	Y, $\geq$3 bytes	Y, 1st byte
TTTech Space Lab SW	Y, $\leq$1451 bytes	Y, $\geq$3 bytes	Y, 1st byte
TTTech A664 Lab NIC	Y, $\leq$1451 bytes	Y, $\geq$3 bytes	Y, 1st two bytes

Table 4.1: **TTE Device Testing** — The table indicates whether TTE switches (SWs) and network cards (NICs) accept PCFs with non-standard preambles. "Too Long, Y, $\leq$ 11 bytes" means the device accepts PCFs with longer-than-normal preambles up to 11 bytes. Devices labeled "1G" are an older generation of devices.

4.4.2.2 Susceptibility of TTE Hardware

We tested a wide variety of modern and legacy TTE devices to determine how susceptible they are to both types of packet-in-packet attacks. For each device, we used an XMOS XCORE-200 [250] to generate PCFs with various non-standard preamble and SFD patterns, and determined whether the PCFs were accepted.

Our results are shown in Table 4.1. In all cases, a PCF is only accepted if either (1) the preamble contains only 0x55 bytes, or (2) the preamble *starts* with one or two non-0x55 bytes, but the rest of the preamble is all 0x55 bytes.

These results show that, unlike standard Ethernet hardware [248], TTE hardware can *completely prevent runaway preamble attacks*. There are two reasons. First, the original frame's header, as well as the FCS complement, *will not* be treated as preamble by the receiver unless they both only contain 0x55 bytes. Second, the switch can prevent the frame from ever being forwarded to the receiver by filtering all BE traffic with a destination MAC starting with a valid preamble/SFD pattern (e.g., 0xd5, 0x55d5, 0x5555d5). This is a capability of all TTE switches we have tested.

In contrast, the fact that modern TTE devices accept frames with such long preambles makes them *very susceptible* to link reset attacks. The reason is the attacker can send packet-in-packet frames with 1000+ bytes of fake preamble, maximizing the chance that the link recovers while this fake preamble is being forwarded, *while still* ensuring the resulting PCF is accepted by downstream TTE devices.

4.4.2.3 Enabling PCF Injection

In the past, link reset packet-in-packet attacks on wired Ethernet have been considered impractical, since there was *no known way* for an attacker to cause link resets without physically manipulating the network — e.g., unplugging a cable or rebooting a switch [249].

In contrast, PCSPOOF allows a *networked device*, controlled by the attacker, to cause link resets

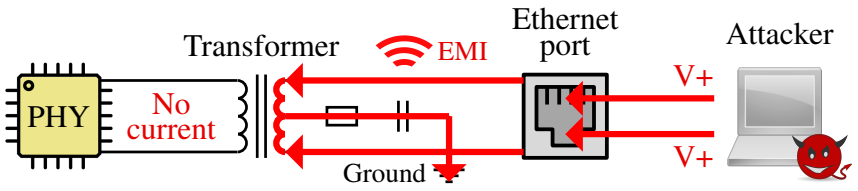

Figure 4.6: **EMI Injection** — Effect that a common mode surge on an Ethernet twisted pair has on the inside of a switch.

between the switch and other devices. In general, it accomplishes this by conducting electrical noise into the switch over the Ethernet cable, which results in radiated EMI inside the switch and disrupts the operation of the PHYs on other switch ports.

Figure 4.6 shows how this EMI is generated in more detail. The figure depicts a malicious BE device connected to a TTE switch. For simplicity, we assume the connection uses a twisted-pair Ethernet cable (e.g., 100BASE-TX), which is the most popular choice today due to cost and reliability [251, 252]. When the cable is plugged in, the wires in the cable are electrically connected to copper traces on the switch printed circuit board (PCB) inside the switch chassis.

Faraday's Law tells us that, by causing rapid high-voltage surges on the wires in the cable, and thus on the above traces, it is possible to generate a changing magnetic field that induces errors in *different* traces and chips on the switch PCB through inductive coupling [253]. Similarly, it is possible to generate strong electric fields that induce errors in parallel traces on the switch PCB through capacitive coupling [253]. Both fields are examples of EMI.

Due to the proximity and parallel orientation of traces and circuitry related to different switch ports on the switch PCB, it is common for EMI generated from one port to cause link resets on other ports. This is due to the EMI directly causing glitches in other PHYs, or causing noise on traces between other PHYs and their respective switch ports.

Of course, it is not possible to cause surges on wires while they are being used for communication. Instead, the attacker has two options. First, they can cause surges on *unused* wires in the cable. For example, a Cat 5/6 cable has 4 twisted pairs, but only two are used for 100BASE-TX communication. Second, the attacker can alternate between using the same pairs for inducing link resets and sending BE frames.

We note that, in addition to causing link resets in *other* ports, PCSPOOF causes link resets in the port connected to the attacker. This is fine; as long as the attacker's link recovers before the outgoing port, meaning the outgoing port could wake up while a frame is in flight, PCF injection is possible. In our tests, this happened about half the time.

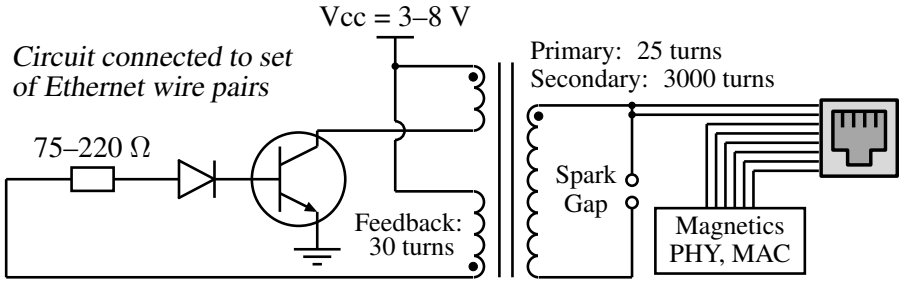

Figure 4.7: **Noise Generation Circuit** — Simple transformer driver circuit that, when placed inside a BE device, generates EMI inside a switch.

4.4.2.4 Avoid Killing the Switch

The challenge of using high voltage to induce link resets is that we must be careful not to kill the switch or PHY connected to the attacker. Doing so would close our attack vector and result in the network continuing to operate normally over the redundant planes.

To accomplish this, PCSPOOF takes advantage of the fact that IEEE 802.3 requires galvanic isolation to protect the PHY from large voltage surges on the Ethernet cable, such as from lightning [239]. In twisted-pair Ethernet, this isolation is performed using small transformers, which in TTE switches are packaged in magnetics chips on the switch PCB [254] or on a connected daughter card [242].

Figure 4.6 shows the design of an Ethernet transformer. A different transformer is connected to each twisted pair in the Ethernet cable. In normal operation, opposite voltages are applied to each wire in a pair, causing current to flow through the primary winding of the transformer. As the current changes, it creates a changing magnetic field, inducing a voltage across the secondary that is seen by the PHY.

This means that, if PCSPOOF caused high-voltage surges on only one wire in a twisted pair, high voltages would be induced in the PHY and kill it. To avoid this, PCSPOOF generates *common mode* surges [255] in which the same voltage is applied to *both* wires in a pair. In this case, a "center tap" in the primary winding allows the current to return to ground. Since current flows in opposite directions from each wire towards the tap, the magnetic fields cancel and high voltage is not directly induced in the PHY.

4.4.2.5 Example Attack Circuit

Figure 4.7 shows a simple circuit capable of generating the common mode surges needed for PCSPOOF. *Disclaimer: The high voltages produced by this circuit can be extremely dangerous.*

The circuit briefly works as follows. When power is applied, current flows to the base of the

transistor, turning it on and letting current flow through the primary winding of the transformer. As current in the primary rises, the magnetic field increases in the transformer core, decreasing current in the feedback winding and turning the transistor back off.

When this happens, current abruptly stops flowing through the primary, generating a high-voltage spike known as an *inductive kick*, which we measured as around 100 V.

The kick induces a voltage across the secondary, causing current to rush into the Ethernet cable. Since the secondary has so many turns, this voltage is very large (10–20 kV). Eventually, the voltage gets so large that current arcs across the spark gap, and the secondary voltage drops back to zero.

Meanwhile, the power supply turns the transistor back on, and the process repeats (i.e., the circuit oscillates on its own). This means that, as long the circuit is powered, it will generate EMI in the switch that can cause link resets.

The design of the circuit makes it easy to hide inside another device. Even with large through-hole parts, the whole circuit fits on a 2.5 cm $\times$ 2.5 cm PCB. There exist suitable transformers (the largest part) that are just 2.5 cm $\times$ 1.25 cm [256] and look like those in typical embedded computers and power supplies. Similarly, glass-enclosed spark gaps that look like small light emitting diodes are available [257]. Finally, we note that the circuit oscillates so fast that, with small gap sizes, it makes almost no audible noise.

4.5 Implementation

To evaluate our attack, we implemented PCSPOOF and executed it on a real TTE testbed used to verify real-life avionic systems. The testbed was designed to mimic a typical fault-tolerant network in a crewed spacecraft or aircraft [59, 11, 52]. Four switches acted as compression masters, and four end systems acted as sync masters. Also, a fifth end system acted as a sync client [32]. The end systems communicated over two redundant planes, each containing half of the switches.

The end systems were implemented on a Dell PowerEdge T620 running CentOS 7.9 with kernel 3.10.0-1160.11. We used TTTech A664 Lab NICs — lab versions of real TTE NICs used in flight. Due to limited hardware availability, we also used older TTTech 1G NICs for some network-level experiments. However, in these cases, the older and newer NICs were verified to behave the same.

For switches, we used modern TTTech OBC HiRel switches [258], as well as older TTTech Development 1G switches in cases where behavior differences were not relevant. We selected the OBC HiRel switch because it is an engineering development unit of a real radiation-hardened spaceflight switch. It also uses TTTech's Space ASIC, which is currently being used in real space vehicles [52, 259].

For scheduling the TTE network, we used TTTech's TTE Tools v5.4 [241], which were the most

up-to-date scheduling tools for our hardware, and the same tools used in real systems. We stress that we used the same configurations and settings as are used for real spacecraft avionic systems.

We created an attack device that connected to one TTE switch in one plane. The device used a Raspberry Pi (RPi) 4B with Ubuntu 20.04 LTS and kernel 5.4.0-1041 to run the PCSPOOF code. The device used a high-voltage circuit, based on Figure 4.7, to induce link resets and enable PCF injections. The RPi communicated using 100BASE-TX, and the two unused cable pairs carried the high voltage signal.

We used SF/FTP shielded Cat 6A cables for all the connections between the various devices. The connection from the attack device to the switch consisted of a 10 m cable, an Ethernet coupler, and a 3 m cable.

4.6 Evaluation

As we showed in §4.4.1, an attacker can reasonably determine how to craft a legitimate PCF in a matter of hours, even with modest embedded hardware and limited network bandwidth. Therefore, we focused our evaluation on assessing the *likelihood* and *impact* of successful PCF injections.

Specifically, we conducted a series of experiments on our testbed to answer four key questions: (1) What is the probability of successfully injecting a PCF? (2) How much does a PCF injection disrupt synchronization between TTE devices? (3) How much does a PCF injection disrupt the delivery of TT messages? and (4) How much damage do PCF injections cause in a real spaceflight application?

4.6.1 Probability

Experimental setup. To determine the probability of successfully injecting a PCF, we needed to answer two questions. Question 1: How often is an attacker able to inject a PCF — i.e., transform a BE frame into a PCF that gets forwarded by the switch connected to the attack device? Question 2: Given a PCF is injected, how often does a *downstream* TTE device, which receives the PCF from the switch, accept that PCF? Moreover, we wanted to determine how the network settings (e.g., transmission rate, background traffic load, drop rate) impacted the answers to these questions.

To answer Question 1, we varied the distance between the switch port connected to the attack device (attack port) and the nearest switch port connected to a TTE device (target port) from 4 ports away (14 cm) to 7 ports away (21 cm), where 7 ports is the maximum separation between two ports on the OBC HiRel switch. For each distance, we continuously sent a 1500-byte BE frame containing a PCF from the attack device to the switch for 5 minutes. To induce link resets in the target port, we enabled/disabled the high-voltage circuit approximately every 1.5–2 s. We used

a Fluke OptiView XG analyzer to capture frames forwarded out of the target port and identify the injected PCF.

We repeated the above experiments in four different setups. In the first, there was no background traffic, and we varied the attacker's transmission rate from {25, 50, 100} Mbps, where 100 Mbps is the maximum rate of the network. In the second, we configured background BE traffic flowing through the switch and out the target port to consume all but {20, 50, 80} Mbps of the bandwidth, and the attacker to send at the maximum rate. This range was chosen to span the network's total bandwidth, while reflecting the fact that real systems leave bandwidth margin for performance reasons and to enable future expansion [260]. In the third, we repeated the second setup except with background TT traffic instead of BE traffic. In the fourth, we configured background TT traffic to flow at 20 Mbps, background BE traffic to flow at 70 Mbps, the attacker to send at the maximum rate, and the network to drop {0.01, 0.1, 1}% of all frames. The fourth setup is a realistic representation of real systems, where TT traffic is commonly limited to 10–20 Mbps [71, 261], and a bandwidth margin of at least 10% is typical [262].

To answer Question 2, we varied the integration cycle of the network from {0.5, 1, 2, 4, 8} ms, where a smaller integration cycle causes tighter synchronization. This range reflects values used in real avionic systems [263, 264, 265]. For each cycle, we used a hardware tap to inject 1000 PCFs on the link between two switches, and recorded the number of injections that caused at least one end system to lose synchronization. We then repeated the process for a link between a switch and end system.

We performed the above experiment in four different setups representing the most extreme configurations from Question 1: (1) no background traffic, (2) 80 Mbps of background BE traffic, (3) 80 Mbps of background TT traffic, and (4) 20 Mbps of background TT traffic, 70 Mbps of background BE traffic, and a 1% drop rate. We report results for coldstart acknowledgement (CA) PCFs only, as the results for integration (IN) PCFs are nearly identical.

Results. Figure 4.8 shows our results for Question 1. As the figure shows, PCSPOOF can inject PCFs in *a matter of seconds*. This is true even in configurations with heavy background traffic. For example, in a realistic configuration with 90% of the bandwidth consumed by TT and BE traffic, and with an extremely high 1% drop rate [266], we observed 20 injections in 5 minutes (roughly one every 15 seconds). The injection rate decreases as the amount of background TT traffic increases, since the attacker is limited to a smaller portion of the bandwidth, resulting in more of the attacker's frames being dropped. However, even in the unlikely case of TT traffic consuming 80% of the bandwidth [71], we still observed more than one injection per minute on average. The figure also shows that PCF injections are possible when the attack and target ports are far apart from each other.

Tables 4.2 and 4.3 show our results for Question 2. With a 1 ms integration cycle, which is

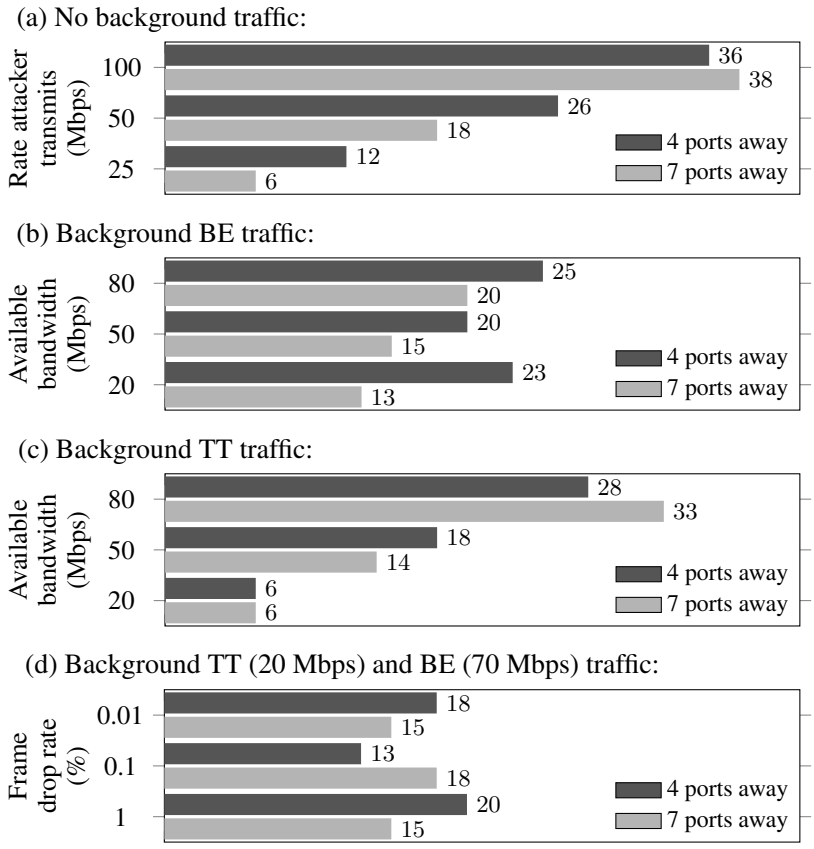

Figure 4.8: **PCF Injections** — Number of PCF injections in 5 minutes under different conditions (e.g., background traffic, drop rate) and with different distances between attack and target ports.

common in practice [265], switches accepted 3.9% of injected PCFs under realistic traffic loads and a 1% drop rate. When combined with the results from Figure 4.8, this means PCSPOOF is likely to inject a PCF *and* get it accepted by a switch *in 6–7 minutes*. Even with a larger 4 ms integration cycle, PCSPOOF is likely to inject a PCF and get it accepted by a switch in 30–40 minutes. Unlike switches, end systems accept all PCFs they receive (i.e., all injections succeed). Thus, PCSPOOF is likely to inject a PCF and get it accepted by an end system *in tens of seconds*, regardless of the integration cycle.

4.6.2 Interrupts

Experimental setup. TTE uses synchronized periodic interrupts to coordinate the execution of software on the end systems [11]. Often, two interrupts are used, a *major* interrupt and a *minor* interrupt, where the major interrupt period divides evenly by the minor interrupt period [105, 87, 95].

Integration Cycle	Background Traffic			
	None	BE	TT	TT+BE (1% drop)
8 ms	0.4%	0.3%	0.3%	0.5%
4 ms	0.7%	0.7%	0.5%	0.8%
2 ms	2.0%	1.7%	1.8%	1.6%
1 ms	4.4%	4.2%	3.3%	3.9%
500 μs	8.0%	7.1%	7.2%	6.3%

Table 4.2: **Successful Injections (Switches)** — Percentage of injected PCFs that were accepted on links to switches.

Integration Cycle	Background Traffic			
	None	BE	TT	TT+BE (1% drop)
8 ms	100%	100%	100%	99.0%
4 ms	100%	100%	100%	99.1%
2 ms	100%	100%	100%	99.0%
1 ms	100%	100%	100%	99.1%
500 μs	100%	100%	100%	98.8%

Table 4.3: **Successful Injections (End Systems)** — Percentage of injected PCFs that were accepted on links to end systems.

By interfering with these interrupts, an attacker could cause significant problems that systems are not designed to handle, such as end systems performing computations on old information, sending data when it is not expected, or failing to generate outputs when needed [28, 267, 268].

To determine how PCSPOOF affects these interrupts, we configured our testbed with a 1 s major interrupt, 25 ms minor interrupt, and 4 ms integration cycle. These values are commonly used in real systems [104, 87], and match those in our case study of a real spaceflight mission (§4.6.4). We note that 4 ms is the smallest integration cycle allowed with our interrupt configuration [241], and minimizes the time it takes the network to recover from PCF injections.

We used a hardware tap to perform 250 successful PCF injections on both inter-switch links and links between switches and end systems. We repeated this process for both CA and IN PCFs, and report the time between the interrupts immediately before and after each injection. We also report interrupt timing for a 5 minute control case, in which no PCFs were injected.

Results. Figure 4.9 shows our results; a single PCF injection can significantly disrupt interrupt timing. For example, a single PCF injection on an end system link can delay the major interrupt by *more than a second* and the minor interrupt by *more than 25 ms*. An inter-switch link injection can delay the minor interrupt by *more than 40 ms*.

In addition, PCF injections on inter-switch links cause interrupt delays *on multiple end systems at once*. This happens because the injected PCF is forwarded to multiple end systems. Importantly, this means that N-modular redundancy [269], a fault tolerance technique where the same function

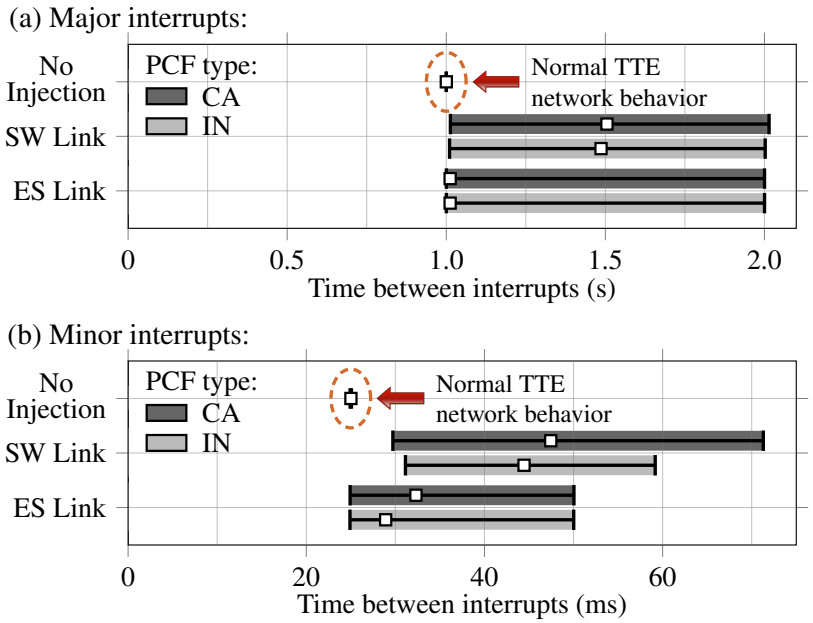

Figure 4.9: **Impact on Interrupts** — Average-Max-Min charts showing the time between interrupts following successful PCF injections on links to switches (SWs) and end systems (ESs).

is performed on multiple end systems, cannot protect systems from PCSPOOF. A single injection could simply delay the interrupts on *all* redundant end systems simultaneously.

In safety-critical systems, where N-modular redundancy is widely trusted for important functions [61, 60, 71], these delays could be disastrous. For example, in automobiles, steering outages exceeding 50 ms can be non-recoverable [29]. Similarly, aircraft can require inputs as often as every 40 ms to avoid failures [26]. Moreover, as we show in §4.6.4, even in cases where a single widespread outage (such as from an inter-switch injection) does not cause system failure, repeated isolated outages (such as from end system injections) can cause redundant systems to fail.

Moreover, we observed that, in the worst case, a single PCF injection can disrupt the interrupt timing of *all end systems in the network*. This is because, when used in a fault-tolerant configuration, TTE requires 3 sync masters to be operational for synchronization between *any end systems* to be possible [241]. Thus, if enough end systems lose synchronization, all end systems do — even ones that never receive the injected PCF.

4.6.3 Messaging

Experimental setup. To determine the effect PCSPOOF has on TT messaging, we repeated the experiment from §4.6.2, with one end system configured as the *sender*, and the others as *receivers*. The sender continuously wrote messages with 100-byte payloads to its NIC, representing typical

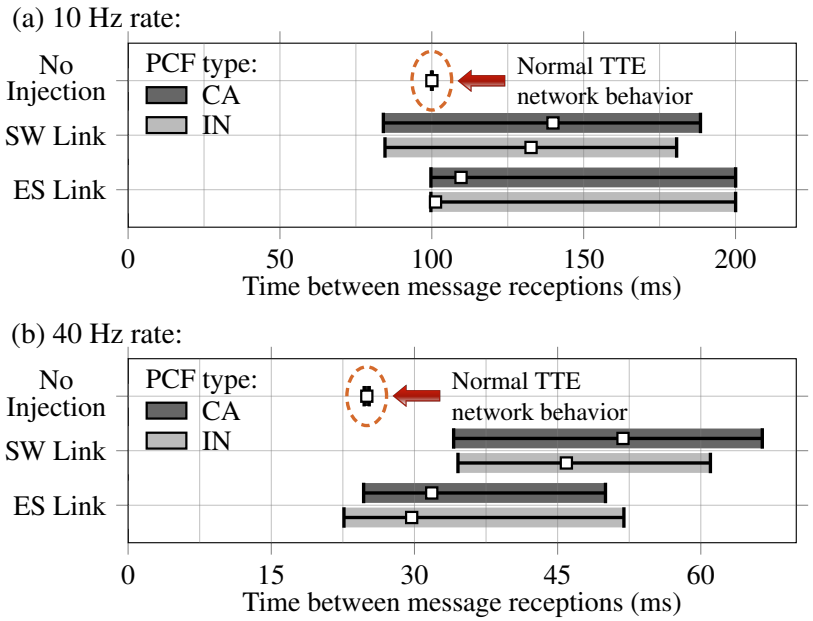

Figure 4.10: **Impact on Messages** — Average-Max-Min charts showing the message delays following successful PCF injections on links to switches (SWs) and the sender end system (ES).

traffic in embedded systems [57]. The messages were stored in a queuing buffer of default size for our hardware [241]. The network was scheduled to continuously transmit the oldest message in the queue to the receivers at a rate from $\{5, 40\}$ Hz, representing the minimum and maximum data rates typically found in real systems [104, 28].

For each rate, we used a hardware tap to perform 250 successful PCF injections on both inter-switch links and links between switches and end systems. We repeated the process for both CA and IN PCFs, and report the time between when an end system last received a message before each injection and next received a message after the injection. We also report the number of message drops caused by each injection — i.e., the number of times the sender successfully stored a message in its NIC, but the message was not received. Finally, we report results for a 5 minute control case with no PCF injections.

Results. Figure 4.10 shows our results. As expected, by disrupting synchronization, PCF injections can cause large message delays. For example, a single PCF injection caused a message expected every 25 ms to not arrive for *up to 65 ms*. As discussed in §4.6.2, such delays can be disastrous for critical applications like steering and engine control, where delays beyond 40–50 ms can be nonrecoverable [29, 26].

Table 4.4 shows the number of message drops caused by each PCF injection. In summary, each successful injection resulted in approximately 20 message drops in a row for all receivers. Thus, successful PCF injections do not only result in message delays but also cause TT messages *to be*

Data Rate:	10 Hz				40 Hz			
PCF Type:	CA		IN		CA		IN	
Link Type:	SW	ES	SW	ES	SW	ES	SW	ES
Min Drops:	20	20	20	20	20	19	20	20
Max Drops:	21	20	21	20	21	20	21	20

Table 4.4: **Message Drops** — Message drops following successful PCF injections on links to switches and the sender end system.

permanently lost. Interestingly, we observed a similar number of drops regardless of the rate at which messages were transmitted, seemingly due to messages being purged from NIC and switch buffers when synchronization is disrupted.

We stress that PCF injections caused these message drops even though TT traffic *travels over multiple planes simultaneously*, and PCFs were only injected *on a single plane*. Therefore, redundant communication paths are not an effective way of mitigating PCSPOOF. The reason is that, since PCSPOOF disrupts the synchronization protocol, it disrupts communication on all planes simultaneously.

4.6.4 Case Study: NASA Asteroid Redirect Mission

To determine how much damage PCSPOOF causes in a real spaceflight application, we conducted a case study based on NASA's planned Asteroid Redirect Mission [270], in which a robotic spacecraft would move an asteroid into a stable orbit around the Moon. A crewed spacecraft, such as NASA's Orion, would then carry astronauts to the asteroid in order to study it, take samples, and return the samples to Earth.

We executed a subset of the mission on a real avionics testbed, during which a representative Orion capsule approached and attempted to dock with the robotic spacecraft. The Orion guidance software, which included several genuine Orion flight software components (e.g., for optical navigation) [95], ran against NASA's Trick Simulation [271], which modeled the vehicles in space, as well as Orion's sensors and actuators. The Orion subsystems and simulation communicated over a fault-tolerant TTE network, similar to Figure 4.1. We used network settings from the real mission.

We executed the mission twice. In the first trial, no PCFs were injected. In the second trial, we executed the full end-to-end attack, including finding the critical traffic marker and injecting PCFs with the attack circuit. After determining the rate at which PCFs were injected on links to the flight computers (roughly every 16 seconds), we switched to injecting PCFs on those links with a hardware tap. This let us assess the impact of PCSPOOF over a long mission (hours), without risk of damaging the switch.

Our results are shown in Figure 4.11. As expected, in the absence of PCF injections, the mission

Figure 4.11: **Asteroid Redirect Mission** — NASA spaceflight simulation without (left) and with (right) PCSPOOF. PCSPOOF caused a significant deviation in the vehicle's flight path, which prevented docking.

completed successfully. Orion approached the robotic spacecraft at a relative velocity of 2–3 m/s until it was approximately 300 m away, aligned itself with the robotic spacecraft, and proceeded straight at 0.1–0.5 m/s until docking was complete.

In contrast, PCSPOOF caused message drops and delays that caused Orion to deviate from its intended flight path. Rather than aligning with the robotic spacecraft, Orion swung underneath it at a distance of approximately 115 m, missed the docking opportunity, and floated away at a rate of 1–2 m/s. These results show that PCSPOOF can significantly disrupt the operation of critical systems that rely on TTE. PCSPOOF also threatens safety, as the uncontrolled maneuvers we observed could easily cause collisions with other objects or vehicles.

4.7 Mitigations

In this section, we discuss potential mitigations to PCSPOOF. In general, mitigations fit into four major categories.

4.7.1 Prevent Attackers from Crafting PCFs

Hide the critical traffic marker. In PCSPOOF, attackers determine the critical traffic marker used in PCFs by observing which BE frames are dropped by the switches (see §4.4.1.2). Designers can prevent attackers from finding the critical traffic marker by configuring switches to drop additional BE traffic that does not contain the critical traffic marker. However, this causes switches to deviate from the IEEE 802.1D standard [272], prevents BE devices whose MAC addresses overlap with the blacklisted addresses from receiving messages, and causes certain BE multicast addresses to become unusable.

Another option is to increase how long it takes to deduce the critical traffic marker, e.g., with switch port rate limiting. However, since the critical traffic marker typically does not change over a system's lifetime [173], this method cannot *prevent* attacks, only delay them. Regularly changing the critical traffic marker would improve security, but could increase costs due to repeated verification and validation (see §4.4.1.2).

Hide virtual link ID assignments. Designers can also prevent attackers from creating PCFs by hiding the virtual link IDs switches use for transmitting PCFs. In existing schedulers, virtual link IDs for PCFs are assigned in reverse order from publicly-available values, making them easy for attackers to predict (see §4.4.1.3). Assigning these values using less predictable methods (e.g., randomly), or regularly changing them, would improve security. However, like in the case of the CT Marker, neither mitigation is likely sufficient for highly-critical systems.

Check the source MAC address. Since TTE devices use the virtual link ID to identify the source of incoming PCFs, they do not check a PCF's source MAC address field (see §4.4.1.1). This means attackers do not need to determine the MAC address of a real compression master in the network in order to craft authentic-looking PCFs. TTE devices could improve security by checking the source MAC address in received PCFs against known correct values. However, doing so would require changes to the TTE hardware.

4.7.2 Prevent Attackers from Injecting PCFs

Block EMI. PCSPOOF enables PCF injections by conducting EMI into the switch over an Ethernet cable. This interference can be prevented by using optocouplers or surge protector devices between the Ethernet cables and TTE switch ports. However, such devices often suffer from performance limitations [273], decrease system reliability by introducing new points of failure, and can increase size, weight, and power due to the inclusion of new hardware [274].

Another option is to use fiber-optic cables, which are incapable of conducting EMI into the switch. However, such cables have several downsides compared to copper, including higher cost, worse durability, and decreased compatibility with commercial hardware [252]. For these reasons, most TTE systems use copper physical layers [133, 275, 276, 277].

Compression master placement. PCSPOOF requires injected PCFs to look like they came from real compression masters (CMs). By carefully placing the CMs, designers can ensure injected PCFs will not be used. For example, if CMs and BE devices are placed on opposite sides of the network, injected PCFs will come from paths with no CMs, and thus be ignored. However, this separation may not be possible in networks with few switches, and can increase size, weight, and power by requiring long cable runs between where BE devices are needed and allowed to connect to the network.

Limit the BE traffic rate. Our results in §4.6.1 show that lowering the rate BE devices are allowed to send messages, e.g., with switch port rate limiting, can decrease the probability of successful PCF injections. However, many TTE switches do not have this feature [247, 254]. Also, as we have shown, successful PCF injections are still highly probable, even when the BE traffic rate is low. In our evaluation, an attacker transmitting at 25 Mbps could still inject more than 1 PCF per minute. Moreover, BE rate limits can disrupt BE data flows that require high data rates, like high-definition video [278].

Disable dangerous state transitions. PCSPOOF exploits the fact that end systems that receive a coldstart acknowledgement PCF will temporarily lose synchronization (see §4.2.2). Thus, one way to combat PCSPOOF is to disable this state transition in the configurations loaded on the end systems [32]. Unfortunately, removing this transition also impacts the ability for the network to detect cliques at system startup [32]. It also does nothing to prevent an attacker from injecting integration PCFs, which can also disrupt synchronization (see §4.2.2).

4.7.3 Prevent Devices from Accepting Injected PCFs

Use link-layer security. One way to mitigate PCSPOOF is to use a link-layer authentication protocol, like IEEE 802.1AE [279]. Unless the attacker knows the cryptographic key used in the network, they cannot inject PCFs that are accepted by TTE devices. Unfortunately, link-layer security is not implemented in existing TTE devices. Adding authentication would require updates to the TTE hardware, as well as impact compatibility with existing TTE systems.

Check the preamble length. PCSPOOF cuts the headers off frames in flight, causing receivers to get injected PCFs with potentially very long preambles (see §4.4.2). TTE devices would be less likely to accept these PCFs if they rejected frames with long preambles. This would force attackers to send smaller BE frames, making the link less likely to recover in the region required for successful injections. Modern TTE devices do not limit the preamble length (see Table 4.1), and doing so would require updates to the device hardware.

Decrease the resynchronization rate. Our results in §4.6.1 show that decreasing the TTE resynchronization rate (i.e., increasing the integration cycle) decreases the probability of PCSPOOF succeeding on inter-switch links. Unfortunately, decreasing this rate also increases timing skew between synchronized devices, and reduces the usable network bandwidth for TT traffic [241, 280]. Moreover, since end systems accept all injected PCFs regardless of the resynchronization rate, decreasing the rate has no effect on the probability of successful injections on links between switches and end systems.

4.7.4 Minimize the Impact of Accepting Injected PCFs

Use more sync masters. If PCSPOOF disrupts enough sync masters, it causes the whole network to lose synchronization, regardless of whether all devices received an injected PCF or not (see §4.6.2). Increasing the number of sync masters can reduce the probability of this happening, but may not be possible in small systems like automobiles. Also, even if a network has many sync masters, care must be taken in choosing their locations in the network. Otherwise, a single injected PCF could still take out all the sync masters.

Decrease periods. PCSPOOF causes larger timing disruptions to interrupts and traffic flows with larger periods (see §4.6). Therefore, PCSPOOF can be somewhat mitigated by using smaller periods than needed. For example, messages needed every 20 ms could be scheduled every 5–10 ms instead. However, this approach wastes network bandwidth, increases processing demands (e.g., CPU, memory), and makes the network more difficult to schedule [241]. Also, this approach does not work for interrupts that require large periods, like the 1 second major interrupt used in avionic systems to trigger the start of the flight software schedule [104, 87].

4.8 Related Work

Attacks on TTE's isolation guarantees. Because of its use in critical applications, much effort has been spent trying to break TTE's isolation guarantees [101, 281, 282, 227]. For example, the Aviation Cyber Security Study [281] analyzed the ability of BE devices to interfere with TT traffic via denial-of-service or MAC flooding attacks. To our knowledge, no successful attacks have ever been reported. TTE's security has also been studied in much less restricted threat models, such as when an attacker controls critical TTE devices [283, 284], or has physical access to the system and can thus unplug cables and intercept messages [229, 285]. In contrast, PCSPOOF does not require the attacker to have physical access to the system, and it can succeed from a single BE device connected to a single network plane.

Ethernet packet-in-packet attacks. Several packet-in-packet attacks have been developed [225, 286, 248, 249]. Recently, EtherOops [248] showed that runaway preamble attacks on wired Ethernet were possible by exploiting data corruptions, like those caused by faulty cables. However, as shown in §4.4.2.2, this attack can easily be prevented in modern TTE networks. Other researchers showed that Ethernet packet-in-packet attacks could be accomplished by exploiting link resets, like those caused by unplugging and replugging cables [249]. However, the attacker had no ability to cause these link resets to occur. PCSPOOF also uses link resets to let an attacker inject malicious PCFs into the network. However, importantly, PCSPOOF lets the attacker induce those link resets at will from a networked BE device.

EMI attacks on Ethernet. Several studies have explored methods for inducing errors in Ethernet networks by exposing switches and cables to EMI [248, 287, 288, 289, 290]. For example, [288] studied the susceptibility of office networks to nearby electromagnetic pulse devices. However, such attacks are mostly effective only on networks with unshielded cables, require the attacker to be in close proximity to the network (e.g., a few meters), and require large antennas to radiate the EMI [248, 288]. In contrast, PCSPOOF induces switch errors by conducting EMI from a networked device, *through* an Ethernet cable, and *into* the switch. This means PCSPOOF works on networks with shielded cables, works from any distance — provided the attack device connects to a switch — and takes up little physical space (see §4.4.2.3).

Timing attacks on real-time systems. There exist several timing-based attacks on TT and other real-time systems [291, 292, 293, 294, 295, 296, 297, 298]. Generally, these attacks work by (1) using side channels to infer the task or communication schedule [291, 292, 293, 294, 295, 296], then (2) interfering with critical tasks or traffic by consuming shared resources when they are needed [297, 298]. PCSPOOF is orthogonal to these methods. In particular, PCSPOOF does not rely on knowledge of timing to be successful. Also, while designers can mitigate interference attacks by planning for the worst-case contention that critical TT tasks and messages may experience [297, 298], PCSPOOF can disrupt all TT tasks and messages regardless of temporal overprovisioning.

Destructive high-voltage circuits. Several devices use high voltage to damage electronic equipment [299, 300, 301]. For example, the venerable EtherKiller [300] destroys Ethernet switches by shorting a switch port to a wall socket. USBKill [299] devices destroy computers by discharging high-voltage capacitors into the computer's USB port. Unlike these devices, PCSPOOF is not intended to damage a TTE switch; doing so would prevent successful attacks. Rather, it is designed to induce controlled errors in the switch that cause link drops on other ports. Also, PCSPOOF is designed to introduce these errors from a fully-functioning Ethernet device.

4.9 Conclusion

TTE is a popular choice for mixed-criticality systems because of its ability to share the network between critical TT and non-critical BE devices. However, this design requires that the critical TT services be completely isolated from the BE devices. We presented PCSPOOF, the first attack capable of breaking TTE's isolation guarantees. Our results show that PCSPOOF threatens the safety of critical TTE systems, like spacecraft and aircraft. We hope the description of our attack, as well as the mitigations we identified, will influence the deployment of current TTE systems, as well as the designs of future mixed-criticality network technologies.

CHAPTER 5

Conclusion and Future Work

5.1 Conclusion

Modern embedded systems, like those in spacecraft and aircraft, can be large distributed systems with hundreds of nodes and thousands of traffic flows. Meeting requirements in these systems is a significant design challenge. For example, these systems are often required to operate in the presence of faults, which typically requires redundant processing nodes to coordinate by exchanging messages. Unfortunately, this need for coordination means traditional fault tolerance protocols are often unable to meet real-time requirements in modern systems, where nodes are often not co-located and the network is heavily loaded. Another challenge is security. Modern embedded systems feature a mix of critical and non-critical devices, with critical devices undergoing a stringent verification and review process, and non-critical devices purchased off-the-shelf and undergoing little review. All these devices often share the same network to reduce size, weight, power, and costs, providing a theoretical means by which non-critical devices could disrupt critical systems. It is the responsibility of the network to ensure such disruption cannot happen, isolating the critical systems from a potentially malicious non-critical device.

In Chapter 2, I described IGOR, a novel Byzantine fault-tolerant (BFT) state machine replication (SMR) protocol that reduces the worst-case latency of real-time embedded systems by letting replicas parallelize coordination and execution. Unlike traditional BFT SMR protocols, the replicas are not required to agree on one set of sensor data before executing. Instead, they eagerly execute on data from multiple redundant sensors, some of which may be arbitrarily faulty, and reach agreement on which execution will become persistent in the background. IGOR's approach allows it to meet deadlines that are not practical for other state-of-the-art protocols.

In Chapter 3, I described CROSSTALK. CROSSTALK shows that by making a few assumptions about the network topology, it is possible to reduce the worst-case latency of BFT SMR in real-time systems without requiring the replicas to have multiple cores. Specifically, CROSSTALK takes advantage of the prevalence of redundant switched networks in practice, routing messages between

the planes to ensure that replicas maintain identical states without needing the replicas to run any coordination protocol. As a result, CROSSTALK can achieve similar or better worst-case latency than IGOR while also having improved schedulability.

In Chapter 4, I described PCSPOOF, a new attack on mixed-criticality Time-Triggered Ethernet (TTE) networks used in critical applications, and the first attack capable of breaking TTE's isolation guarantees. PCSPOOF allows a malicious non-critical device to craft fake synchronization messages and inject them into the network, potentially disrupting the operation of some or even all critical devices. In general, PCSPOOF shows that it is possible to compromise the safety of even the most robust mixed-criticality network technologies. I also described multiple defenses designers can use to protect their systems from PCSPOOF.

5.2 Future Work

In this dissertation, I introduced several approaches to make distributed fault-tolerant embedded systems more performant and secure. However, multiple challenges still remain. I describe some of these challenges below.

5.2.1 Probabilistic BFT SMR with Eager Execution

The IGOR protocol in Chapter 2 is *error-free* [47], meaning it is guaranteed to produce correct results as long as the fault model is valid (i.e., $f < n/3$ replicas are faulty). IGOR was designed to be error-free intentionally, since error-free protocols are required in a variety of safety-critical applications, such as commercial aerospace [122, 123]. It is well known that, without making any assumptions about the network topology, at least $3f + 1$ replicas are needed to solve BFT SMR in the error-free case [46]. IGOR exploits the fact there are at least $3f + 1$ replicas to ensure at least $f + 1$ replicas are non-faulty and end up with the same correct state, which is enough to out-vote up to f faulty replicas when delivering results to the actuators.

However, in some less critical systems, like automotive systems, it may not be necessary to have perfect correctness. Similarly, it may be too expensive to have $3f + 1$ replicas. In such cases, it is more attractive to use a *probabilistic* [58] BFT protocol, which has some small probability of being incorrect even when the fault model holds, but only requires $2f + 1$ replicas [46],

Trying to apply IGOR's approach to the probabilistic setting is not trivial. For example, with $2f + 1$ replicas, the standard IGOR protocol would only guarantee that a single non-faulty replica ends up with the correct state, which is not enough to out-vote any faulty replicas. From some preliminary investigation, it appears like a solution might be possible by starting the protocol with a new pre-processing stage consisting of two rounds of consistent broadcast [168]. However, it

seems likely that such a design would result in little or no latency savings compared to traditional BFT protocols when tolerating small numbers of faults (e.g., $f = 1$ or 2).

5.2.2 CROSSTALK with Byzantine Switches

The CROSSTALK protocol is Chapter 3 uses an omission fault model for the switches. This model is common in safety-critical applications, like crewed spaceflight, where switches may be implemented as self-checking pairs [71]. The model is powerful because it allows a replica that received a message from a sensor to know the message must have taken a specific path. In CROSSTALK, the path is chosen to traverse a sufficient number of switches with cross-links such that one of them must be non-faulty and ensure the message ends up on a non-faulty plane.

However, in some domains, constraining faulty switches to omission may be too costly. Unfortunately, it is not trivial to adapt CROSSTALK to tolerate Byzantine switches. The reason is that a message would no longer be constrained to specific known paths. For example, a message from a faulty sensor could proceed directly to a replica without first traversing the cross-link switches. Similarly, the message could travel to a Byzantine non-replica node that incorrectly retransmits the message on a different network plane. One method to ensure a message takes a specific path would be to have the switches sign messages in flight. However, the use of signatures has been shown to significantly degrade the performance of embedded hardware [302]. Signatures would also prevent CROSSTALK from being used with commodity network switches.

5.2.3 Optimizing the Network Topology for BFT SMR

As discussed in Chapter 3, traditional BFT SMR protocols are expensive in latency and message complexity because they make no assumptions about the network topology. CROSSTALK shows that, by making a few assumptions about the topology, it is possible to significantly reduce the cost of BFT SMR. However, the topology used in CROSSTALK is still based on the redundant switched topology systems commonly use in practice, which is not explicitly intended to make BFT SMR cheap. Rather, this topology is derived from older redundant bus designs [71] and is meant to strike a balance between cable mass, message latency, and design complexity [187].

This raises the question of whether the cost of BFT SMR could be further reduced by better optimizing the network topology. Researchers have had success using techniques such as binary programming to find topologies that minimize straightforward objectives like cable mass and the number of hops between nodes [187]. However, I am not aware of any work that uses network topology optimization to reduce the cost of executing complex distributed protocols. One challenge is that the ideal choice of topology depends on the steps of the protocol and vice versa. As such, a network/protocol co-design optimization approach might be most fruitful.

5.2.4 Compromising Security in Mixed-Criticality Networks

The PCSPOOF attack in Chapter 4 exploits a flaw in wired Ethernet to allow a malicious end device to inject an arbitrary message into an Ethernet network. The reason such an attack is dangerous in TTE is that, in the TTE synchronization protocol [32], a single message (a compressed protocol control frame) has the power to make critical devices lose synchronization. Other Ethernet-based protocols used in critical applications have a similar deficiency. For example, the recent time-triggered Time-Sensitive Networking (TSN) standards [34] use a variant of IEEE 1588 (PTP) [303] for time synchronization. In IEEE 1588, time is distributed to the switches and end devices from a number of designated time servers. If one of these time messages was spoofed, it could cause parts of the network to lose synchronization and critical messages to be dropped. Synchronization protocols used on top of Avionics Full-Duplex Switched Ethernet networks (e.g., [177]) also appear to be susceptible to message-injection attacks.

Similarly, message-injection attacks could be possible in non-Ethernet-based networks that rely on time synchronization. For example, in SpaceFibre [304], a popular network technology used in spaceflight applications, time-critical traffic is sent at prescheduled times according to a global synchronized clock.[1] The synchronized time is distributed with simple broadcast messages [305]. If these messages were spoofed, it could disrupt synchronization and prevent operation of critical devices. However, unlike in Ethernet (Figure 4.5), it has not yet been shown to be possible to inject malicious messages into a SpaceFibre network.

[1] Contrary to its name, SpaceFibre can be implemented on copper physical layers [304].

BIBLIOGRAPHY

[1] Raphael Some, Richard Doyle, Larry Bergman, William Whitaker, Wesley Powell, Michael Johnson, Montgomery Goforth, and Michael Lowry. Human and Robotic Space Mission Use Cases for High-Performance Spaceflight Computing. August 2013. https://trs.jpl.nasa.gov/handle/2014/44422.

[2] Jean-Bernard Itier. A380 Integrated Modular Avionics — The History, Objectives and Challenges of the Deployment of IMA on A380. In *Proc. ARTIST2 Meeting on Integrated Modular Avionics*, Rome, Italy, November 2007. http://www.artist-embedded.org/docs/Events/2007/IMA/Slides/ARTIST2_IMA_Itier.pdf.

[3] Julia Badger, Don Higbee, Tim Kennedy, Sharada Vitalpur, Miriam Sargusingh, Sarah Shull, Bill Othon, Francis J. Davies, Eric Hurlbert, Neil Townsend, Jeff Mauldin, Emily Nelson, Kornel Nagy, Katy Hurlbert, Jeremy Frank, and Stan Love. Spacecraft Dormancy Autonomy Analysis for a Crewed Martian Mission. Technical Report NASA/TM-2018-219965, National Aeronautics and Space Administration, July 2018.

[4] Paul Eisenstein. Chip Shortages Force Automakers To Slash Production. *Forbes*, January 2021. https://www.forbes.com/wheels/news/chip-shortages-force-automakers-slash-production/.

[5] Dragon's "Radiation-Tolerant" Design. *Aviation Week*, November 2012. https://aviationweek.com/dragons-radiation-tolerant-design.

[6] G. D. Scanlon. SAE AS6675 Use Case Summary. Lockheed Martin, January 2021. https://www.ieee802.org/1/files/public/docs2021/dp-Scanlon-aerospace-use-case-summary-0121-v01.pdf.

[7] Jennifer Walter. 40 Years Ago: Detroit's Brainy New Autos, Now With Microprocessors. *Discover Magazine*, August 2020. https://www.discovermagazine.com/technology/40-years-ago-detroits-brainy-new-autos-now-with-microprocessors.

[8] Remy Melina. International Space Station: By the Numbers, February 2020. https://www.space.com/8876-international-space-station-numbers.html.

[9] Andrew Loveless. Impact of Switch Plane Redundancy on Network Availability. NASA Johnson Space Center, February 2022. https://ntrs.nasa.gov/api/citations/20220003523/downloads/2022-02-25_network-reliability.pdf.

[10] Henning Butz. The Airbus Approach to Open Integrated Modular Avionics (IMA): Technology, Methods, Processes, and Future Roadmap. In *Proc. Workshop on Aircraft System Technologies*, Hamburg, Germany, March 2007.

[11] Andrew Loveless. On Time-Triggered Ethernet in NASA's Lunar Gateway. In *Avionics Architectures Community of Practice*. NASA Johnson Space Center, July 2020. https://ntrs.nasa.gov/api/citations/20205005104/downloads/2020-07-26-AA-CoP.pdf.

[12] R. W. Butler. The SURE Approach to Reliability Analysis. *IEEE Transactions on Reliability*, 41(2), June 1992.

[13] Jacek Krywko. Space-Grade CPUs: How Do You Send More Computing Power Into Space? *Ars Technica*, November 2019. https://arstechnica.com/science/2019/11/space-grade-cpus-how-do-you-send-more-computing-power-into-space/.

[14] Wesley Powell. High-Performance Spaceflight Computing (HPSC) Project Overview. In *Proc. Radiation Hardened Electronics Technology Conference (RHET)*, Phoenix, AZ, USA, November 2018.

[15] Natasa Miskov-Zivanov and Diana Marculescu. Multiple Transient Faults in Combinational and Sequential Circuits: A Systematic Approach. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 29(10), October 2010.

[16] Michelle Rucker and John Connolly. Deep Space Gateway — Enabling Missions to Mars. National Aeronautics and Space Administration, November 2017. https://ntrs.nasa.gov/api/citations/20180000122/downloads/20180000122.pdf.

[17] ACTUV "Sea Hunter" Prototype Transitions to Office of Naval Research for Further Development. DARPA, January 2018, https://www.darpa.mil/news-events/2018-01-30a.

[18] Auntowhan Andrews. Concept of Operations for the Tactical use of Autonomous Unmanned Surface Systems. Technical report, US Naval War College, February 2018.

[19] Philip Koopman and Michael Wagner. Autonomous Vehicle Safety: An Interdisciplinary Challenge. *IEEE Intelligent Transportation Systems Magazine*, 9(1), 2017.

[20] Thomas Ormston. Time Delay Between Mars and Earth. *ESA Blogs*, August 2012. https://blogs.esa.int/mex/2012/08/05/time-delay-between-mars-and-earth/.

[21] Stephen Steffes and Gregg Barton. Deep Space Autonomous Navigation Options for Future Missions. In *Proc. AIAA SPACE and Astronautics Forum and Exposition*, Orlando, FL, USA, September 2017.

[22] Richard Wood, Belle Upadhyaya, and Dan Floyd. An Autonomous Control Framework for Advanced Reactors. *Nuclear Engineering and Technology*, 49(5), August 2017.

[23] Alwyn Scott. Boeing Studies Pilotless Planes as It Ponders Next Jetliner. *Reuters*, June 2017. https://www.reuters.com/article/us-boeing-airshow-autonomous-idUSKBN18Z12M.

[24] Marshall Pease, Robert Shostak, and Leslie Lamport. Reaching Agreement in the Presence of Faults. *Journal of the ACM*, 27(2), April 1980.

[25] Kevin Driscoll, Brendan Hall, and Kevin Schweiker. Application Agreement and Integration Services. Technical Report NASA/CR–2013-217963, National Aeronautics and Space Administration, February 2013.

[26] Jaynarayan Lala and Richard Harper. Architectural Principles for Safety-Critical Real-Time Applications. *Proceedings of the IEEE*, 82(1), January 1994.

[27] Michael Fischer and Nancy Lynch. A Lower Bound for the Time to Assure Interactive Consistency. *Information Processing Letters*, 14(4), 1982.

[28] Andrew Loveless, Ronald Dreslinski, Baris Kasikci, and Linh Thi Xuan Phan. IGOR: Accelerating Byzantine Fault Tolerance for Real-Time Systems with Eager Execution. In *Proc. Real-Time and Embedded Technology and Applications Symposium (RTAS)*, Nashville, TN, USA, May 2021.

[29] Günter Heiner and Thomas Thurner. Time-Triggered Architecture for Safety-Related Distributed Real-Time Systems in Transportation Systems. In *Proc. International Symposium on Fault-Tolerant Computing (FTCS)*, Munich, Germany, June 1998.

[30] Alysson Neves Bessani and Eduardo Alchieri. A Guided Tour on the Theory and Practice of State Machine Replication, November 2014.

[31] ARINC 664 P7: Aircraft Data Network Part 7 — Avionics Full-Duplex Switched Ethernet Network. Aeronautical Radio, Incorporated, January 2009.

[32] SAE AS6802: Time-Triggered Ethernet. SAE International, November 2016.

[33] Steve Parkes, Chris McClements, David McLaren, Albert Ferrer Florit, and Alberto Gonzalez Villafranca. SpaceFibre Networks: SpaceFibre, Long Paper. In *Proc. International SpaceWire Conference*, Yokohama, Japan, October 2016.

[34] IEEE 802.1Qbv-2015: IEEE Standard for Local and Metropolitan Area Networks — Bridges and Bridged Networks — Amendment 25: Enhancements for Scheduled Traffic. Institute of Electrical and Electronics Engineers, May 2016.

[35] Andrew Loveless, Linh Thi Xuan Phan, Ronald Dreslinski, and Baris Kasikci. PCspooF: Compromising the Safety of Time-Triggered Ethernet. In *Proc. Symposium on Security and Privacy (S&P)*, San Francisco, CA, USA, May 2023.

[36] Roman Obermaisser. Properties of Time-Triggered Communication Systems — Fault Containment and Error Containment. In *Time-Triggered Communication*. CRC Press, 2011.

[37] Robert Hodson, Yuan Chen, John Pandolf, Kuok Ling, Kristen Boomer, Christopher Green, Jesse Leitner, Peter Majewicz, Scott Gore, Carlton Faller, Erik Denson, Ronald Hodge, Angela Thoren, and Michael Defrancis. Recommendations on Use of Commercial-Off-The-Shelf (COTS) Electrical, Electronic, and Electromechanical (EEE) Parts for NASA Missions. Technical Report NESC-RP-19-01490, National Aeronautics and Space Administration, December 2020.

[38] Ensuring Successful Implementation of Commercial Items In Air Force Systems. Technical Report SAB-TR-99-03, United States Air Force Scientific Advisory Board, April 2000.

[39] Wilfried Steiner and Bruno Dutertre. The TTEthernet Synchronization Protocols and Their Formal Verification. *International Journal of Critical Computer-Based Systems*, 4(3), December 2013.

[40] Robert Hodson, Yuan Chen, Dwayne Morgan, Marc Butler, Joseph Sdhuh, Jennifer Petelle, David Gwaltney, Lisa Coe, Terry Koelbl, and Hai Nguyen. Heavy Lift Vehicle (HLV) Avionics Flight Computing Architecture Study. Technical Report NASA/TM–2011-217168, National Aeronautics and Space Administration, August 2011.

[41] Jaynarayan Lala, Richard Harper, Kenneth Jaskowiak, Gene Rosch, Linda Alger, and Andrei Schor. Advanced Information Processing System (AIPS)-based Fault Tolerant Avionics Architecture for Launch Vehicles. In *Proc. Digital Avionics Systems Conference (DASC)*, Virginia Beach, VA, USA, January 1990.

[42] Wilfredo Torres-Pomales, Mahyar Malekpour, and Paul Miner. ROBUS-2: A Fault-Tolerant Broadcast Communication System. Technical Report NASA/TM-2005-213540, NASA Langley Research Center, March 2005.

[43] John Bengtson, Victor Donescu, Philip Carne Kjaer, and Kenneth Skaug. Wind Turbine Control System with Decentralized Voting (Patent US9670907B2). Vestas Wind Systems AS, June 2017.

[44] Jaynarayan Lala. A Byzantine Resilient Fault Tolerant Computer for Nuclear Power Plant Applications. In *Proc. International Symposium on Fault-Tolerant Computing Systems (FTCS)*, Vienna, Austria, July 1986.

[45] Dariusz Kowalski and Achour Mostéfaoui. Synchronous Byzantine Agreement with Nearly a Cubic Number of Communication Bits. In *Proc. Symposium on. Principles of Distributed Computing (PODC)*, Montreal, Canada, July 2013.

[46] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems*, 4(3), July 1982.

[47] Guanfeng Liang and Nitin Vaidya. Error-Free Multi-Valued Consensus with Byzantine Failures. In *Proc. Symposium on Principles of Distributed Computing (PODC)*, San Jose, CA, USA, June 2011.

[48] Tyler Lovelly and Alan George. Comparative Analysis of Present and Future Space-Grade Processors with Device Metrics. *Journal of Aerospace Information Systems*, 14(3), March 2017.

[49] Case Study: Vestas — Delivering Safety Control Systems for Wind Turbines. *TTTech Industrial Automation AG*, 2023. https://www.tttech-industrial.com/wp-content/uploads/TTTech-Industrial-Casestudy-Vestas.pdf.

[50] Christian Fuchs. The Evolution of Avionics Networks From ARINC 429 to AFDX. In *Proc. Seminars Future Internet (FI), Innovative Internet Technologies and Mobile Communication (IITM), and Aerospace Networks (AN)*, Munich, Germany, 2013.

[51] Olivier Notebaert and Franck Wartel. Ethernet for Space with TSN (Time Sensitive Networking). In *Proc. ESA Workshop on Avionics, Data, Control and Software Systems (ADCSS)*, Noordwijk, Netherlands, November 2019.

[52] Christian Fidi, Ivan Masar, Jean-Francois Dufour, and Mirko Jakovljevic. Radiation-Tolerant System-On-Chip (SOC) With Deterministic Ethernet Switching for Scalable Modular Launcher Avionics. In *Proc. Embedded Real Time Systems (ERTS) European Congress*, Toulouse, France, January 2018.

[53] D. A. Gwaltney and J. M. Briscoe. Comparison of Communication Architectures for Spacecraft Modular Avionics Systems. Technical Report NASA/TM-2006-214431, National Aeronautics and Space Administration, June 2006.

[54] Glenn Rakow. NASA SpaceWire Architectures: Present and Future. In *Proc. Military and Aerospace Programmable Logic Devices International Conference (MAPLD), Spacewire 101 Seminar*, Washington, DC, USA, September 2006.

[55] Richard Alena, Patrick Collier, Mohammad Ahkter, Soumik Sinharoy, and Deepak Shankar. High Performance Space VPX Payload Computing Architecture Study. In *Proc. Aerospace Conference (AeroConf)*, Big Sky, MT, USA, March 2016.

[56] Wesley Powell. NASA GSFC Avionics Architectures and Future Directions. In *Proc. Space Computing and Connected Enterprise Resiliency Conference (SCCERC)*, Bedford, MA, USA, June 2018.

[57] Jean-Luc Scharbarg and Christian Fraboul. Dimensioning of Civilian Avionics Networks. In *Industrial Communication Technology Handbook*. CRC Press, August 2014.

[58] Edo Roth and Andreas Haeberlen. Do Not Overpay for Fault Tolerance! In *Proc. Real-Time and Embedded Technology and Applications Symposium (RTAS)*, Nashville, TN, USA, May 2021.

[59] Mitch Fletcher. Progression of an Open Architecture: From Orion to Altair and LSS. Technical Report S65-5000-20-0, Honeywell, International, May 2009.

[60] Kenneth Hoyme and Kevin Driscoll. SAFEbus. In *Proc. Digital Avionics Systems Conference (DASC)*, Seattle, WA, USA, October 1992.

[61] Coy Kouba, Deborah Buscher, and Joseph Busa. The X-38 Spacecraft Fault-Tolerant Avionics System. In *Proc. Military and Aerospace Programmable Logic Devices International Conference (MAPLD)*, Washington, DC, USA, August 2003.

[62] Bernd Wolff and Peter Scheffers. "DMS-R, the Brain of the ISS", 10 Years of Continuous Successful Operation in Space. In *Proc. Data Systems in Aerospace Conference (DASIA)*, Drubrovnik, Croatia, August 2012.

[63] Christopher Marchant. Ares I Avionics Introduction. In *Proc. NASA/ARMY Software and Systems Forum*, Huntsville, AL, USA, November 2009.

[64] J. T. Sims. Redundancy Management Software Services for Seawolf Ship Control System. In *Proc. International Symposium on Fault-Tolerant Computing (FTCS)*, Seattle, WA, USA, June 1997.

[65] GE Fanuc Intelligent Platforms Announces TTEthernet for Safety-Critical Avionics Applications. *GE Enterprise Solutions*, April 2009. https://www.ge.com/news/press-releases/ge-fanuc-intelligent-platforms-announces-ttethernet-safety-critical-avionics.

[66] TTTech to Provide ARINC 664 p7 Products for Mission System on UK AW101 Merlin Mk4/4a Helicopters. *TTTech Computertechnik AG*, April 2015. https://www.tttech.com/press/tttech-to-provide-arinc-664-p7-products-for-mission-system-on-uk-aw101-merlin-mk4-4a-helicopters/.

[67] Woodrow Bellamy. TTEthernet Avionics Backbone a Technology Breakthrough for S-97 Raider. *Aviation Today*, July 2015. https://www.aviationtoday.com/2015/07/20/ttethernet-avionics-backbone-a-technology-breakthrough-for-s-97-raider/.

[68] Advanced Wind Turbine Control Architecture: High Availability DCS With Integrated Safety Used by Vestas. *TTTech Computertechnik AG*, 2023. https://www.tttech.com/wp-content/uploads/TTTech-Vestas-Casestudy.pdf.

[69] Wilfried Steiner, Günther Bauer, Brendan Hall, and Michael Paulitsch. Time-Triggered Ethernet — Protocol Services, Startup and Restart, Clique Detection. In *Time-Triggered Communication*. CRC Press, 2011.

[70] Andrew Loveless, Christian Fidi, and Stefan Wernitznigg. A Proposed Byzantine Fault-Tolerant Voting Architecture using Time-Triggered Ethernet. In *Proc. SAE AeroTech Conference*, Fort Worth, TX, USA, September 2017.

[71] Andrew Loveless. Notional 1FT Voting Architecture with Time-Triggered Ethernet. NASA Johnson Space Center, November 2016. https://ntrs.nasa.gov/api/citations/20170001652/downloads/20170001652.pdf.

[72] John Wensley, Leslie Lamport, Jack Goldberg, Milton Green, Karl Levitt, P. M. Melliar-Smith, Robert Shostak, and Charles Weinstock. SIFT: Design and Analysis of a Fault-Tolerant Computer for Aircraft Control. *Proceedings of the IEEE*, 66(10), October 1978.

[73] John Hanaway and Robert Moorehead. Space Shuttle Avionics System. Technical Report NASA SP-504, National Aeronautics and Space Administration, January 1989.

[74] Robert Hammett. Ultra-Reliable Real-Time Control Systems — Future Trends. In *Proc. Digital Avionics Systems Conference (DASC)*, Bellevue, WA, USA, October 1998.

[75] Robert Hammett, Michael Coakley, David Sevigny, and Ron Zamojski. Automatic Performance Monitoring Enhances Seawolf Submarine Ship Control Maintainability. *Naval Engineers Journal*, 110(2), March 1998.

[76] Tara Polsgrove, Jack Chapman, Steve Sutherlin, Brian Taylor, Leo Fabisinski, Tim Collins, Alicia Dwyer Cianciolo, Jamshid Samareh, Ed Robertson, Bill Studak, Sharada Vitalpur, Allan Lee, and Glenn Rakow. Human Mars Lander Design for NASA's Evolvable Mars Campaign. In *Proc. Aerospace Conference (AeroConf)*, Big Sky, MT, USA, March 2016.

[77] Richard Harper and Jaynarayan Lala. Fault-Tolerant Parallel Processor. *Journal of Guidance, Control, and Dynamics*, 14(3), May 1991.

[78] Daniel Stine. Digital Signatures for a Byzantine Resilient Computer System. Master's thesis, Massachusetts Institute of Technology, 1995.

[79] Manos Kapritsos, Yang Wang, Vivien Quema, Allen Clement, Lorenzo Alvisi, and Mike Dahlin. All About Eve: Execute-Verify Replication for Multi-Core Servers. In *Proc. Conference on Operating Systems Design and Implementation (OSDI)*, Hollywood, CA, USA, October 2012.

[80] Amos Albert. Comparison of Event-Triggered and Time-Triggered Concepts with Regard to Distributed Control Systems. *Embedded World*, January 2004.

[81] Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong. Zyzzyva: Speculative Byzantine Fault Tolerance. In *Proc. Symposium on Operating Systems Principles (SOSP)*, Stevenson, Washington, USA, October 2007.

[82] James Hendricks, Shafeeq Sinnamohideen, Gregory Ganger, and Michael Reiter. Zzyzx: Scalable fault Tolerance through Byzantine Locking. In *Proc. International Conference on Dependable Systems and Networks (DSN)*, Chicago, IL, USA, June 2010.

[83] Atul Singh, Pedro Fonseca, Petr Kuznetsov, Rodrigo Rodrigues, and Petros Maniatis. Zeno: Eventually Consistent Byzantine-Fault Tolerance. In *Proc. Symposium on Networked Systems Design and Implementation (NSDI)*, Boston, MA, USA, April 2009.

[84] Sisi Duan, Sean Peisert, and Karl Levitt. hBFT: Speculative Byzantine Fault Tolerance with Minimum Cost. *IEEE Transactions on Dependable and Secure Computing*, 12(1), January 2015.

[85] Pierre-Louis Aublin, Rachid Guerraoui, Nikola Knežević, Vivien Quéma, and Marko Vukolić. The Next 700 BFT Protocols. *ACM Transactions on Computer Systems*, 32(4), January 2015.

[86] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, Srinivasan Muralidharan, Chet Murthy, Binh Nguyen, Manish Sethi, Gari Singh, Keith Smith, Alessandro Sorniotti, Chrysoula Stathakopoulou, Marko Vukolić, Sharon Weed Cocco, and Jason Yellick. Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains. In *Proc. European Conference on Computer Systems (EuroSys)*, Porto, Portugal, April 2018.

[87] David McComas. NASA/GSFC's Flight Software Core Flight System. In *Proc. Flight Software Workshop*, San Antonio, TX, USA, November 2012.

[88] Lorraine Prokop. NASA's Core Flight Software — A Reusable Real-Time Framework. NASA Johnson Space Center, November 2014. https://ntrs.nasa.gov/api/citations/20140017040/downloads/20140017040.pdf.

[89] Jörn Migge, Josetxo Villanueva, Nicolas Navet, and Marc Boyer. Insights on the Performance and Configuration of AVB and TSN in Automotive Ethernet Networks. In *Proc. Embedded Real Time Systems (ERTS) European Congress*, Toulouse, France, January 2018.

[90] International Deep Space Interoperability Standards — Software Standard. National Aeronautics and Space Administration, June 2019. https://www.internationaldeepspacestandards.com/.

[91] Richard Zurawski. *Industrial Communication Technology Handbook*. CRC Press, August 2014.

[92] Amotz Bar-Noy, Danny Dolev, Cynthia Dwork, and H. Raymond Strong. Shifting Gears: Changing Algorithms on the Fly to Expedite Byzantine Agreement. In *Proc. Symposium on Principles of Distributed Computing (PODC)*, Vancouver, Canada, August 1987.

[93] Nicolas Navet, Jan Seyler, and Jörn Migge. Timing Verification of Real-Time Automotive Ethernet Networks: What Can We Expect from Simulation? In *Proc. SAE World Congress & Exhibition*, Detroit, MI, USA, April 2015.

[94] Anaïs Finzi and Silviu Craciunas. Breaking vs. Solving: Analysis and Routing of Real-Time Networks with Cyclic Dependencies using Network Calculus. In *Proc. International Conference on Real-Time Networks and Systems (RTNS)*, Toulouse, France, November 2019.

[95] Andrew Loveless. On TTEthernet for Integrated Fault-Tolerant Spacecraft Networks. In *Proc. AIAA SPACE and Astronautics Forum and Exposition*, Pasadena, CA, USA, August 2015.

[96] Marc Boyer, Hugo Daigmorte, Nicolas Navet, and Jörn Migge. Performance Impact of the Interactions Between Time-Triggered and Rate-Constrained Transmissions in TTEthernet. In *Proc. Embedded Real Time Systems (ERTS) European Congress*, Toulouse, France, January 2016.

[97] Piotr Berman and Juan Garay. Efficient Distributed Consensus with N = (3 + Epsilon)T Processors (Extended Abstract). In *Proc. International Workshop on Distributed Algorithms (WDAG)*, Delphi, Greece, October 1991.

[98] Juan Garay and Yoram Moses. Fully Polynomial Byzantine Agreement for Processors in Rounds. *SIAM Journal on Computing*, 27(1), February 1998.

[99] ISO 17458-1:2013: Road Vehicles — FlexRay Communications System — Part 1: General Information and Use Case Definition. International Organization for Standardization, February 2013.

[100] SAE AS6003: TTP Communication Protocol. SAE International, February 2011.

[101] Allen Starke, D. Kumar, M. Ford, Janise McNair, and Aaron Bell. A Test Bed Study of Network Determinism for Heterogeneous Traffic using Time-Triggered Ethernet. In *Proc. Military Communications Conference (MILCOM)*, Baltimore, MD, USA, October 2017.

[102] Cédric Wilwert, Nicolas Navet, Yeqiong Song, and Françoise Simonot-Lion. Design of Automotive X-by-Wire Systems. In *The Industrial Communication Technology Handbook*. CRC Press, 2005.

[103] David McComas. Core Flight System (cFS) Background and Overview. NASA Goddard Space Flight Center. https://cfs.gsfc.nasa.gov/cFS-OviewBGSlideDeck-ExportControl-Final.pdf.

[104] Loraine Prokop, Robert Hirsh, and Carlos Pagan. Requirements-Based Execution Time Prediction of a Partitioned Real-Time System Using I/O and SLOC Estimates. *Innovations in Systems and Software Engineering*, 8, October 2012.

[105] Core Flight System Scheduler (SCH) Application Design Review. National Aeronautics and Space Administration, October 2019. https://github.com/nasa/SCH.

[106] Jean-Baptiste Chaudron, David Saussié, Pierre Siron, and Martin Adelantado. Real-Time Distributed Simulations in an HLA Framework: Application to Aircraft Simulation. *Simulation*, 90(6), June 2014.

[107] Michaël Lauer, Frédéric Boniol, Claire Pagetti, and Jérôme Ermont. End-to-End Latency and Temporal Consistency Analysis in Networked Real-Time Systems. *International Journal of Critical Computer-Based Systems*, 5(3/4), September 2014.

[108] Vassos Hadzilacos and Sam Toueg. A Modular Approach to Fault-Tolerant Broadcasts and Related Problems. Technical report, Cornell University, May 1994.

[109] Mohammad Hassan Azadmanesh and Roger Kieckhafer. Exploiting Omissive Faults in Synchronous Approximate Agreement. *IEEE Transactions on Computers*, 49(10), October 2000.

[110] Jeremy Brown and Brad Martin. How Fast Is Fast Enough? Choosing Between Xenomai and Linux for Real-Time Applications. In *Proc. Real-Time Linux Workshop*, Nairobi, Kenya, October 2010.

[111] Benjamin Ip. Performance Analysis of VxWorks and RTLinux. Technical report, Columbia University, 2001.

[112] Rainer Makowitz and Christopher Temple. Flexray — A Communication Network for Automotive Control Systems. In *Proc. International Workshop on Factory Communication Systems (WFCS)*, Torino, Italy, June 2006.

[113] Yi-fan Zeng and Gui-jun Chen. Research on Methods to Improve Precise Time Synchronization for IRIG-B Code Encoder. In *Proc. Asia-Pacific Power and Energy Engineering Conference (APPEEC)*, Shanghai, China, March 2012.

[114] Gabriel Bracha and Sam Toueg. Resilient Consensus Protocols. In *Proc. Symposium on Principles of Distributed Computing (PODC)*, Montreal, Canada, August 1983.

[115] David Chi-Shing Chau. Authenticated Messages for a Real-Time Fault-Tolerant Computer System. Master's thesis, Massachusetts Institute of Technology, 2006.

[116] Miguel Castro and Barbara Liskov. Practical Byzantine Fault Tolerance. In *Proc. Symposium on Operating Systems Design and Implementation (OSDI)*, Berkeley, CA, USA, February 1999.

[117] Bhavitavya Bhadviya. Testing Byzantine Faults. Technical report, University of Michigan, 2010.

[118] Pankaj Khanchandani and Christoph Lenzen. Self-Stabilizing Byzantine Clock Synchronization with Optimal Precision. *Theory of Computing Systems*, 63(2), February 2019.

[119] Kevin Driscoll, Brendan Hall, Michael Paulitsch, Phil Zumsteg, and Håkan Sivencrona. The Real Byzantine Generals. In *Proc. Digital Avionics Systems Conference (DASC)*, Salt Lake City, UT, USA, October 2004.

[120] Roman Obermaisser. *Time-Triggered Communication*. CRC Press, 2011.

[121] Data Network Evaluation Criteria Handbook. Technical Report DOT/FAA/AR-09/24, Federal Aviation Administration, June 2009.

[122] Michael Paulitsch, Jennifer Morris, Brendan Hall, Kevin Driscoll, Elizabeth Latronico, and Philip Koopman. Coverage and the Use of Cyclic Redundancy Codes in Ultra-Dependable Systems. In *Proc. International Conference on Dependable Systems and Networks (DSN)*, Yokohama, Japan, June 2005.

[123] Kevin Driscoll, Brendan Hall, Håkan Sivencrona, and Phil Zumsteg. Byzantine Fault Tolerance, From Theory to Reality. In *Proc. International Conference on Computer Safety, Reliability, and Security (SAFECOMP)*, Edinburgh, Scotland, September 2003.

[124] Li Gong, Patrick Lincoln, and John Rushby. Byzantine Agreement with Authentication: Observations and Applications in Tolerating Hybrid and Link Faults. *Dependable Computing and Fault Tolerant Systems*, 10, September 1995.

[125] Russell Turpin and Brian Coan. Extending Binary Byzantine Agreement to Multivalued Byzantine Agreement. *Information Processing Letters*, 18(2), February 1984.

[126] Andrew Loveless, Ronald Dreslinski, and Baris Kasikci. Optimal and Error-Free Multi-Valued Byzantine Consensus Through Parallel Execution. March 2020. https://eprint.iacr.org/2020/322.pdf.

[127] Chaya Ganesh and Arpita Patra. Optimal Extension Protocols for Byzantine Broadcast and Agreement. March 2017. https://eprint.iacr.org/2017/063.pdf.

[128] Arpita Patra. Error-Free Multi-Valued Broadcast and Byzantine Agreement with Optimal Communication Complexity. In *Proc. International Conference on Principles of Distributed Systems (OPODIS)*, Toulouse, France, December 2011.

[129] Piotr Berman, Juan Garay, and Kenneth Perry. Bit Optimal Distributed Consensus. *Computer Science: Research and Applications*, January 1992.

[130] Brian Coan and Jennifer Welch. Modular Construction of a Byzantine Agreement Protocol with Optimal Message Bit Complexity. *Information and Computation*, 97(1), March 1992.

[131] Yang Xiao, Ning Zhang, Jin Li, Wenjing Lou, and Y. Thomas Hou. Distributed Consensus Protocols and Algorithms. In *Blockchain for Distributed Systems Security*. Wiley, 2019.

[132] Cary Spitzer. *The Avionics Handbook*. CRC Press, December 2000.

[133] International Deep Space Interoperability Standards — Avionics Standard. National Aeronautics and Space Administration, June 2019. https://www.internationaldeepspacestandards.com/.

[134] Yoram Moses and Orli Waarts. Coordinated Traversal: (t+1)-round Byzantine Agreement in Polynomial Time. In *Proc. Symposium on Foundations of Computer Science (FOCS)*, White Plains, NY, USA, October 1988.

[135] Juan Garay and Yoram Moses. Fully Polynomial Byzantine Agreement in t + 1 Rounds. In *Proc. Symposium on Theory of Computing (STOC)*, San Diego, CA, USA, May 1993.

[136] Chaya Ganesh and Arpita Patra. Broadcast Extensions with Optimal Communication and Round Complexity. In *Proc. Symposium on Principles of Distributed Computing (PODC)*, Chicago, IL, USA, July 2016.

[137] Kartik Nayak, Ling Ren, Elaine Shi, Nitin H. Vaidya, and Zhuolun Xiang. Improved Extension Protocols for Byzantine Broadcast and Agreement. February 2020. https://arxiv.org/pdf/2002.11321v1.pdf.

[138] Melinda Tang. Wireless Reconfigurability of Fault-Tolerant Processing Systems. Master's thesis, Massachusetts Institute of Technology, September 2008.

[139] Theodore Baker and Alan Shaw. The Cyclic Executive Model and Ada. *Real-Time Systems*, 1(1), June 1989.

[140] Dan R. K. Ports, Jialin Li, Vincent Liu, Naveen Kr. Sharma, and Arvind Krishnamurthy. Designing Distributed Systems Using Approximate Synchrony in Data Center Networks. In *Proc. Conference on Networked Systems Design and Implementation (NSDI)*, Oakland, CA, USA, May 2015.

[141] Parisa Jalili Marandi, Marco Primi, and Fernando Pedone. High Performance State-Machine Replication. In *Proc. International Conference on Dependable Systems and Networks (DSN)*, Hong Kong, June 2011.

[142] Ricardo Jimenez-Peris, Marta Patino-Martinez, Bettina Kemme, and Gustavo Alonso. Improving the Scalability of Fault-Tolerant Database Clusters. In *Proc. International Conference on Distributed Computing Systems (ICDCS)*, Vienna, Austria, July 2002.

[143] Bettina Kemme, Fernando Pedone, Gustavo Alonso, and André Schiper. Processing Transactions Over Optimistic Atomic Broadcast Protocols. In *Proc. International Conference on Distributed Computing Systems (ICDCS)*, Austin, Texas, USA, June 1999.

[144] Fernando Pedone and André Schiper. Optimistic Atomic Broadcast. In *Proc. International Symposium on Distributed Computing (DISC)*, Andros, Greece, September 1998.

[145] Carlos Eduardo Bezerra, Fernando Pedone, Robbert van Renesse, and Claudio Geyer. Providing Scalability and Low Latency in State Machine Replication. Technical Report USI-INF-TR-2015/06, Università della Svizzera Italiana, December 2015.

[146] Odorico M. Mendizabal, Parisa Jalili Marandi, Fernando Luís Dotti, and Fernando Pedone. Checkpointing in Parallel State-Machine Replication. In *Proc. International Conference on Principles of Distributed Systems (OPODIS)*, Cortina d'Ampezzo, Italy, December 2014.

[147] Parisa Jalili Marandi, Carlos Eduardo Bezerra, and Fernando Pedone. Rethinking State-Machine Replication for Parallelism. In *Proc. International Conference on Distributed Computing Systems (ICDCS)*, Madrid, Spain, June 2014.

[148] Ramakrishna Kotla and Mike Dahlin. High Throughput Byzantine Fault Tolerance. In *Proc. International Conference on Dependable Systems and Networks (DSN)*, Florence, Italy, June 2004.

[149] Sachin Hirve, Roberto Palmieri, and Binoy Ravindran. Archie: A Speculative Replicated Transactional System. In *Proc. International Middleware Conference*, Bordeaux, France, December 2014.

[150] Nuno Santos and André Schiper. Achieving High-Throughput State Machine Replication in Multi-Core Systems. In *Proc. International Conference on Distributed Computing Systems (ICDCS)*, Philadelphia, Pennsylvania, USA, July 2013.

[151] Zhenyu Guo, Chuntao Hong, Mao Yang, Dong Zhou, Lidong Zhou, and Li Zhuang. Rex: Replication at the Speed of Multi-Core. In *Proc. European Conference on Computer Systems (EuroSys)*, Amsterdam, Netherlands, April 2014.

[152] Matthias Fitzi and Martin Hirt. Optimally Efficient Multi-Valued Byzantine Agreement. In *Proc. Symposium on Principles of Distributed Computing (PODC)*, Denver, CO, USA, July 2006.

[153] Zuzana Beerliová-Trubíniová and Martin Hirt. Perfectly-Secure MPC with Linear Communication Complexity. In *Proc. Theory of Cryptography Conference (TCC)*, New York, NY, USA, March 2008.

[154] Guanfeng Liang, Benjamin Sommer, and Nitin Vaidya. Experimental Performance Comparison of Byzantine Fault-Tolerant Protocols for Data Centers. In *Proc. International Conference on Computer Communications (INFOCOM)*, Orlando, FL, USA, March 2012.

[155] Arpita Patra and C. Pandu Rangan. Communication Optimal Multi-Valued Asynchronous Broadcast Protocol. In *Proc. International Conference on Cryptology and Information Security in Latin America (LATINCRYPT)*, Puebla, Mexico, August 2010.

[156] Arpita Patra and C. Pandu Rangan. Communication Optimal Multi-valued Asynchronous Byzantine Agreement with Optimal Resilience. January 2011. https://eprint.iacr.org/2009/433.pdf.

[157] Lewis Tseng and Nitin Vaidya. Byzantine Broadcast Under a Selective Broadcast Model for Single-hop Wireless Networks. January 2015. https://arxiv.org/pdf/1502.00075.pdf.

[158] Ashish Choudhury. Multi-Valued Asynchronous Reliable Broadcast With a Strict Honest Majority. In *Proc. International Conference on Distributed Computing and Networking (ICDCN)*, Hyderabad, India, January 2017.

[159] Martin Hirt and Pavel Raykov. Multi-Valued Byzantine Broadcast: The t $<$ n Case. In *Proc. International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, Kaoshiung, Taiwan, December 2014.

[160] Wutichai Chongchitmate and Rafail Ostrovsky. Information-Theoretic Broadcast with Dishonest Majority for Long Messages. In *Proc. Theory of Cryptography Conference (TCC)*, Goa, India, November 2018.

[161] Byung-Gon Chun, Petros Maniatis, Scott Shenker, and John Kubiatowicz. Attested Append-only Memory: Making Adversaries Stick to Their Word. In *Proc. Symposium on Operating Systems Principles (SOSP)*, Stevenson, WA, USA, October 2007.

[162] Allen Clement, Flavio Junqueira, Aniket Kate, and Rodrigo Rodrigues. On the (Limited) Power of Non-Equivocation. In *Proc. Symposium on Principles of Distributed Computing (PODC)*, Madeira, Portugal, July 2012.

[163] Miguel Correia, Giuliana S. Veronese, and Lau Cheuk Lung. Asynchronous Byzantine Consensus with 2F+1 Processes. In *Proc. Symposium on Applied Computing (SAC)*, Sierre, Switzerland, March 2010.

[164] Rüdiger Kapitza, Johannes Behl, Christian Cachin, Tobias Distler, Simon Kuhnle, Seyed Vahid Mohammadi, Wolfgang Schröder-Preikschat, and Klaus Stengel. CheapBFT: Resource-Efficient Byzantine Fault Tolerance. In *Proc. European Conference on Computer Systems (EuroSys)*, Bern, Switzerland, April 2012.

[165] Dave Levin, John Douceur, Jacob Lorch, and Thomas Moscibroda. TrInc: Small Trusted Hardware for Large Distributed Systems. In *Proc. Symposium on Networked Systems Design and Implementation (NSDI)*, Boston, MA, USA, April 2009.

[166] Michael Backes, Fabian Bendun, Ashish Choudhury, and Aniket Kate. Asynchronous MPC with a Strict Honest Majority Using Non-Equivocation. In *Proc. Symposium on Principles of Distributed Computing (PODC)*, Paris, France, July 2014.

[167] Danny Dolev. The Byzantine Generals Strike Again. *Journal of Algorithms*, 3(1), March 1982.

[168] Sam Toueg. Randomized Byzantine Agreements. In *Proc. Symposium on Principles of Distributed Computing (PODC)*, Vancouver, Canada, August 1984.

[169] Achour Mostéfaoui and Michel Raynal. Signature-Free Broadcast-Based Intrusion Tolerance: Never Decide a Byzantine Value. In *Proc. International Conference on Principles of Distributed Systems (OPODIS)*, Tozeur, Tunisia, December 2010.

[170] Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. Secure and Efficient Asynchronous Broadcast Protocols. In *Proc. International Cryptology Conference on Advances in Cryptology (CRYPTO)*, Santa Barbara, CA, USA, August 2001.

[171] Daniel Siewiorek and Priya Narasimhan. Fault-Tolerant Architectures for Space and Avionics Applications, 2005.

[172] John Knight, Rajat Tikoo, and Lori Stotler. Dependability in Avionics Systems. In *Digital Avionics: A Computing Perspective*. IEEE, 2006.

[173] Brendan Luksik, Andrew Loveless, and Alan George. Gatekeeper: A Reliable Reconfiguration Protocol for Real-Time Ethernet Systems. In *Proc. Digital Avionics Systems Conference (DASC)*, San Antonio, TX, USA, October 2021.

[174] Safe4RAIL-2 Newsletter, April 2019. https://safe4rail.eu/downloads/Safe4RAIL-2-Newsletter-Issue-1-April-2019.pdf.

[175] TSN Is Set to Become a Must for Industry. *Industrial Ethernet Book*, March 2022. https://iebmedia.com/technology/tsn/tsn-is-set-to-become-a-must-for-industry/.

[176] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the Presence of Partial Synchrony. *Journal of the ACM*, 35(2), April 1988.

[177] Frédéric Boulanger, Dominique Marcadet, Martin Rayrole, Safouan Taha, and Benoît Valiron. A Time Synchronization Protocol for A664 P7. In *Proc. Digital Avionics Systems Conference (DASC)*, London, UK, September 2018.

[178] Günther Bauer, Hermann Kopetz, and Peter Puschner. Assumption Coverage Under Different Failure Modes in the Time-Triggered Architecture. In *Proc. Conference on Emerging Technologies and Factory Automation (ETFA)*, Antibes-Juan les Pins, France, October 2001.

[179] Amira Zammali, Agnan de Bonneval, Yves Crouzet, Pascal Izzo, and Jean-Maxime Massimi. Communication Integrity for Future Helicopter Flight Control Systems. In *Proc. Digital Avionics Systems Conference (DASC)*, Prague, Czech Republic, September 2015.

[180] Anis Youssef, Yves Crouzet, Agnan de Bonneval, Jean Arlat, Jean-Jacques Aubert, and Patrice Brot. Communication Integrity in Networks for Critical Control Systems. In *Proc. European Dependable Computing Conference (EDCC)*, Coimbra, Portugal, October 2006.

[181] Jennifer Lundelius and Nancy Lynch. An Upper and Lower Bound for Clock Synchronization. *Information and Control*, 62(2), August 1984.

[182] TTEthernet Product Overview. Avionics Interface Technologies, 2015. http://konaka.com.tr/pdf/AS6802_TTEthernet.pdf.

[183] John Rushby. Formal Verification of Transmission Window Timing for the Time-Triggered Architecture. Technical report, SRI International, March 2001.

[184] IEEE 1588 Precise Time Protocol: The New Standard in Time Synchronization. Technical report, Microsemi, 2017.

[185] Aric Hagberg, Pieter Swart, and Daniel Chult. Exploring Network Structure, Dynamics, and Function Using NetworkX. In *Proc. Python in Science Conference (SciPy)*, Pasadena, CA, USA, August 2008.

[186] Aitech's New Customizable 3U CPCI Enclosure Combines Flexible Electronic Configurations with Rugged, Reliable Operation. Aitech Defense Systems Inc, March 2010. https://picmg.mil-embedded.com/news/aitechs-configurations-rugged-reliable-operation/.

[187] Bjoern Annighoefer, Caroline Reif, and Frank Thieleck. Network Topology Optimization for Distributed Integrated Modular Avionics. In *Proc. Digital Avionics Systems Conference (DASC)*, Colorado Springs, CO, USA, October 2014.

[188] Survivability of Systems. Technical Report AC 25.795-7, Federal Aviation Administration, October 2008.

[189] SpaceWire Cable GNSSW10028MS, March 2023. https://www.wiremasters.com/suppliers/W-L-Gore-And-Associates/catalog/products/wire-and-cable/w-l-gore-and-associates-inc-/.

[190] Cost of Space Launches to LEO, March 2023. https://ourworldindata.org/grapher/cost-space-launches-low-earth-orbit.

[191] Klaus Kursawe. Optimistic Byzantine Agreement. In *Proc. Symposium on Reliable Distributed Systems*, Suita, Japan, October 2010.

[192] Ittai Abraham, Kartik Nayak, Ling Ren, and Zhuolun Xiang. Good-case Latency of Byzantine Broadcast: A Complete Categorization. In *Proc. Symposium on. Principles of Distributed Computing (PODC)*, Virtual Event, Italy, July 2021.

[193] Leslie Lamport. Fast Paxos. Technical Report MSR-TR-2005-112, Microsoft Research, April 2006.

[194] Dan Dobre and Neeraj Suri. One-Step Consensus with Zero-Degradation. In *Proc. International Conference on Dependable Systems and Networks (DSN)*, Philadelphia, PA, USA, June 2006.

[195] Yee Jiun Song and Robbert Renesse. Bosco: One-Step Byzantine Asynchronous Consensus. In *Proc. International Symposium on Distributed Computing (DISC)*, Berlin, Heidelberg, September 2008.

[196] Nazreen Banu, Taisuke Izumi, and Koichi Wada. Doubly-Expedited One-Step Byzantine Consensus. In *Proc. International Conference on Dependable Systems and Networks (DSN)*, Chicago, IL, USA, June 2010.

[197] Jean-Philippe Martin and Lorenzo Alvisi. Fast Byzantine Consensus. In *Proc. International Conference on Dependable Systems and Networks (DSN)*, Yokohama, Japan, June 2005.

[198] Abdol-Hossein Esfahanian and S. Louis Hakimi. Fault-Tolerant Routing in DeBruijn Comrnunication Networks. *IEEE Transactions on Computers*, C-34(9), September 1985.

[199] F. J. Meyer. Flip-Trees: Fault-Tolerant Graphs with Wide Containers. *IEEE Transactions on Computers*, 37(4), April 1988.

[200] F. J. Meyer, Xiao-Tao Chen, Wei Kang Huang, and Fabrizio Lombardi. Using Virtual Links for Reliable Information Retrieval Across Point-to-Point Networks. In *Proc. International Symposium on Fault-Tolerant Computing (FTCS)*, Seattle, WA, USA, June 1997.

[201] Lewis Tseng and Nitin Vaidya. Fault-Tolerant Consensus in Directed Graphs. In *Proc. Symposium on. Principles of Distributed Computing (PODC)*, New York, NY, USA, July 2015.

[202] Lewis Tseng and Nitin Vaidya. A Note on Fault-Tolerant Consensus in Directed Networks. *ACM SIGACT News*, 47, September 2016.

[203] Günther Bauer, Wilfried Steiner, and Christian Fidi. Method and Computer System for Establishing an Interactive Consistency Property (Patent US10880043B2). TTTech Computertechnik AG, April 2020.

[204] Huynh Tu Dang, Daniele Sciascia, Marco Canini, Fernando Pedone, and Robert Soulé. NetPaxos: Consensus at Network Speed. In *Proc. Symposium on Software Defined Networking Research*, New York, NY, USA, June 2015.

[205] Huynh Tu Dang, Pietro Bressana, Han Wang, Ki Suh Lee, Noa Zilberman, Hakim Weatherspoon, Marco Canini, Fernando Pedone, and Robert Soulé. P4xos: Consensus as a Network Service. *IEEE/ACM Transactions on Networking*, 28(4), August 2020.

[206] Jialin Li, Ellis Michael, Naveen Kr Sharma, Adriana Szekeres, and Dan R. K. Ports. Just Say NO to Paxos Overhead: Replacing Consensus with Network Ordering. In *Proc. Conference on Operating Systems Design and Implementation (OSDI)*, Savannah, GA, USA, November 2016.

[207] Michael Fischer, Nancy Lynch, and Michael Paterson. Impossibility of Distributed Consensus with One Faulty Process. *Journal of the ACM*, 32(2), April 1985.

[208] Illia Safiulin and Anna Geissler. Deterministic Ethernet and IEEE TSN for Aerospace. *Modern Avionics*, August 2019. http://www.modern-avionics.com/Files/22-TTTech-Safiulin-29.08.2019.pdf.

[209] Wolfram Zischka and Mirko Jakovljevic. IEEE TSN in Aerospace: Capabilities and Future Outlook. *Modern Avionics*, July 2021. http://www.modern-avionics.com/English/Files/25-TTTech-Wolfram-22.07.2021.pdf.

[210] Max Felser. Industrial Ethernet — Real-Time Ethernet for Automation Applications. In *Industrial Communication Technology Handbook*. CRC Press, August 2014.

[211] Brendan Hall, Michael Paulitsch, Dewey Benson, and Alireza Behbahani. Jet Engine Control Using Ethernet with a BRAIN. In *Proc. Joint Propulsion Conference and Exhibit*, Hartford, CT, USA, June 2012.

[212] PCIe Card from TTTech Enables Deterministic Ethernet Connectivity for Industrial PCs. *TTTech Computertechnik AG*, February 2015. https://www.tttech.com/press/pcie-card-from-tttech-enables-deterministic-ethernet-connectivity-for-industrial-pcs/.

[213] Till Steinbach, Hyung-Taek Lim, Franz Korf, Thomas Schmidt, Daniel Herrscher, and Adam Wolisz. Tomorrow's In-Car Interconnect? A Competitive Evaluation of IEEE 802.1 AVB and Time-Triggered Ethernet (AS6802). In *Proc. Vehicular Technology Conference (VTC)*, Quebec City, Canada, September 2012.

[214] Till Steinbach, Franz Korf, and Thomas Schmidt. Comparing Time-Triggered Ethernet with FlexRay: An Evaluation of Competing Approaches to Real-Time for In-Vehicle Networks. In *Proc. International Workshop on Factory Communication Systems (WFCS)*, Nancy, France, May 2010.

[215] Wilfried Steiner and Michael Paulitsch. Time-Triggered Ethernet. In *Industrial Communication Technology Handbook*. CRC Press, August 2014.

[216] Wilfried Steiner. TTEthernet: Time-Triggered Services for Ethernet Networks. In *Proc. Digital Avionics Systems Conference (DASC)*, Orlando, FL, USA, October 2009.

[217] Time-Triggered Ethernet Slims Down Critical Data Systems. *NASA Spinoff*, 2018. https://spinoff.nasa.gov/Spinoff2018/t_4.html.

[218] David Escorial and Mark Hann. A Combined Dependability and Security Approach for Third Party Software in Space Systems. In *Proc. European Dependable Computing Conference (EDCC)*, Gothenburg, Sweden, September 2016.

[219] Jessica Fichuk. Safe to Fly: Certifying COTS Hardware for Spaceflight. Technical Report JSC-CN-22144, NASA Johnson Space Center, August 2013.

[220] Compositional Assurance Cases and Arguments for Distributed MILS. Technical report, D-MILS Project Partners, April 2015.

[221] Roman Obermaisser. Properties of Time-Triggered Communication Systems — Certifiability. In *Time-Triggered Communication*. CRC Press, 2011.

[222] Matt Cichowicz. From Schenley Place to Outer Space: Pitt Team Developing Computers for Space Station. *Pitt Virtual Newsroom*, July 2017. https://news.engineering.pitt.edu/from-schenley-place-to-outer-space/.

[223] Risk Classification for NASA Payloads. Technical Report NPR 8705.4, National Aeronautics and Space Administration, June 2004.

[224] RFC 826: An Ethernet Address Resolution Protocol. Internet Engineering Task Force (IETF), November 1982.

[225] Travis Goodspeed, Sergey Bratus, Ricky Melgares, Rebecca Shapiro, and Ryan Speers. Packets in Packets: Orson Welles' In-Band Signaling Attacks for Modern Radios. In *Proc. Workshop on Offensive Technologies (WOOT)*, San Francisco, CA, USA, August 2011.

[226] Andrew Loveless. On TTEthernet for Integrated Fault-Tolerant Spacecraft Networks [Slides]. In *Proc. AIAA SPACE and Astronautics Forum and Exposition*, Pasadena, CA, USA, August 2015. https://ntrs.nasa.gov/api/citations/20170009923/downloads/20170009923.pdf.

[227] Klaus Kainrath, Martin Fruhmann, Klaus Gebeshuber, Erich Leitgeb, and Mario Gruber. Evaluation of Cyber Security in Digital Avionic Systems. In *Proc. Vehicular Technology Conference (VTC)*, Antwerp, Belgium, May 2020.

[228] Wilfried Steiner, Günther Bauer, Brendan Hall, and Michael Paulitsch. Time-Triggered Ethernet — Protocol Services, Communication Services, Media Access Control. In *Time-Triggered Communication*. CRC Press, 2011.

[229] Wilfried Steiner. Candidate Security Solutions for TTEthernet. In *Proc. Digital Avionics Systems Conference (DASC)*. East Syracuse, NY, USA, October 2013.

[230] Ted Bonk, William Todd Smithgall, Mitch Fletcher, and Greg Carlucci. System and Method for a Cross Channel Data Link (Patent US20100183016A1). Honeywell International Inc, July 2010.

[231] Markus Plankensteiner. TTTech Company Overview. TTTech Computertechnik AG, March 2011. https://www.slideshare.net/TTTech/tttech-2011companyoverview.

[232] Timothy Trippel, Kang Shin, Kevin Bush, and Matthew Hicks. Bomberman: Defining and Defeating Hardware Ticking Timebombs at Design-Time. In *Proc. Symposium on Security and Privacy (S&P)*, San Francisco, CA, USA, May 2021.

[233] Samuel King, Joseph Tucek, Anthony Cozzie, Chris Grier, Weihang Jiang, and Yuanyuan Zhou. Designing and Implementing Malicious Hardware. In *Proc. Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, San Francisco, CA, USA, April 2008.

[234] M. Orlandi, T. Rohr, M. Stienstra, and C. Semprimoschnig. The Role of ESA TEC-QTE in the ISS Safety Process. In *Proc. International Association for the Advancement of Space Safety (IAASS) Conference*, Montreal, Canada, May 2013.

[235] Gail Johnson-Roth. Mission Assurance Guidelines for A-D Mission Risk Classes. Technical Report TOR-2011(8591)-21, The Aerospace Corporation, June 2011.

[236] Christopher Wilson, Jacob Stewart, Patrick Gauvin, James MacKinnon, James Coole, Jonathan Urriste, Alan George, Gary Crum, Elizabeth Timmons, Jaclyn Beck, Tom Flatley, Mike Wirthlin, Alex Wison, and Aaron Stoddard. CSP Hybrid Space Computing for STP-H5/ISEM on ISS. In *Proc. Conference on Small Satellites*, Logan, UT, USA, August 2015.

[237] New Semi-Rugged Laptop for Manufacturing and Automotive Professionals, January 2021. https://www.logisticsbusiness.com/it-in-logistics/mobile-computing-rfid/new-semi-rugged-laptop/.

[238] Alexander Deutschinger. TTE Controller. EtherSpace Consortium, April 2021. https://www.tttech.com/wp-content/uploads/2021-04-22_P1b_Webinar-EtherSpace_TTTech_presentation.pdf.

[239] IEEE 802.3-2018: IEEE Standard for Ethernet. Institute of Electrical and Electronics Engineers, June 2018.

[240] Christian Fidi and Andrew Loveless. A Modular, Scalable Avionics Architecture for Future Exploration Missions. In *Proc. AIAA SPACE and Astronautics Forum and Exposition*, Orlando, FL, USA, September 2017.

[241] TTE-Plan User Manual (v.5.4.6). TTTech Computertechnik AG, July 2020.

[242] TTE Switch Module A664 Pro. TTTech Computertechnik AG, August 2022. https://www.tttech.com/wp-content/uploads/TTTech_TTE-Switch_Module_Core_2-1.pdf.

[243] TTE Verify. TTTech Computertechnik AG, August 2022. https://www.tttech.com/products/products/aerospace/development-test-vv/verification-tools/tte-verify/.

[244] Ananda Basu, Saddek Bensalem, Marius Bozga, Benoît Delahaye, Axel Legay, and Emmanuel Sifakis. Verification of an AFDX Infrastructure Using Simulations and Probabilities. In *Proc. Conference on Runtime Verification*, Berlin, Germany, November 2010.

[245] iSAFT Test Tool for Deterministic On-board Ethernet Networks. In *Proc. ESA Workshop on ADCSS*, Noordwijk, Netherlands, November 2019. https://indico.esa.int/event/323/contributions/5041/attachments/3743/5198/11.30b_-_Teletel_-_Ethernet_for_Space_with_TSN.pdf.

[246] TT6802-1-SW: The TTE Switch Controller Space. TTTech Computertechnik AG, August 2022. https://www.tttech.com/wp-content/uploads/TTTech_12792_TTE-Switch_Controller_Space.pdf.

[247] TTE Switch Space 3U cPCI. TTTech Computertechnik AG, August 2022. https://www.tttech.com/products/products/space/flight-rugged-hardware/ttethernet-space-equipment/switch-space-3u-cpci/.

[248] Ben Seri, Gregory Vishnepolsky, and Yevgeny Yusepovsky. EtherOops: Bypassing Firewalls and NATs by Exploiting Packet-in-Packet Attacks in Ethernet. Technical report, Armis, August 2020.

[249] Andrea Barisani and Daniele Bianco. Fully Arbitrary 802.3 Packet Injection — Maximizing the Ethernet Attack Surface. In *Proc. Black Hat*, Las Vegas, NV, USA, July 2013.

[250] XCORE-200. XMOS, August 2022. https://www.xmos.ai/xcore-200/.

[251] Arne Neumann, Martin Jan Mytych, Derk Wesemann, Lukasz Wisniewski, and Jürgen Jasperneite. Approaches for In-Vehicle Communication — An Analysis and Outlook. In *Proc. International Conference on Computer Networks (CN)*, Landek, Poland, June 2017.

[252] Bob Rutemiller and John Weber. Communications and Connectivity — Ethernet In-depth, Wiring Types. In *Automation of Water Resource Recovery Facilities: WEF Manual of Practice No. 21*. Water Environment Federation, 2013.

[253] Mark Baker. Test Circuit Design Considerations — Printed Circuit Board Physics. In *Demystifying Mixed-Signal Test Methods*. Elsevier Science, 2003.

[254] TTE Switch A664 Lab v2.0. TTTech Computertechnik AG, August 2022. https://www.tttech.com/wp-content/uploads/TTTech_TTE-Switch_A664_Lab_v2.0.pdf.

[255] Joe Randolph. Designing Ethernet Cable Ports to Withstand Lightning Surges. *In Compliance Magazine*, December 2016. https://incompliancemag.com/article/designing-ethernet-cable-ports-to-withstand-lightning-surges/.

[256] Comidox 15KV Boost High Voltage Generator, August 2022. https://www.amazon.com/dp/B07JG4K6S6?psc=1&ref=ppx_yo2_dt_b_product_details.

[257] vacuum spark gap, August 2022. https://www.ebay.com/sch/i.html?_from=R40&_trksid=p2380057.m570.l1313&_nkw=vacuum+spark+gap&_sacat=0.

[258] TTE Switch OBC HiRel. TTTech Computertechnik AG, August 2022. https://www.tttech.com/wp-content/uploads/TTTech_TTE-Switch-OBC-HiRel-Box-1.pdf.

[259] Nick Flaherty. Time Triggered Ethernet for Space Gateway. *EE News Europe*, May 2021. https://www.eenewseurope.com/news/time-triggered-ethernet-space.

[260] Data Network Evaluation Criteria Report. Technical Report DOT/FAA/AR-09/27, Federal Aviation Administration, June 2009.

[261] Andrew Loveless. Overview of TTE Applications and Development at NASA/JSC. In *Proc. CCSDS SOIS SUBNET Working Group Meeting*, Rome, Italy, October 2016. https://ntrs.nasa.gov/api/citations/20160012363/downloads/20160012363.pdf.

[262] Joan Vila-Carbo, Joaquim Tur-Massanet, and Enrique Hernandez-Orallo. Analysis of Switched Ethernet for Real-Time Transmission. In *Factory Automation*. IntechOpen, March 2010.

[263] Miladin Sandić, Ivan Velikić, and Aleksandar Jakovljević. Calculation of Number of Integration Cycles for Systems Synchronized using the AS6802 Standard. In *Proc. Zooming Innovation in Consumer Electronics International Conference (ZINC)*, Novi Sad, Serbia, May 2017.

[264] Mohammed Abuteir, Zaher Owda, Cristina Zubia, Felix Casado, Marcello Coppola, Lukas Kohutka, Arjan Geven, Jorn Migge, and Manuel Muñoz. Distributed Real-Time Architecture for Mixed Criticality Systems (DREAMS). Technical Report Fault Injection Framework D5.2.3, University of Siegen, July 2016.

[265] Jean-Baptiste Chaudron. TTEthernet: Theory, Concepts, and Applications. In *Proc. Ecole temps-réel (ETR)*, Rennes, France, August 2015. http://etr2015.irisa.fr/images/presentations/TTEthernet_ETR_2015_Rennes.pdf.

[266] Cristina Plettner. COTS for Space. In *Proc. Radiation Effects on Components and Systems (RADECS)*, Geneva, Switzerland, October 2017. https://indico.cern.ch/event/649606/sessions/246265/attachments/1522111/2403688/COTS_plettner_v2.pdf.

[267] Gerald Cohen, William Lee, and Michael Strickland. Final Report: Design of an Integrated Airframe/Propulsion Control System Architecture. NASA Contractor Report 182007, Boeing Advanced Systems, March 1992.

[268] Ricky Butler. A Survey of Provably Correct Fault-Tolerant Clock Synchronization Techniques. Technical Report NASA-TM-100553, NASA Langley Research Center, February 1988.

[269] Fred Schneider. Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial. *ACM Computing Surveys*, 22(4), December 1990.

[270] Asteroid Redirect Robotic Mission. NASA Jet Propulsion Laboratory, August 2022. https://www.jpl.nasa.gov/missions/asteroid-redirect-robotic-mission-arrm.

[271] Trick 13 Simulation Environment. NASA Johnson Space Center, August 2022. https://software.nasa.gov/software/MSC-25665-1.

[272] IEEE 802.1D-2004: IEEE Standard for Local and Metropolitan Area Networks — Media Access Control (MAC) Bridges. Institute of Electrical and Electronics Engineers, June 2004.

[273] Ronn Kliger. Integrated Transformer-Coupled Isolation. *IEEE Instrumentation and Measurement Magazine*, March 2003.

[274] Tupavco TP302 Ethernet Surge Protector PoE+ Gigabit RJ45 Lightning Suppressor, August 2022. https://www.tupavco.com/products/ethernet-surge-protector.

[275] Wilfried Steiner and Michael Paulitsch. TTEthernet in Orion. In *Industrial Communication Technology Handbook*. CRC Press, August 2014.

[276] Anselm Breitenreiter, Jesús López, Pedro Reviriego, Milos Krstic, Úrsula Gutierro, Manuel Sánchez-Renedo, and Daniel González. A Radiation Tolerant 10/100 Ethernet Transceiver for Space Applications. In *Proc. International Symposium on On-Line Testing and Robust System Design (IOLTS)*, Rhodes, Greece, July 2019.

[277] SEPHY: Space Ethernet Physical Layer Transceiver. European Union's Horizon 2020 Programme, 2022. https://www.tttech.com/innovation/research-projects/eu-h2020/sephy/.

[278] Shannon Melton, Scott Simmons, Byron Smith, and Douglas Hamilton. Telemedicine. In *Principles of Clinical Medicine for Space Flight*. Springer Nature, 2019.

[279] IEEE 802.1AE-2018: IEEE Standard for Local and Metropolitan Area Networks — Media Access Control (MAC) Security. Institute of Electrical and Electronics Engineers, December 2018.

[280] John Rushby. A Comparison of Bus Architectures for Safety-Critical Embedded Systems. Technical report, SRI International, September 2001.

[281] K. Kainrath, K. Gebeshuber, M. Fruhmann, and M. Gruber. Aviation Cyber Security Study. Technical report, Munich, Germany, 2017.

[282] Generation of a Testbed for the Validation of the TTEthernet Technology. Technical report, Airbus Defence and Space, Bremen, Germany, December 2019.

[283] Daniel Onwuchekwa and Roman Obermaisser. Fault Injection Framework for Assessing Fault Containment of TTEthernet Against Babbling Idiot Failures. In *Proc. International Symposium on Quality of Service (IWQoS)*, Alberta, Canada, June 2018.

[284] Daniel Onwuchekwa. *Fault Injection Framework for Time-Triggered Systems*. PhD thesis, University of Siegen, Siegen, Germany, October 2020.

[285] Florian Skopik, Albert Treytl, Arjan Geven, Bernd Hirschler, Thomas Bleier, Andreas Eckel, Christian El-Salloum, and Armin Wasicek. Towards Secure Time-Triggered Systems. In *Proc. International Conference on Computer Safety, Reliability, and Security (SAFECOMP)*, Magdeburg, Germany, September 2012.

[286] Pieter Robyns, Peter Quax, and Wim Lamotte. Injection Attacks on 802.11n MAC Frame Aggregation. In *Proc. Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*, New York, NY, USA, June 2015.

[287] Elya Joffe. Assessment of the Robustness of Commercial Data Communication Interfaces to a Military EMI Environment. In *Proc. International Symposium on Electromagnetic Compatibility (EMC)*, Detroit, MI, August 2008.

[288] Matthias Kreitlow, Frank Sabath, and Heyno Garbe. Analysis of IEMI Effects on a Computer Network in a Realistic Environment. In *Proc. International Symposium on Electromagnetic Compatibility (EMC)*, Dresden, Germany, August 2015.

[289] Sebastian Jeschke, J. Loos, Michael Kleinen, O. Kurt, Jörg Bärenfänger, Christian Hangmann, and Ingo Wüllner. Susceptibility of 100Base-T1 Communication Lines to Coupled Fast Switching High-Voltage Pulses. In *Proc. International Symposium on Electromagnetic Compatibility (EMC) Europe*, Rome, Italy, September 2020.

[290] Akira Tsukada, Ken Okamoto, Yuichiro Okugawa, Jun Kato, and Makoto Nagata. System-Level Response of Ethernet Linkage to Bulk Current Injection into Cables. In *Proc. International Symposium on Electromagnetic Compatibility (EMC) Europe*, Rome, Italy, September 2020.

[291] Chien-Ying Chen, Sibin Mohan, Rodolfo Pellizzoni, Rakesh Bobba, and Negar Kiyavash. A Novel Side-Channel in Real-Time Schedulers. In *Proc. Real-Time and Embedded Technology and Applications Symposium (RTAS)*, Montreal, Canada, April 2019.

[292] Songran Liu and Wang Yi. Task Parameters Analysis in Schedule-Based Timing Side-Channel Attack. *IEEE Access*, 8, September 2020.

[293] Man-Ki Yoon, Sibin Mohan, Chien-Ying Chen, and Lui Sha. TaskShuffler: A Schedule Randomization Protocol for Obfuscation against Timing Inference Attacks in Real-Time Systems. In *Proc. Real-Time and Embedded Technology and Applications Symposium (RTAS)*, Vienna, Austria, April 2016.

[294] Chien-Ying Chen, Monowar Hasan, AmirEmad Ghassami, Sibin Mohan, and Negar Kiyavash. REORDER: Securing Dynamic-Priority Real-Time Systems Using Schedule Obfuscation. June 2018. http://arxiv.org/abs/1806.01393.

[295] Nils Vreman, Richard Pates, Kristin Krüger, Gerhard Fohler, and Martina Maggio. Minimizing Side-Channel Attack Vulnerability via Schedule Randomization. In *Proc. Conference on Decision and Control (CDC)*, Nice, France, December 2019.

[296] Mitra Nasri, Thidapat Chantem, Gedare Bloom, and Ryan Gerdes. On the Pitfalls and Vulnerabilities of Schedule Randomization Against Schedule-Based Attacks. In *Proc. Real-Time and Embedded Technology and Applications Symposium (RTAS)*, Montreal, Canada, April 2019.

[297] Kristin Krüger, Gerhard Fohler, and Marcus Völp. Improving Security for Time-Triggered Real-Time Systems against Timing Inference Based Attacks by Schedule Obfuscation. In *Proc. Euromicro Conference on Real-Time Systems (ECRTS)*, Dubrovnik, Croatia, June 2017.

[298] Kristin Krüger, Marcus Völp, and Gerhard Fohler. Vulnerability Analysis and Mitigation of Directed Timing Inference Based Attacks on Time-Triggered Systems. In *Proc. Euromicro Conference on Real-Time Systems (ECRTS)*, Barcelona, Spain, July 2018.

[299] USBKill, August 2022. https://usbkill.com/.

[300] Etherkiller — Coming Soon to a NIC Near You, August 2022. https://etherkiller.org/.

[301] Grigorios Fragkos. A Weapon for the Mass Destruction of Computer Infrastructures. September 2015. https://gfragkos.blogspot.com/2015/09/a-weapon-for-mass-destruction-of.html.

[302] Ertem Esiner, Utku Tefek, Hasan S. M. Erol, Daisuke Mashima, Binbin Chen, Yih-Chun Hu, Zbigniew Kalbarczyk, and David M. Nicol. LoMoS: Less-Online/More-Offline Signatures for Extremely Time-Critical Systems. *IEEE Transactions on Smart Grid*, 13(4), July 2022.

[303] IEEE 1588-2008: IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems. Institute of Electrical and Electronics Engineers, March 2008.

[304] ECSS-E-ST-50-11C: SpaceFibre — Very High-Speed Serial Link. European Cooperation for Space Standardization, May 2019.

[305] SpaceFibre Quality of Service. March 2023. https://www.star-dundee.com/spacefibre/getting-started/spacefibre-quality-of-service/.