IGOR: Accelerating Byzantine Fault Tolerance for Real-Time Systems with Eager Execution

Andrew Loveless*‡ Ronald Dreslinski* Baris Kasikci* Linh Thi Xuan Phan†
*University of Michigan †University of Pennsylvania ‡NASA Johnson Space Center
*{loveless, rdreslin, barisk}@umich.edu †linhphan@seas.upenn.edu

***Abstract*—Critical real-time systems like spacecraft and aircraft commonly use Byzantine fault-tolerant (BFT) state machine replication (SMR) to mask faulty processors and sensors. Unfortunately, existing BFT SMR techniques require replicas to reach agreement on redundant sensor data and perform source selection before executing, which adds unavoidable latency to every execution and inevitably decreases control performance. The standard way to reduce the latency of BFT SMR in non-real-time systems is to use speculation, forgoing agreement on inputs altogether, and repeating executions when faults occur. However, this approach is not suitable for real-time systems, since its worst-case latency when faults occur can be even higher than that of non-speculative systems.**

This paper presents IGOR, a new speculative BFT SMR approach that leverages multi-core processors to avoid the added latency inherent to traditional BFT SMR techniques in both the absence and presence of faults. The key idea of IGOR is to eagerly execute on data from redundant sensors in parallel. While these executions are underway, the replicas reach agreement on which execution will determine the system's final state. As a result, IGOR's latency is reduced to the time taken by the executions or by the agreement process, whichever is longer. Our evaluation shows that IGOR reduces latency by up to 1.75× and improves schedulability by 1.88–3.22× compared to the state of the art. We also show that when used to execute spacecraft guidance, navigation, and control software during a dynamic mission phase, IGOR noticeably increases vehicle stability.

***Index Terms*—Byzantine fault tolerance, state machine replication, speculative execution, multi-core, real-time systems, distributed systems**

I. 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 [1]–[16]. 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 [8], [15], [17]. The replicas then perform a *source selection* to choose a single "best" sensor value as the system input [5], [15], [18]. The source selection step is typically application-specific, and can be as simple as a mid-value selection [5], [15]. 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 [20], 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 §VI, 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 [21].

A standard approach to addressing this challenge in non-real-time systems, such as data centers and blockchains [22]–[27], 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 [22]–[26]. 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 [10], [28] to enable speculation <u>without</u> 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 §VI), 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.

[1]We borrow the term "agree-execute" from [19], but narrow its meaning to refer specifically to BFT systems that agree on inputs before executing.

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. §IV-A). 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) [29], an open-source general-purpose flight software framework used in a variety of real spacecraft, including the Lunar Reconnaissance Orbiter and Parker Solar Probe [30]. 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 (§IV).
- A prototype of IGOR for NASA's cFS framework (§V).
- An experimental evaluation of IGOR, including benchmarks (§VI-A) and schedulability analysis (§VI-B).
- A case study of IGOR in a spaceflight application (§VI-E).

II. Background and Challenges

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

A. 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 [1]–[16]. 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 [5], [31]–[33]. 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".

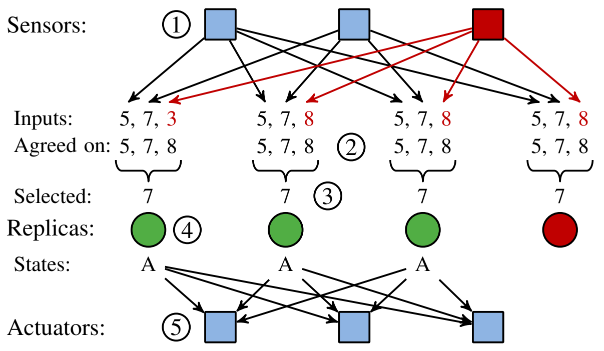

Fig. 1: Overview of traditional agree-execute BFT SMR systems. Sensor 3 and replica 4 are faulty.

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 [34]). To ensure that the actual *content* of the inputs is identical, the replicas run an agreement protocol on all inputs before executing [5], [17].

Figure 1 illustrates this "agree-execute" approach: ① Each sensor sends its value to each of the replicas; since sensor 3 is faulty, some replicas receive different values from this sensor. ② The replicas then use an agreement protocol to agree on the value from each sensor, and ③ perform source selection[2] to choose a single "best" sensor value to use as input for the computation. ④ 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] ⑤ 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.

B. 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 [35]–[37]. Therefore, the round length

[2]The source selection process is typically application-specific; it could be as simple as a mid-value selection [15].

[3]A state change does not always require an output to be sent to the actuators.

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 [38]–[40], the message WCTT – and thus minimum round length – can be on the order of *several milliseconds* [38]–[42]. For example, in deterministic Ethernet (AFDX [43]) networks used in avionic systems, the WCTT can be up to 3–4 ms for individual Ethernet frames [38], [42]. Large messages must be fragmented into multiple frames sent *at least* 1 ms apart [43], and more commonly 4+ ms apart [38], which can result in message WCTTs that are multiple times that of a single frame [43]. We note that since agreement protocols often require replicas to send increasingly larger messages in each round [35]–[37], [44], [45], 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 [46]–[49] can reduce frame WCTTs to hundreds of microseconds [31], [50]. 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 [31], [51]. 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 [52], [53], which is often 10+ ms to avoid excessive processing overhead [53], [54]. Lastly, large messages still need to be fragmented across multiple time slots [41].

Since a single round can take several milliseconds, and any deterministic agreement protocol must take at least $f + 1$ rounds [20], 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 [55], [56], thus resulting in end-to-end latencies that are 1.5–2$\times$ higher than that of a non-replicated system (see §VI).

The standard way to reduce the *average* latency of BFT SMR in non-real-time systems is to adopt a *speculative* approach [22]–[26] 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.

III. Models

This section describes our system and failure models. Our models are consistent with real-time systems that rely on BFT in practice [1], [2], [4], [8], [15], [32].

A. 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 [36], [57], [58]. This is typically accomplished using specialized real-time operating systems [59], [60] and networking protocols [8], [43], [48], [49], [61], [62].

We assume all devices are synchronized, either via the network [48], [49], [61] or external timing equipment [63], 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 [5], [31], [32], [64]. The network can use any physical topology, which may include switches [43], [48], [61], buses [49], [64], or point-to-point links [1], [3], [8]. 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 [36], since otherwise a faulty device could impersonate all the other devices [65]. 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 [43], [48], [61], or (2) a time-division multiple access scheme [4], [32], [62], [64], in which the sender is implied by the time a message is received.

B. Failure Model

We consider the classical Byzantine failure model, where, in a system with n replicas and m sensors, $f < n/3$ of the replicas [36] and $g < m/2$ of the sensors [14], [66] 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 [67], device wear-out [68], and bits flips from charged particles [69]. Broadcast networks can prevent some of these behaviors, since messages are sent only once by the device and replicated by the switch or bus [43], [48]. 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

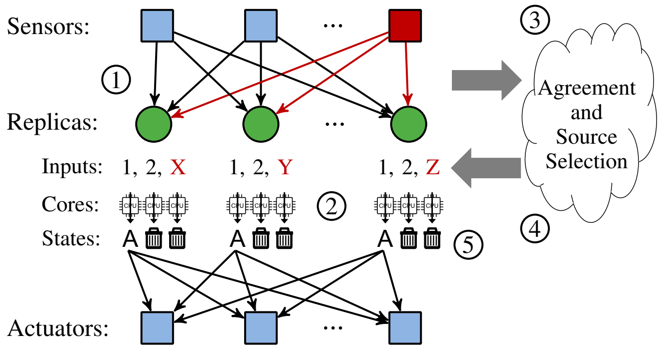

Fig. 2: High-level design of IGOR. Sensor 3 is faulty.

while others from network B first [4]. A faulty device can also transmit a marginal signal that is interpreted differently by different receivers receiving the same message [70].

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 [61], or redundant network planes [43], [48], [49], 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 [4], [5], or by using specialized self-checking switches that are proven to fail silent with sufficiently high probability [32], [64], [71]. We note that by not worrying about network failures, 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 failures.

Finally, in safety-critical systems, where BFT is perhaps most used in practice [1]–[8], [11]–[15], cryptographic methods like digital signatures are typically considered an insufficient means of constraining the behavior of faulty devices [72]–[75]. We design IGOR to avoid cryptographic assumptions and to be correct in all possible executions.

IV. 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 deadlines otherwise. It also lets IGOR take advantage of emerging aerospace processors, which can feature 4, 8, or more cores [10], [28].

Figure 2 gives an overview of IGOR's design. ① 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. ② 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. ③ 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. ④ 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. ⑤ 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 [76]–[79] 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 §VI-A). Second, IGOR exploits the 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 (§IV-A). We then modify the protocol using a binary reduction technique to make a version of IGOR optimized for tolerating multiple faults (§IV-B).

A. Optimizing Eager Execution for the Single-Fault Case

Figure 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.

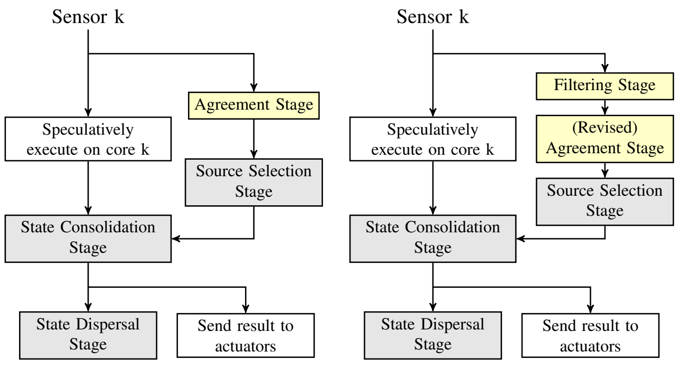

Fig. 3: Stages of IGOR optimized for the single (left) and multiple (right) fault-tolerant cases.

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 [35], [80], [81]. Even agreement protocols with stronger guarantees [82], 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 1, which the replicas execute on each sensor value. It uses an existing Byzantine agreement (BA) protocol [36], [80], [81] as a primitive, which guarantees that, when a faulty replica broadcasts a value, all other non-faulty replicas receive the same value [36].

In Protocol 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 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 $\mathit{Values}_k$ 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 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. □

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 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: 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 $\mathcal{C}$
  $\mathit{SelectedSource} \leftarrow$ the ID of the selected sensor
```

LEMMA 2. *All non-faulty replicas have the same set $\mathcal{C}$ and the same SelectedSource.*

Proof. By Lemma 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 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). □

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 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 3: State Consolidation Stage

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

LEMMA 4. At least $n - 2f$ non-faulty replicas have $\mathit{MyAccept}_{\mathit{SelectedSource}} =$ `True` and have the same *SavedState*. At most f non-faulty replicas have $\mathit{MyAccept}_{\mathit{SelectedSource}} =$ `False` (i.e., do not have the state).

Proof. By Lemma 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 $\mathit{MyAccept}_k \leftarrow$ `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. □

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 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 4: State Dispersal Stage

```
• if $\mathit{MyAccept}_{\mathit{SelectedSource}}$ = True then:
      Broadcast SavedState to all replicas
• if $\mathit{MyAccept}_{\mathit{SelectedSource}}$ = 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.
Deliver Outputs. At the same time as the State Dispersal Stage, the replicas deliver outputs to the actuators. The protocol is shown in Protocol 5. The actuators use a majority vote of the outputs to resolve the final system output.

Protocol 5: Deliver Outputs

```
• if $\mathit{MyAccept}_{\mathit{SelectedSource}}$ = True then:
      Reference SavedState to determine the output
      Send the output to actuators
```

We now prove the correctness of the overall protocol.

THEOREM 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 and 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 [5], [15], [83], 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 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. □

B. 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 §VI-A, 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 [76]–[79]. 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 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.

Filtering Stage. The protocol for the Filtering Stage is shown in Protocol 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 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 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 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. □

(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 7. The replicas run a separate instance of the protocol for each sensor. The protocol uses any existing BA protocol as a primitive [36], [80], [81]. *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 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 §IV-A.

(REVISED) LEMMA 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 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 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 5, all non-faulty replicas set *MyValue*$_k \leftarrow v$ and *MyAccept*$_k \leftarrow$ `True`. All non-faulty replicas then broadcast `True` with a BA protocol, which each non-faulty replica stores in *Accepts*$_k$. As there are $\geq n-f$ non-faulty replicas, *Accepts*$_k$ contains $\geq n-f$ `True` bits for all non-faulty replicas. Thus, all non-faulty replicas set *Candidate*$_k \leftarrow$ `True`, and $\mathcal{C}$ contains (k, v). □

(REVISED) LEMMA 4. At least $n - 2f$ non-faulty replicas have *MyAccept*$_{\textit{SelectedSource}}$ = `True` and have the same *SavedState*. At most f non-faulty replicas have *MyAccept*$_{\textit{SelectedSource}}$ = `False` (i.e., do not have the state).

Proof. By Lemma 2, all non-faulty replicas set *SelectedSource* to the same sensor k from $\mathcal{C}$. A sensor k is in $\mathcal{C}$ only if *Candidate*$_k$ = `True`, which means $\geq n-f$ replicas claimed to set *MyAccept*$_k \leftarrow$ `True`. Up to f of these replicas are faulty, so $\geq n - 2f$ non-faulty replicas actually did set *MyAccept*$_k \leftarrow$ `True`. By Lemma 6, these $\geq n-2f$ non-faulty replicas all had the same *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 *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 *MyAccept*$_{\textit{SelectedSource}} \leftarrow$ `True`. In the worst case, all replicas which did not claim to set *MyAccept*$_{\textit{SelectedSource}} \leftarrow$ `True` are non-faulty. Thus, at most f non-faulty replicas do not have the state. □

V. PROTOTYPE IMPLEMENTATION

To evaluate our solution, we built a prototype of IGOR in NASA's Core Flight System (cFS) [29], an open-source software framework used in a variety of real spacecraft [30] (and planned to be used in several future NASA missions [84]). 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 [10] (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 [54], 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 [43] network found in typical aircraft [85]. 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 [38], [42]. 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 [36], [37], which has the theoretical minimum number of rounds [20] and is used extensively in practice [1]–[3], [6], [8]. OM is also the basis for all existing Byzantine agreement protocols that meet this lower bound [35], [86], [87]. TC is an agree-execute system based on Turpin and Coan's reduction protocol [76], which uses a similar reduction approach to IGOR and takes fewer rounds than any existing protocol that uses a binary reduction [77], [79], [88], [89]. 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.

VI. 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?

A. 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 [38], [56]. (Note that the sensor data size is relatively large, since data delivered to the flight computers are often batched by downstream devices [32], [83].) 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 system used in our case study (§VI-E). The number of sensors was 3, the typical redundancy level found in aircraft and spacecraft [14]. The band allocation gap

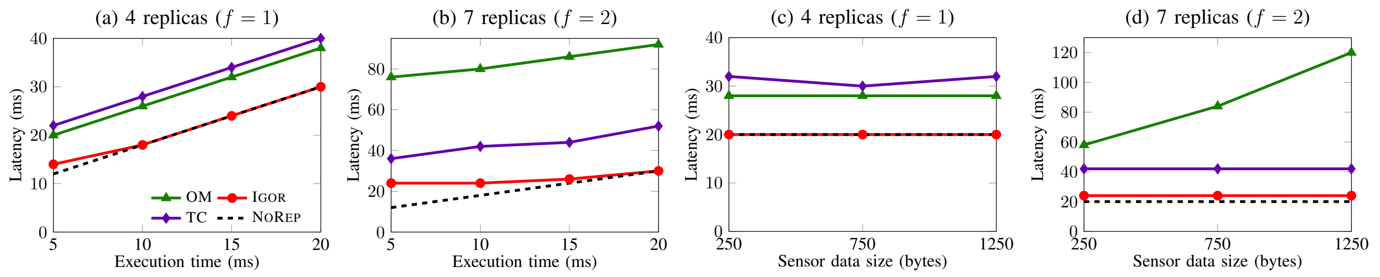

Fig. 4: End-to-end latencies of IGOR and other state-of-the-art protocols compared to a non-replicated system. Plots (a) and (b) use 750 bytes of sensor data with 3 sensors. Plots (c) and (d) use 12 ms execution times with 3 sensors.

(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 [43].

The workloads executed on the experimental platform described in §V. 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 [41], [54], [90]). 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 4 shows the results of all four systems under each fault setting, as we varied the 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 (§IV-B) 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.

B. 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 [53]. 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 [91]. Per cFS' design, tasks were scheduled without splitting into smaller sub-tasks. We used 2.5 ms slots to accommodate the tasks' periods. We determined 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 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

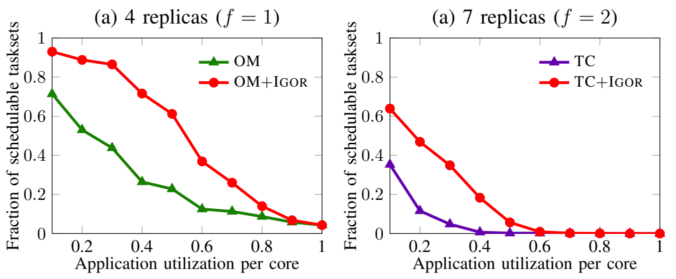

Fig. 5: Increase in schedulability when using IGOR.

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.

C. Computation Overhead

Experimental setup. To evaluate IGOR's computation overhead, we repeated the same experiment as in §VI-B. 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 IGOR, where IGOR was used for the tasks that could not meet deadlines in (1).

Results. Figure 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 4 and 5.

D. 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

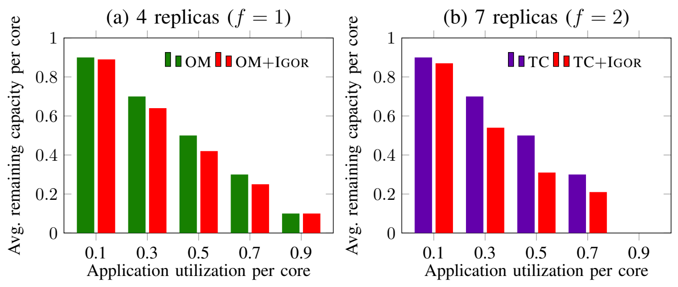

Fig. 6: Available compute capacity when using IGOR.

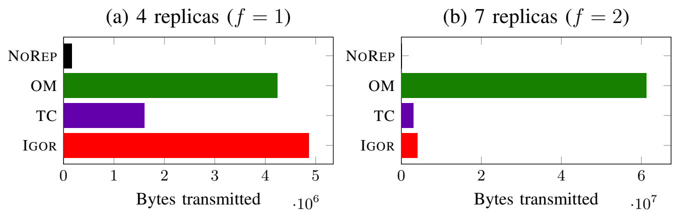

Fig. 7: Total bytes transmitted over 100 iterations.

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 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 [19], and 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).

E. 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.

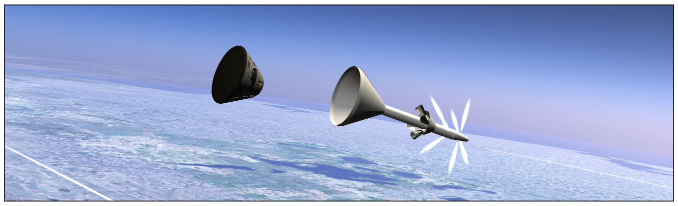

Fig. 8: Orion AA-2 simulation moments after the crew module (left) separates from the launch abort system (right).

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 [41]. The software ran against a high-fidelity simulation, which models the vehicle's trajectory, as well as the sensors and actuators. Figure 8 shows a screenshot of the simulation.

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 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.

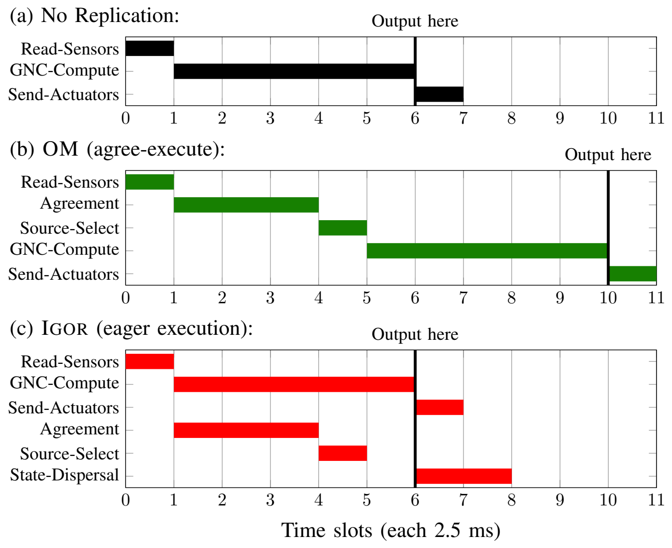

Fig. 9: Time slot allocations for IGOR and OM when running the AA-2 flight software in the single-fault case.

VII. 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 [22]–[26]. 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 [92]–[96]. 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 [97]. 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 [19], [98]–[103]. This approach greatly increases throughput when requests are mostly independent, but does not significantly reduce the latency of executing individual requests. In general, IGOR is orthogonal to these techniques, but it does significantly reduce latency. Execute-Verify [19] systems allow multi-core replicas to execute dependent requests non-deterministically; 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 [76]–

[79], [104]–[113]. 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 Byzantine agreement protocol [76].

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 [111], [114]–[119]. Other protocols prevent equivocation by having devices echo values they receive to one another, and make decisions based on a quorum of matching echoes [65], [120]–[123]. 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.

VIII. 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 spaceflight application, and we believe it is broadly applicable to other BFT systems seeking improved real-time control performance.

ACKNOWLEDGEMENTS

We thank the anonymous reviewers for their helpful suggestions on improving the paper. This research was supported in part by the NSF Graduate Research Fellowship (award DGE 1256260) and NSF grant CNS-1750158.

AVAILABILITY

The IGOR implementation used in our evaluation is available at https://github.com/efeslab/igor-ae.

REFERENCES

[1] B. Wolff and P. Scheffers, "DMS-R, the Brain of the ISS: 10 Years of Continuous Successful Operation in Space," in *Proc. DASIA*, Drubrovnik, Croatia, Aug. 2012.

[2] C. Marchant, "Ares I Avionics Introduction," in *Proc. NASA/ARMY Software and Systems Forum*, Huntsville, AL, USA, Nov. 2009.

[3] C. Kouba, D. Buscher, and J. Busa, "The X-38 Spacecraft Fault-Tolerant Avionics System," in *Proc. MAPLD*, Washington, DC, USA, Aug. 2003.

[4] A. Loveless, C. Fidi, and S. Wernitznigg, "A Proposed Byzantine Fault-Tolerant Voting Architecture using Time-Triggered Ethernet," in *Proc. SAE AeroTech*, Fort Worth, TX, USA, Sep. 2017.

[5] A. Loveless, "Notional 1FT Voting Architecture with Time-Triggered Ethernet," Houston, TX, USA, Nov. 2016. [Online]. Available: https://ntrs.nasa.gov/citations/20170001652

[6] J. T. Sims, "Redundancy Management Software Services for Seawolf Ship Control System," in *Proc, FTCS*, Seattle, WA, USA, Jun. 1997.

[7] J. H. Lala, "A Byzantine Resilient Fault Tolerant Computer for Nuclear Power Plant Applications," in *Proc. FTCS*, Vienna, Austria, Jul. 1986.

[8] W. Torres-Pomales, M. R. Malekpour, and P. S. Miner, "ROBUS-2: A Fault-Tolerant Broadcast Communication System," NASA Langley Research Center, Tech. Rep. NASA/TM-2005-213540, Mar. 2005. [Online]. Available: https://ntrs.nasa.gov/citations/20050158766

[9] J. H. Wensley, L. Lamport, J. Goldberg, M. W. Green, K. N. Levitt, P. M. Melliar-Smith, R. E. Shostak, and C. B. Weinstock, "SIFT: Design and Analysis of a Fault-tolerant Computer for Aircraft Control," *Proc. IEEE*, vol. 66, no. 10, pp. 1240–1255, Oct. 1978.

[10] W. Powell, "High-Performance Spaceflight Computing (HPSC) Project Overview," Phoenix, AZ, USA, Nov. 2018. [Online]. Available: https://ntrs.nasa.gov/citations/20180007636

[11] J. Hanaway and R. Moorehead, "Space Shuttle Avionics System," Tech. Rep. NASA SP-504, Jan. 1989. [Online]. Available: https://ntrs.nasa.gov/citations/19900015844

[12] R. Hammett, "Ultra-Reliable Real-Time Control Systems – Future Trends," in *Proc. DASC*, Bellevue, WA, USA, Oct. 1998.

[13] R. Hammett, M. Coakley, D. Sevigny, and R. Zamojski, "Automatic Performance Monitoring Enhances Seawolf Submarine Ship Control Maintainability," *Naval Engineers Journal*, vol. 110, no. 2, pp. 49–59, 1998.

[14] R. Hodson, Y. Chen, D. Morgan, M. Butler, J. Sdhuh, J. Petelle, D. Gwaltney, L. Coe, T. Koelbl, and H. Nguyen, "Heavy Lift Vehicle (HLV) Avionics Flight Computing Architecture Study," Tech. Rep. NASA/TM–2011-217168, Aug. 2011. [Online]. Available: https://ntrs.nasa.gov/citations/20110014793

[15] J. H. Lala and R. E. Harper, "Architectural Principles for Safety-Critical Real-Time Applications," *Proc. IEEE*, vol. 82, no. 1, pp. 25–40, Jan. 1994.

[16] T. Polsgrove, J. Chapman, S. Sutherlin, B. Taylor, L. Fabisinski, T. Collins, A. Dwyer Cianciolo, J. Samareh, E. Robertson, B. Studak, S. Vitalpur, A. Lee, and G. Rakow, "Human Mars Lander Design for NASA's Evolvable Mars Campaign," in *Proc. AeroConf*, Big Sky, MT, USA, Mar. 2016.

[17] R. E. Harper and J. H. Lala, "Fault-Tolerant Parallel Processor," *J. Guid. Control Dyn*, vol. 14, no. 3, pp. 554–563, May 1991.

[18] D. E. Stine, "Digital Signatures for a Byzantine Resilient Computer System," Master's thesis, Massachusetts Institute of Technology, 1995. [Online]. Available: http://dspace.mit.edu/handle/1721.1/36578

[19] M. Kapritsos, Y. Wang, V. Quema, A. Clement, L. Alvisi, and M. Dahlin, "All about Eve: Execute-Verify Replication for Multi-Core Servers," in *Proc. OSDI*, Hollywood, CA, USA, Oct. 2012.

[20] M. J. Fischer and N. A. Lynch, "A Lower Bound for the Time to Assure Interactive Consistency," *Inf. Process. Lett.*, vol. 14, pp. 183–186, 1982.

[21] A. Albert, "Comparison of Event-Triggered and Time-Triggered Concepts with Regard to Distributed Control Systems," *Embedded World*, vol. 171902, pp. 235–252, Jan. 2004.

[22] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong, "Zyzzyva: Speculative Byzantine Fault Tolerance," in *Proc. SOSP*, Stevenson, Washington, USA, Oct. 2007.

[23] J. Hendricks, S. Sinnamohideen, G. R. Ganger, and M. K. Reiter, "Zzyzx: Scalable fault Tolerance through Byzantine Locking," in *Proc. DSN*, Chicago, IL, USA, Jun. 2010.

[24] A. Singh, P. Fonseca, P. Kuznetsov, R. Rodrigues, and P. Maniatis, "Zeno: Eventually Consistent Byzantine-Fault Tolerance," in *Proc NSDI*, Boston, MA, USA, Apr. 2009.

[25] S. Duan, S. Peisert, and K. N. Levitt, "hBFT: Speculative Byzantine Fault Tolerance with Minimum Cost," *IEEE Trans. Dependable Secure Comput.*, vol. 12, no. 1, pp. 58–70, Jan. 2015.

[26] P.-L. Aublin, R. Guerraoui, N. Knežević, V. Quéma, and M. Vukolić, "The Next 700 BFT Protocols," *ACM Trans. Comput. Syst.*, vol. 32, no. 4, pp. 12:1–12:45, Jan. 2015.

[27] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukolić, S. W. Cocco, and J. Yellick, "Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains," in *Proc. EuroSys*, Porto, Portugal, Apr. 2018.

[28] T. M. Lovelly and A. D. George, "Comparative Analysis of Present and Future Space-Grade Processors with Device Metrics," *J. Aerosp. Inf. Syst.*, vol. 14, no. 3, pp. 184–197, 2017.

[29] D. McComas, "NASA/GSFC's Flight Software Core Flight System," San Antonio, TX, USA, Nov. 2012. [Online]. Available: https://ntrs.nasa.gov/citations/20130013412

[30] L. Prokop, "NASA's Core Flight Software - A Reusable Real-Time Framework," Houston, TX, USA, Nov. 2014. [Online]. Available: https://ntrs.nasa.gov/citations/20140017040

[31] J. Migge, J. Villanueva, N. Navet, and M. Boyer, "Insights on the Performance and Configuration of AVB and TSN in Automotive Ethernet Networks," in *Proc. ERTS*, San Jose, CA, USA, Jan. 2018.

[32] M. Fletcher, "Progression of an Open Architecture: From Orion to Altair and LSS," Honeywell, International, Tech. Rep. S65-5000-20-0, May 2009.

[33] "International Deep Space Interoperability Standards," National Aeronautics and Space Administration, Tech. Rep. Software Standard, Jun. 2019. [Online]. Available: https://www.internationaldeepspacestandards.com/

[34] R. Zurawski, *Industrial Communication Technology Handbook*. CRC Press, Aug. 2014.

[35] D. R. Kowalski and A. Mostéfaoui, "Synchronous Byzantine Agreement with Nearly a Cubic Number of Communication Bits," in *Proc. PODC*, Montreal, Canada, Jul. 2013.

[36] L. Lamport, R. Shostak, and M. Pease, "The Byzantine Generals Problem," *ACM Trans. Program. Lang. Syst.*, vol. 4, no. 3, Jul. 1982.

[37] A. Bar-Noy, D. Dolev, C. Dwork, and H. R. Strong, "Shifting Gears: Changing Algorithms on the Fly to Expedite Byzantine Agreement," in *Proc. PODC*, Vancouver, Canada, Aug. 1987.

[38] J.-L. Scharbarg and C. Fraboul, "Dimensioning of Civilian Avionics Networks," in *The Industrial Communication Technology Handbook*. CRC Press, 2014.

[39] N. Navet, J. Seyler, and J. Migge, "Timing Verification of Real-Time Automotive Ethernet Networks: What Can We Expect from Simulation?" in *Proc. SAE WCX*, Detroit, MI, Apr. 2015.

[40] A. Finzi and S. S. Craciunas, "Breaking vs. Solving: Analysis and Routing of Real-Time Networks with Cyclic Dependencies using Network Calculus," in *Proc. RTNS*, Toulouse, France, Nov. 2019.

[41] A. Loveless, "On TTEthernet for Integrated Fault-Tolerant Spacecraft Networks," in *Proc. AIAA SPACE*, Pasadena, CA, USA, Aug. 2015. [Online]. Available: https://ntrs.nasa.gov/citations/20170009923

[42] M. Boyer, H. Daigmorte, N. Navet, and J. Migge, "Performance Impact of the Interactions Between Time-Triggered and Rate-Constrained Transmissions in TTEthernet," in *Proc. ERTS*, Toulouse, France, Jan. 2016.

[43] "ARINC 664 P7: Aircraft Data Network Part 7 Avionics Full-Duplex Switched Ethernet Network," Aeronautical Radio, Incorporated (ARINC), Tech. Rep., Sep. 2009.

[44] P. Berman and J. A. Garay, "Efficient Distributed Consensus with N = (3 + Epsilon)T Processors (Extended Abstract)," in *Proc. WDAG*, Delphi, Greece, Oct. 1991.

[45] J. Garay and Y. Moses, "Fully Polynomial Byzantine Agreement for Processors in Rounds," *SIAM J. Comput*, vol. 27, no. 1, pp. 247–290, Feb. 1998.

[46] "Standard for Local and Metropolitan Area Networks-Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks Amendment: Enhancements for Scheduled Traffic," Institute of Electrical and Electronics Engineers, Tech. Rep. IEEE Std 802.1Qbv-2015, May 2016.

[47] "Road vehicles — FlexRay communications system — Part 1: General information and use case definition," International Organization for Standardization, Tech. Rep. ISO 17458-1:2013, Feb. 2013.

[48] "Time-Triggered Ethernet," SAE International, Tech. Rep. SAE AS6802, Nov. 2016.

[49] "TTP Communication Protocol," SAE International, Tech. Rep. SAE AS6003, Feb. 2011.

[50] A. Starke, D. Kumar, M. Ford, J. McNair, and A. Bell, "A Test Bed Study of Network Determinism for Heterogeneous Traffic Using Time-Triggered Ethernet," in *Proc. MILCOM*, Baltimore, MD, Oct. 2017.

[51] C. Wilwert, N. Navet, Y. Song, and F. Simonot-Lion, "Design of Automotive X-by-Wire Systems," in *The Industrial Communication Technology Handbook*. CRC Press, 2005.

[52] D. McComas, "Core Flight System (cFS) Background and Overview." [Online]. Available: https://cfs.gsfc.nasa.gov/cFS-OviewBGSlideDeck-ExportControl-Final.pdf

[53] L. E. Prokop, R. L. Hirsh, and C. Pagan, "Requirements-Based Execution Time Prediction of a Partitioned Real-Time System Using I/O and SLOC Estimates," *Innov. Syst. Softw. Eng.*, vol. 8, no. 4, pp. 309–320, Dec. 2012.

[54] "Core Flight System Scheduler (SCH) Application Design Review," Oct. 2019. [Online]. Available: https://github.com/nasa/SCH

[55] J.-B. Chaudron, D. Saussié, P. Siron, and M. Adelantado, "Real-Time Distributed Simulations in an HLA Framework: Application to Aircraft Simulation," *Simulation*, vol. 90, no. 6, pp. 627–643, Jun. 2014.

[56] M. Lauer, F. Boniol, C. Pagetti, and J. Ermont, "End-to-End Latency and Temporal Consistency Analysis in Networked Real-Time Systems," *International Journal of Critical Computer-Based Systems*, vol. 5, no. 3/4, pp. 172–196, Sep. 2014.

[57] V. Hadzilacos and S. Toueg, "A Modular Approach to Fault-Tolerant Broadcasts and Related Problems," Cornell University, Tech. Rep., May 1994.

[58] M. H. Azadmanesh and R. M. Kieckhafer, "Exploiting Omissive Faults in Synchronous Approximate Agreement," *IEEE Trans. Comput*, vol. 49, no. 10, pp. 1031–1042, Oct. 2000.

[59] J. Brown and B. Martin, "How Fast Is Fast Enough? Choosing Between Xenomai and Linux for Real-Time Applications," in *Proc. Real-Time Linux Workshop*, Nairobi, Kenya, Oct. 2010.

[60] B. Ip, "Performance Analysis of VxWorks and RTLinux," Columbia University, Tech. Rep., 2001.

[61] S. Parkes, C. McClements, D. McLaren, A. F. Florit, and A. G. Villafranca, "SpaceFibre Networks: SpaceFibre, Long Paper," in *Proc. International SpaceWire Conference*, Yokohama, Japan, Oct. 2016.

[62] R. Makowitz and C. Temple, "Flexray - A Communication Network for Automotive Control Systems," in *Proc. WFCS*, Torino, Italy, Jun. 2006.

[63] Y. Zeng and G. Chen, "Research on Methods to Improve Precise Time Synchronization for IRIG-B Code Encoder," in *Proc. APPEEC*, Shanghai, China, Mar. 2012.

[64] K. Hoyme and K. Driscoll, "SAFEbus," in *Proc. DASC*, Seattle, WA, USA, Oct. 1992.

[65] G. Bracha and S. Toueg, "Resilient Consensus Protocols," in *Proc. PODC*, Montreal, Canada, Aug. 1983.

[66] D. Chi-Shing Chau, "Authenticated Messages for a Real-Time Fault-Tolerant Computer System," Master's thesis, Massachusetts Institute of Technology, 2006. [Online]. Available: http://dspace.mit.edu/handle/1721.1/36909

[67] M. Castro and B. Liskov, "Practical Byzantine Fault Tolerance," in *Proc. OSDI*, Berkeley, CA, USA, Feb. 1999.

[68] B. Bhadviya, "Testing Byzantine Faults," University of Michigan, Tech. Rep., 2010.

[69] P. Khanchandani and C. Lenzen, "Self-Stabilizing Byzantine Clock Synchronization with Optimal Precision," *Theory Comput. Syst.*, vol. 63, no. 2, pp. 261–305, Feb. 2019.

[70] K. Driscoll, B. Hall, M. Paulitsch, P. Zumsteg, and H. Sivencrona, "The Real Byzantine Generals," in *Proc. DASC*, Salt Lake City, UT, USA, Oct. 2004.

[71] R. Obermaisser, *Time-Triggered Communication*. CRC Press, 2011.

[72] "Data Network Evaluation Criteria Handbook," Federal Aviation Administration, Tech. Rep. DOT/FAA/AR-09/24, Jun. 2009.

[73] M. Paulitsch, J. Morris, B. Hall, K. Driscoll, E. Latronico, and P. Koopman, "Coverage and the Use of Cyclic Redundancy Codes in Ultra-Dependable Systems," in *Proc. DSN*, Yokohama, Japan, Jun. 2005.

[74] K. Driscoll, B. Hall, H. Sivencrona, and P. Zumsteg, "Byzantine Fault Tolerance, From Theory to Reality," in *Proc. SAFECOMP*, Edinburgh, Scotland, Sep. 2003.

[75] L. Gong, P. Lincoln, and J. Rushby, "Byzantine Agreement with Authentication: Observations and Applications in Tolerating Hybrid and Link Faults," *Dependable Computing for Critical Applications*, vol. 10, pp. 139–157, Sep. 1995.

[76] R. Turpin and B. A. Coan, "Extending Binary Byzantine Agreement to Multivalued Byzantine Agreement," *Inf. Process. Lett.*, vol. 18, no. 2, pp. 73–76, Feb. 1984.

[77] A. Loveless, R. Dreslinski, and B. Kasikci, "Optimal and Error-Free Multi-Valued Byzantine Consensus Through Parallel Execution," Tech. Rep., Mar. 2020. [Online]. Available: https://eprint.iacr.org/2020/322

[78] C. Ganesh and A. Patra, "Optimal Extension Protocols for Byzantine Broadcast and Agreement," Tech. Rep., Mar. 2017. [Online]. Available: https://eprint.iacr.org/2017/063

[79] A. Patra, "Error-free Multi-valued Broadcast and Byzantine Agreement with Optimal Communication Complexity," in *Proc. OPODIS*, Toulouse, France, Dec. 2011.

[80] P. Berman, J. A. Garay, and K. J. Perry, "Bit Optimal Distributed Consensus," *Computer Science: Research and Applications*, pp. 313–321, 1992.

[81] B. A. Coan and J. L. Welch, "Modular Construction of a Byzantine Agreement Protocol with Optimal Message Bit Complexity," *Inf. Comput.*, vol. 97, no. 1, pp. 61–85, Mar. 1992.

[82] Y. Xiao, N. Zhang, J. Li, W. Lou, and Y. T. Hou, "Distributed Consensus Protocols and Algorithms," in *Blockchain for Distributed Systems Security*. Wiley, 2019, pp. 25–50.

[83] C. Spitzer, *The Avionics Handbook*. CRC Press, 2001.

[84] "International Deep Space Interoperability Standards," National Aeronautics and Space Administration, Tech. Rep. Avionics Standard, Mar. 2019. [Online]. Available: https://www.internationaldeepspacestandards.com/

[85] C. M. Fuchs, "The Evolution of Avionics Networks From ARINC 429 to AFDX," in *Proc. Seminars Future Internet, Innovative Internet Technologies and Mobile Communication, and Aerospace Networks*, Munich, Germany, 2013.

[86] Y. Moses and O. Waarts, "Coordinated Traversal: (t+1)-round Byzantine Agreement in Polynomial Time," in *Proc. FOCS*, White Plains, NY, USA, Oct. 1988.

[87] J. A. Garay and Y. Moses, "Fully Polynomial Byzantine Agreement in t + 1 Rounds," in *Proc. STOC*, San Diego, CA, USA, May 1993.

[88] C. Ganesh and A. Patra, "Broadcast Extensions with Optimal Communication and Round Complexity," in *Proc. PODC*, Chicago, IL, USA, Jul. 2016.

[89] K. Nayak, L. Ren, E. Shi, N. H. Vaidya, and Z. Xiang, "Improved Extension Protocols for Byzantine Broadcast and Agreement," Tech. Rep., Feb. 2020. [Online]. Available: https://arxiv.org/abs/2002.11321v1

[90] M. Y. Tang, "Wireless Reconfigurability of Fault-Tolerant Processing Systems," Master's thesis, Massachusetts Institute of Technology, Sep. 2008. [Online]. Available: https://dspace.mit.edu/handle/1721.1/53168

[91] T. P. Baker and A. Shaw, "The Cyclic Executive Model and Ada," *Real-Time Systems*, vol. 1, no. 1, pp. 7–25, Jun. 1989.

[92] D. R. K. Ports, J. Li, V. Liu, N. K. Sharma, and A. Krishnamurthy, "Designing Distributed Systems Using Approximate Synchrony in Data Center Networks," in *Proc. NSDI*, Oakland, CA, USA, May 2015.

[93] P. J. Marandi, M. Primi, and F. Pedone, "High Performance State-Machine Replication," in *Proc. DSN*, Hong Kong, Jun. 2011.

[94] R. Jimenez-Peris, M. Patino-Martinez, B. Kemme, and G. Alonso, "Improving the Scalability of Fault-Tolerant Database Clusters," in *Proc. ICDCS*, Vienna, Austria, Jul. 2002.

[95] B. Kemme, F. Pedone, G. Alonso, and A. Schiper, "Processing Transactions Over Optimistic Atomic Broadcast Protocols," in *Proc. ICDCS*, Austin, Texas, USA, Jun. 1999.

[96] F. Pedone and A. Schiper, "Optimistic Atomic Broadcast," in *Proc. DISC*, Andros, Greece, Sep. 1998.

[97] C. E. Bezerra, F. Pedone, R. van Renesse, and C. Geyer, "Providing Scalability and Low Latency in State Machine Replication," Università della Svizzera Italiana, Lugano, Switzerland, Tech. Rep. USI-INF-TR-2015/06, Dec. 2015.

[98] O. M. Mendizabal, P. Jalili Marandi, F. L. Dotti, and F. Pedone, "Checkpointing in Parallel State-Machine Replication," in *Proc. OPODIS*, Cortina d'Ampezzo, Italy, Dec. 2014.

[99] P. J. Marandi, C. E. Bezerra, and F. Pedone, "Rethinking State-Machine Replication for Parallelism," in *Proc. ICDCS*, Madrid, Spain, Jun. 2014.

[100] R. Kotla and M. Dahlin, "High Throughput Byzantine Fault Tolerance," in *Proc. DSN*, Florence, Italy, Jun. 2004.

[101] S. Hirve, R. Palmieri, and B. Ravindran, "Archie: A Speculative Replicated Transactional System," in *Proc. Middleware*, Bordeaux, France, Dec. 2014.

[102] N. Santos and A. Schiper, "Achieving High-Throughput State Machine Replication in Multi-core Systems," in *Proc. ICDCS*, Philadelphia, Pennsylvania, USA, Jul. 2013.

[103] Z. Guo, C. Hong, M. Yang, D. Zhou, L. Zhou, and L. Zhuang, "Rex: Replication at the Speed of Multi-Core," in *Proc. EuroSys*, Amsterdam, The Netherlands, Apr. 2014.

[104] M. Fitzi and M. Hirt, "Optimally Efficient Multi-Valued Byzantine Agreement," in *Proc PODC*, Denver, CO, USA, Jul. 2006.

[105] Z. Beerliová-Trubíniová and M. Hirt, "Perfectly-Secure MPC with Linear Communication Complexity," in *Proc. TCC*, New York, NY, USA, 2008.

[106] G. Liang, B. Sommer, and N. Vaidya, "Experimental Performance Comparison of Byzantine Fault-Tolerant Protocols for Data Centers," in *Proc. INFOCOM*, Orlando, FL, USA, Mar. 2012.

[107] G. Liang and N. Vaidya, "Error-Free Multi-Valued Consensus with Byzantine Failures," in *Proc. PODC*, San Jose, CA, USA, Jun. 2011.

[108] A. Patra and C. P. Rangan, "Communication Optimal Multi-valued Asynchronous Broadcast Protocol," in *Proc. LATINCRYPT*, Puebla, Mexico, Aug. 2010.

[109] A. Patra and C. Rangan, "Communication Optimal Multi-valued Asynchronous Byzantine Agreement with Optimal Resilience." Tech. Rep., Jan. 2011. [Online]. Available: https://eprint.iacr.org/2009/433.pdf

[110] L. Tseng and N. Vaidya, "Byzantine Broadcast Under a Selective Broadcast Model for Single-hop Wireless Networks," Tech. Rep., Jan. 2015. [Online]. Available: https://arxiv.org/abs/1502.00075

[111] A. Choudhury, "Multi-Valued Asynchronous Reliable Broadcast with a Strict Honest Majority," in *Proc. ICDCN*, Hyderabad, India, Jan. 2017.

[112] M. Hirt and P. Raykov, "Multi-valued Byzantine Broadcast: The t<n Case," in *Proc. ASIACRYPT*, Kaoshiung, Taiwan, Dec. 2014.

[113] W. Chongchitmate and R. Ostrovsky, "Information-Theoretic Broadcast with Dishonest Majority for Long Messages," in *Proc. TCC*, Goa, India, Nov. 2018.

[114] B.-G. Chun, P. Maniatis, S. Shenker, and J. Kubiatowicz, "Attested Append-only Memory: Making Adversaries Stick to Their Word," in *Proc. SOSP*, Stevenson, WA, USA, Oct. 2007.

[115] A. Clement, F. Junqueira, A. Kate, and R. Rodrigues, "On the (Limited) Power of Non-Equivocation," in *Proc. PODC*, Madeira, Portugal, Jul. 2012.

[116] M. Correia, G. S. Veronese, and L. C. Lung, "Asynchronous Byzantine Consensus with 2F+1 Processes," in *Proc. SAC*, Sierre, Switzerland, 2010.

[117] R. Kapitza, J. Behl, C. Cachin, T. Distler, S. Kuhnle, S. V. Mohammadi, W. Schröder-Preikschat, and K. Stengel, "CheapBFT: Resource-efficient Byzantine Fault Tolerance," in *Proc. EuroSys*, Bern, Switzerland, Apr. 2012.

[118] D. Levin, J. R. Douceur, J. R. Lorch, and T. Moscibroda, "TrInc: Small Trusted Hardware for Large Distributed Systems," in *Proc. NSDI*, Boston, MA, USA, Apr. 2009.

[119] M. Backes, F. Bendun, A. Choudhury, and A. Kate, "Asynchronous MPC with a Strict Honest Majority Using Non-Equivocation," in *Proc. PODC*, Paris, France, Jul. 2014.

[120] D. Dolev, "The Byzantine Generals Strike Again," *Algorithms*, vol. 3, no. 1, pp. 14–30, Mar. 1982.

[121] S. Toueg, "Randomized Byzantine Agreements," in *Proc. PODC*, Vancouver, Canada, Aug. 1984.

[122] A. Mostéfaoui and M. Raynal, "Signature-Free Broadcast-Based Intrusion Tolerance: Never Decide a Byzantine Value," in *Proc. OPODIS*, Tozeur, Tunisia, Dec. 2010.

[123] C. Cachin, K. Kursawe, F. Petzold, and V. Shoup, "Secure and Efficient Asynchronous Broadcast Protocols," in *Proc. CRYPTO*, Santa Barbara, CA, USA, Aug. 2001.