Real-Time Replica Consistency over Ethernet with Reliability Bounds

Arpan Gujarati, Sergey Bozhko, and Björn B. Brandenburg
Max Planck Institute for Software Systems (MPI-SWS)

***Abstract*—Ethernet is expected to play a key role in the development of the next generation of safety-critical distributed real-time systems. Unfortunately, the use of switched Ethernet in place of traditional field buses such as CAN exposes systems to the risk of *Byzantine* errors (or inconsistent broadcasts) due to environmentally-induced transient faults.**

Byzantine fault tolerance (BFT) protocols can mitigate such errors to a large extent. However, no BFT protocol has yet been investigated from the perspective of hard real-time predictability. Classical Byzantine safety guarantees (*e.g.*, $3f + 1$ processes can tolerate up to f Byzantine faults) are oblivious to non-uniform fault rates across different system components that arise due to environmental disturbances. Furthermore, existing analyses abstract from the underlying network topology despite its strong influence on actual failure rates.

In this work, we present (i) a hard real-time interactive consistency protocol that allows distributed processes to agree on a common state despite Byzantine errors; and (ii) the first quantitative, real-time-aware reliability analysis of such a protocol deployed over switched Ethernet in the presence of stochastic transient faults. Our analysis is free of *reliability anomalies* and, as we show in our evaluation, can be used for a reliability-aware design space exploration of different fault tolerance alternatives.

I. Introduction

To meet safety certification requirements, commercial aircraft are designed to achieve *quantifiably negligible* failure rates [1]. In particular, they are designed to tolerate not only software errors and manufacturing defects, but also errors due to environmentally-induced transient faults. For example, when Hopkins *et al.* [2] designed one of the first highly reliable fault-tolerant multiprocessors for aircraft control systems, they validated analytically that its expected failure rate in the presence of stochastic faults does not exceed 10^{-10} failures per hour.

In the future, aircraft-like highly reliable designs, which are (in)famously expensive [3], will become necessary in other, much more cost-sensitive, CPS domains too. Recent trends suggest that the cumulative operating time of safety-critical CPS in other domains (*e.g.*, autonomous vehicles, drones, and robots) will likely exceed that of aircraft [4]; commodity Ethernet technology [5] will play a more important role due to its low cost and high bandwidth [6, 7]; and strong economic and time-to-market pressures will further drive the adoption of cheap, but relatively unreliable, *commercial off-the-shelf* (COTS) processors. As a result, to guarantee a minimum level of safety across fleets of next-generation CPS, it will be necessary to tolerate errors due to environmentally-induced transient faults at runtime, as well as to *a priori* quantify the expected residual failure rates for better resource provisioning.

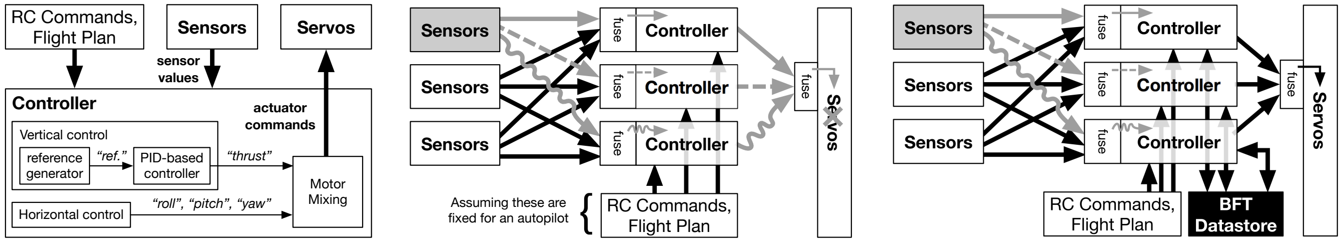
In this work, we focus on tolerating environmentally-induced *Byzantine* errors (*i.e.*, inconsistent broadcasts) in Ethernet-based distributed real-time systems. Non-malicious Byzantine errors pose a significant risk in practice, and have been shown to occur due to electro-magnetic interference and other environmental disturbance sources (such as thermal effects) [8].

Prior work on distributed real-time systems has proposed reliability analyses of active-replication protocols [9], network retransmission protocols [10], OS-level fault tolerance mechanisms [11], *etc.*, but none of these address Byzantine errors. Prior work on Byzantine fault tolerance (BFT) protocols for cloud computing systems does not address the predictability concerns of real-time systems [12]. In addition, classical Byzantine safety guarantees (*e.g.*, $3f + 1$ processes can tolerate up to f Byzantine faults) do not take into account non-uniform fault rates across different system components that arise due to environmental disturbances. They also abstract from the underlying network topology despite its strong influence on actual failure rates (as shown in §V).

To address this gap, we present the first quantitative reliability analysis of a hard real-time *interactive consistency (IC)* protocol over Ethernet in the presence of stochastic transient faults. The protocol allows distributed processes to agree on a common state despite Byzantine errors. Specifically, we analyze a *parameterized* (more generic) version of the classic IC protocol by Pease *et al.* [13] for synchronous networks (§II).

Our work is motivated by next-generation fault-tolerant architectures for safety-critical autonomous CPS, which will require hard real-time fault-tolerant algorithms for replica coordination. For example, Fig. 1(a) illustrates an overview of the legacy drone autopilot system in Paparazzi [14]. An enhanced design with active replication (as shown in Fig. 1(b)) tolerates fault-induced crash errors, but is still susceptible to Byzantine errors. Going a step further, if the active replicas would instead synchronize their local state using a BFT datastore (as shown in Fig. 1(c)), the actuation commands would be protected from Byzantine errors (with very high likelihood). The IC protocol considered and analyzed in this paper would be a suitable choice for implementing the core functionality of such a hard real-time BFT datastore.

The remainder of the paper is organized as follows. In §III, we first propose a blueprint for a hard real-time adaptation of Pease *et al.*'s IC protocol [13] based on Liu and Layland's periodic task model [15] and Ethernet's prioritized traffic classes [5]. Thereafter, we propose a multi-layered analysis (§IV) where we **(i)** abstract the effect of *basic errors* (such



(a) Paparazzi Overview

(b) Active replication vulnerable to Byzantine errors

(c) Active replication with BFT datastore

Fig. 1. Motivating example. **(a)** Overview of the legacy drone autopilot system in Paparazzi [14]. **(b)** An enhanced design with active replication safeguards against crash faults but does not tolerate Byzantine errors. The gray arrows illustrate an example Byzantine error scenario. A sensor replica is faulty and sends inconsistent values to the controller replicas. As a result, the output of the fuse function in each controller replica may differ, causing the replica states to diverge over time. **(c)** Instead, the active replicas can be periodically coordinated using a BFT datastore, which significantly reduces the risk of Byzantine errors. The contribution of this paper is the first quantitative, real-time-aware reliability analysis of an IC protocol that can be used to realize such a BFT data store.

as crashes) into different types of *protocol-specific message errors* (such as crash-induced message omissions); **(ii)** derive system-specific upper bounds on the probabilities with which these protocol-specific message errors occur; and **(iii)** upper-bound the overall protocol failure probability by exhaustively evaluating all scenarios in which one or more protocol-specific message errors result in a failed execution of the IC protocol. Unlike simulations and probabilistic model checking [16], our analysis is free from *reliability anomalies*, which can result in non-monotonic increases in the system's overall failure rate despite local decreases in a component's failure rate.

In our evaluation (§V), we show that the analytical bounds closely predict the failure rates observed in simulation. We also report on a case study in which we assessed reliability gains for different configurations of the IC protocol and for different network topologies. Our analysis was successful in revealing non-trivial reliability trade-offs, such as the significant impact of the number of message exchange rounds and the network topology on the overall reliability of the IC protocol.

II. System Model

Consider a distributed system consisting of N_p processes $\Pi = \{\Pi_1, \Pi_2, \ldots, \Pi_{N_p}\}$ deployed on N_p independent hosts $H = \{E_1, E_2, \ldots, E_{N_p}\}$, respectively. The hosts are connected using N_s ethernet switches $S = \{S_1, S_2, \ldots, S_{N_s}\}$ and N_l links $L = \{L_1, L_2, \ldots, L_{N_l}\}$. Messages from Π_i to Π_k are transmitted through a statically configured route denoted $route_{i,k}$ and defined as an alternating sequence of links and switches, *e.g.*, $route_{i,k} = \langle L_1 S_1 L_2 S_2 L_3 \rangle$ is a 3-hop route.

A. Interactive Consistency Protocol

The IC problem for this system is defined as follows. Each process Π_i has a local state v_i and seeks to compute a vector V_i such that, for $1 \leq k \leq N_p$, item $V_i[k]$ corresponds to the local state of process Π_k. The objective is to ensure that $V_i[k] = V_j[k]$ for any two correct processes $\Pi_i, \Pi_j \in \Pi$, and if process Π_k is also correct, then $V_i[k] = V_j[k] = v_k$.

To solve the IC problem, we consider the classic BFT protocol by Pease *et al.* [13]. The protocol was designed to solve the IC problem in a *synchronous* manner (*i.e.*, using a finite number of communication and processing rounds), and is thus a good match for real-time systems. That is, as we show in §III, it can easily be translated into a real-time implementation.

We actually analyze a generalized version of the IC protocol. Unlike Pease *et al.*, we do not upper-bound the number of faulty processes beforehand and, conversely, also do not lower-bound the number of message exchange rounds. Instead, we parameterize the protocol in terms of an arbitrary number of participating processes N_p and protocol rounds N_r. The reason for this generalization is that, in the presence of environmentally induced transient faults, each process may behave erroneously at different times with non-zero probability. Therefore, depending on the program-visible effects of transient faults, additional protocol rounds or processes do not always increase the chances of solving the IC problem successfully.

The precise IC protocol executed by each process Π_i is explained next (see Algorithm 1 for pseudocode). Process Π_i gathers all received information in the form of a tree, called the *Exponential Information Gathering* (EIG) tree [17], and denoted EIG_i. Each node in EIG_i is a $\langle label, value \rangle$ pair (denoted as $\langle \alpha, v \rangle$ in Algorithm 1), where the label is an ordered sequence of one or more process identities. In the beginning (Line 2), EIG_i is initialized with the root node $\langle \epsilon, v_i \rangle$, where ϵ denotes an empty label. In each of the N_r rounds thereafter, Π_i executes up to three steps.

During the *sending step* in round r (Lines 4–6), Π_i sends to other processes all nodes in the $(r-1)^{\text{st}}$ level of its tree (*i.e.*, nodes with label size $|\alpha| = r - 1$). However, there are two exceptions to this rule. Nodes with value $\perp$ are not sent because they correspond to omitted messages (as explained later), and nodes whose labels contain Π_i are also not sent to avoid cycles in the EIG tree labels.

The next step is the *state transition step* during which Π_i updates its EIG tree based on the received messages (Lines 7–15). In particular, during round r, for every level-$(r-1)$ node $\langle \alpha, v \rangle$ in EIG_i (*i.e.*, for every node with $|\alpha| = r - 1$), Π_i expects other processes to send their corresponding level-$(r-1)$ nodes. If Π_i indeed receives a message of the form $\langle \alpha, v' \rangle$ from another process Π_j, it adds the pair $\langle \alpha\Pi_j, v' \rangle$ as a child of node $\langle \alpha, v \rangle$; if Π_i does not receive such a message, it adds a dummy pair $\langle \alpha\Pi_j, \perp \rangle$ to register an error-induced omission,

Algorithm 1 Π_i's version of the IC protocol.

```
 1: procedure INITIALIZATION
 2:     $EIG_i.addRoot(\langle \epsilon, v_i \rangle)$
 3: procedure ROUND($r$)
 4:     for all $\langle \alpha, v \rangle \in EIG_i.nodes$ s.t. $|\alpha| = r - 1$ do        ▷ sending step
 5:         if $\Pi_i \notin \alpha \wedge v \neq \bot$ then
 6:             send $\langle \alpha, v \rangle$ to all processes in $\Pi \setminus \{\Pi_i\}$
 7:     for all $\langle \alpha, v \rangle \in EIG_i.nodes$ do        ▷ state transition step
 8:         if $|\alpha| = r - 1$ then
 9:             for all $\Pi_j \in \Pi$ s.t. $\Pi_j \notin \alpha$ do
10:                 if $\Pi_i = \Pi_j$ then
11:                     $EIG_i.addChild(\langle \alpha, v \rangle, \langle \alpha\Pi_j, v \rangle)$
12:                 else if $\langle \alpha, v' \rangle$ is received from $\Pi_j$ then
13:                     $EIG_i.addChild(\langle \alpha, v \rangle, \langle \alpha\Pi_j, v' \rangle)$
14:                 else
15:                     $EIG_i.addChild(\langle \alpha, v \rangle, \langle \alpha\Pi_j, \bot \rangle)$
16:     if $r \neq N_r$ then return        ▷ reduction step
17:     for all $\langle \alpha, v \rangle \in EIG_i.nodes$ from $|\alpha| = N_r - 1$ to $|\alpha| = 1$ do
18:         $candidates = \emptyset$, $v_{majority} = \bot$
19:         for all $\langle \alpha\Pi_j, v' \rangle \in EIG_i.getChildren(\langle \alpha, v \rangle)$ do
20:             if $v' \neq \bot$ then
21:                 $candidates = candidates \cup \{v'\}$
22:         if $candidates \neq \emptyset$ then
23:             $v_{majority} = simpleMajority(candidates)$
24:         $EIG_i.updateValue(\langle \alpha, v \rangle, v_{majority})$
25:         if $\alpha = \Pi_k$ then        ▷ if level-1 node
26:             $V_i[k] \leftarrow v_{majority}$        ▷ update decision vector
```

i.e., any message received past its deadline is discarded. Π_i does not expect a message from Π_j if $\Pi_j \in \alpha$ (Line 9), since the sending step avoids cycles in the EIG tree labels.

Information gathering as described above goes on for $N_r \leq N_p$ rounds. In the last round, during the *reduction step*, each sub-tree of EIG_i's root node is recursively reduced (Lines 17–26). If any node $\langle \alpha, v \rangle$ is a leaf node (*i.e.*, $|\alpha| = N_r$), its value does not change; otherwise, if $v_{majority}$ denotes the majority among the values of node $\langle \alpha, v \rangle$'s children, $\langle \alpha, v \rangle$ is updated to $\langle \alpha, v_{majority} \rangle$. (Line 24). The decision vector V_i is finally determined by the level-1 nodes (Line 26).

B. Time-Aware Correctness Criteria

An embedded application may use the IC protocol to achieve *input consistency* over redundant sensor values, where the processes may *fuse* their respective decision vectors using a noise filtering function and forward the results to an actuator, which in turn may use a simple majority hardware for redundancy suppression. In this case, application reliability depends on the IC protocol execution as well as on the fuse function used by the processes. In general, every application may rely on a different set of correctness criteria requiring a slightly different set of reliability analyses. We define below two correctness criteria that form the basis of our analysis.

1) Strong Correctness: We assume that every process Π_i uses *quorum majority* as its fuse function, which we denote as f_q. That is, $f_q(V_i)$ returns either the value that equals at least $\lfloor N_p/2 \rfloor + 1$ elements in V_i, or $\bot$ (if no such value exists).

With a focus on real-time applications, we also assume that $f_q(V_i) = \bot$ if Π_i fails to produce V_i on time. This is possible if environmentally-induced faults delay the execution of IC protocol steps (*i.e.*, deadline misses in the hard real-time realization of the IC protocol, which is provided in §III).

Suppose that at the end of an error-free execution of the IC protocol, $f_q(V_i) = f_{correct} \neq \bot$. Let Π be partitioned into:

$$\mathcal{S}_{correct} = \{\Pi_i \in \Pi \mid f_q(V_i) = f_{correct}\},$$
$$\mathcal{S}_{skipped} = \{\Pi_i \in \Pi \mid f_q(V_i) = \bot\}, \text{ and}$$
$$\mathcal{S}_{faulty} = \{\Pi_i \in \Pi \mid f_q(V_i) \neq f_{correct} \wedge f_q(V_i) \neq \bot\}.$$

The strong correctness criterion requires that $|\mathcal{S}_{correct}| \geq \lfloor N_p/2 \rfloor + 1$. This criterion resembles the guarantees offered by traditional BFT protocols for general-purpose systems.

2) Weak Correctness: We assume that each Π_i uses *simple majority* as its fuse function, denoted as g_s. The simple majority function $g_s(V_i)$ breaks ties deterministically using process IDs, and returns $\bot$ only if all values in V_i are $\bot$ or if Π_i failed to produce vector V_i on time. Once again, suppose that $g_s(V_i) = g_{correct} \neq \bot$ denotes the output at the end of an error-free execution of the IC protocol. Let Π be partitioned into:

$$\mathcal{W}_{correct} = \{\Pi_i \in \Pi \mid g_s(V_i) = g_{correct}\},$$
$$\mathcal{W}_{skipped} = \{\Pi_i \in \Pi \mid g_s(V_i) = \bot\}, \text{ and}$$
$$\mathcal{W}_{faulty} = \{\Pi_i \in \Pi \mid g_s(V_i) \neq g_{correct} \wedge g_s(V_i) \neq \bot\}.$$

The weak correctness criterion requires that $|\mathcal{W}_{correct}| > |\mathcal{W}_{faulty}|$. It is particularly useful for embedded applications which are not concerned if redundant outputs are skipped, as long as at least one correct output is delivered on time. For example, the weak correctness criterion is ideal for the embedded application mentioned above that relies on a simple majority hardware for redundancy suppression.

C. Basic Errors Modeling

We classify the basic errors due to environmentally-induced transient faults into three broad categories: node crashes (*i.e.*, system reboots), memory corruption in nodes, and frame corruption on network links (where *nodes* refers to both compute hosts and network switches).

Crash errors occur if the system suffers a fault-induced corruption that causes the system to be rebooted, or that induces an unbounded hang that causes the system's watchdog timer to trigger a reboot, *e.g.*, see [18]. A crashed system remains unavailable for some time while it reboots and thus causes an interval in which messages are continuously omitted. We assume that the recovery interval of each host E_i and switch S_i is upper-bounded by $\Delta_{reboot}(E_i)$ and $\Delta_{reboot}(S_i)$, respectively, and that any messages queued in a switch are lost upon a crash.

Memory corruption errors occur if the system suffers a fault-induced corruption that does not result in a system reboot, but ends up corrupting one or more bits of the system's memory. The *exposure interval* of a task during which it may be affected by memory corruption errors depends on whether the task is stateful or stateless, and on the memory protection mechanisms in use [19]. With Error-Correcting Code (ECC) memory and lockstep processors (common in safety-critical systems), latent faults are suppressed, and it suffices to consider the *scheduling window* of a message (*i.e.*, the duration from the message's creation to its deadline) as its exposure interval. If no such architectural support is available, then any relevant state can be protected with a *data integrity checker* task that periodically verifies the checksums of all relevant data structures (and that reboots the system in the case of any mismatch). The exposure interval of a message then includes its scheduling window and (in the worst case) an entire period of the data integrity checker. We assume that process states are checked at least once between consecutive activations of the protocol. Thus, an IC protocol task cannot be affected by memory corruptions that occur prior to the end of the previous protocol instance.

Frame corruption errors result in the corruption of message frames during transmission. They occur only if the corruption is not caught by the CRC at the Ethernet layer. In contrast, if the corruption is detected at the Ethernet layer, the frame is discarded, resulting in a *frame omission error*.

Based on prior work, we model the occurrence of the aforementioned basic errors in any given interval of time using Poisson distributions.[1] As per this model, the probability that x instances of any basic error type *err* affect any component *comp* in any interval of length δ is defined as

$$\mathcal{P}(x, \delta, \gamma_{err}(comp)) = \frac{e^{-\delta \cdot \gamma_{err}(comp)} \cdot (\delta \cdot \gamma_{err}(comp))^x}{x!},$$

where $\gamma_{err}(comp)$ denotes the peak error rate. $\gamma_{err}(comp)$ is specified in advance, taking into account the maximum expected rate of environmental interference (including safety margins) as deemed appropriate by reliability engineers or domain experts. For example, $\gamma_{err}(comp)$ can be empirically determined with measurements, or derived from environmental modeling assuming worst-possible operating conditions as well as *derating factors* [22] that account for masked transient faults.

Based on the stochastic nature of transient faults, we consider these basic error events, and their corresponding Poisson arrival processes, to be independent. We do however account explicitly for all correlations that arise from the protocol structure. For example, messages in a later round being incorrectly computed due to corruptions in an earlier round or the loss of a batched payload is explicitly accounted for in our analysis.

III. Hard Real-Time Realization

The IC protocol described in §II-A can be realized in many ways. However, a hard real-time implementation is most beneficial for safety-certification and typically expected when building safety-critical CPS. Moreover, hard real-time predictability is where prior literature on Byzantine fault tolerance falls short, which is why it is important to sketch a design that is certainly analyzable. Hence, we map the IC protocol to Liu and Layland's periodic task model [15], which has been widely studied in the real-time systems community and which therefore provides a solid foundation for temporal certification.

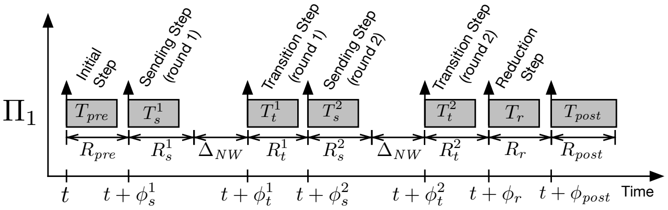


Fig. 2. Periodic tasks as part of process Π_1; the IC protocol starts at time t.

In particular, we propose a design where the execution of the IC protocol by each process Π_i is modeled using multiple periodic tasks deployed on the respective host H_i. The proposed design depends on two assumptions. First, we assume that hosts have synchronized clocks, which can be ensured on commodity platforms using clock synchronization protocols such as the widely used Precision Time Protocol (PTP) [23]. Second, we assume that worst-case network latency is predictable, which can be ensured using Ethernet's Time Sensitive Networking (TSN) standard [5] (or similar other protocols). In the following, we present the detailed task model, which is also illustrated in Fig. 2. Since the IC protocol is symmetric for all processes, identical task sets are deployed on each host; therefore, we omit the process index i from the notations to reduce clutter.

We realize each process by a set of tasks $T_s = \{T_s^1, T_s^2, \ldots, T_s^{N_r}\}$ that execute the N_r sending steps (respectively), a set of tasks $T_t = \{T_t^1, T_t^2, \ldots, T_t^{N_r}\}$ that execute the N_r state transition steps (respectively), and a task T_r that executes the reduction step in the end. Additionally, we model tasks T_{pre} and T_{post} that execute at the beginning and end of the protocol, respectively, and which interface the IC protocol with the application. T_{pre} is also responsible for initializing the EIG tree. We assume that the IC protocol is invoked periodically with time period P, *i.e.*, a new protocol instance with the objective of achieving interactive consistency over a new set of values is initiated every P time units. Hence, all tasks are assigned a time period of P, and each new activation of the task set corresponds to a new IC protocol instance.

To ensure that the tasks are activated in the order required by the IC protocol, each task is also assigned an appropriate *release offset*. Task T_{pre}, which is expected to execute before any other IC protocol tasks, is released periodically with release offset 0 on each host, *i.e.*, T_{pre} becomes ready for execution at time instants 0, P, $2P$, and so on. Suppose that R_{pre} denotes the *global worst-case response time* of T_{pre} across all hosts, *i.e.*, the periodic invocations of T_{pre} on all hosts finish their executions at the latest by time instants R_{pre}, $P + R_{pre}$, $2P + R_{pre}$, and so on, respectively (we omit differences due to clock skew in these absolute time instants to avoid clutter). As per Algorithm 1, task T_s^1, which is responsible for executing the sending step of round one, must follow task T_{pre}. Thus, T_s^1 is assigned a release offset of $\phi_s^1 = R_{pre}$, *i.e.*, T_s^1 becomes ready for execution at time instants R_{pre}, $P + R_{pre}$, $2P + R_{pre}$, and so on. The next step as per Algorithm 1 is the state transition step of round one, which is executed by task T_t^1. Task T_t^1

[1] The applicability of the Poisson distribution for modeling environmentally-induced errors is discussed in detail in prior work [10, 20]. In a nutshell, when higher mean fault rates obtained during periods of high environmental interference are used, a Poisson distribution models transient faults sufficiently well. In addition, a Poisson distribution is also applicable to fault-induced errors since real-time tasks are repeated, short workloads, and hence each job of a task is equally likely to suffer a fault-induced error [21].

must also wait for the messages sent during the preceding sending step to be transmitted. Thus, task T_t^1 is assigned a release offset of $\phi_t^1 = \phi_s^1 + R_s^1 + \Delta_{NW}$, where R_s^1 denotes task T_s^1's global worst-case response time and Δ_{NW} denotes the worst-case latency for the exchange of IC protocol messages over the network. This assignment ensures that, in an error-free scenario, the sending step of round one has finished sending all messages and that these messages have been transmitted before the state transition step of round one begins. Other tasks are assigned their release offsets similarly (see Fig. 2).

The task organization discussed above works only if all the tasks with their respective parameters can be integrated *successfully* on the host platforms, *i.e.*, without any deadline misses. This requires the use of a predictable scheduler at runtime and an *a priori* schedulability analysis of the task set. In this work, we consider the partitioned fixed-priority scheduling policy as our predictable scheduling policy, which is supported on all major real-time platforms such as VxWorks and QNX, and also on Linux (via SCHED_FIFO and suitably chosen processor affinity masks). For schedulability analysis, the existing literature on real-time scheduling theory for periodic task models [24] provides a rich foundation for checking if each task meets its deadline *i.e.*, if each task finishes before a dependent task instance arrives.

The proposed task modeling breaks down the IC algorithm into smaller tasks to ensure that the pessimism incurred in the schedulability analysis is minimal. An alternative design where the entire IC algorithm is modeled as one periodic task with suspensions (while awaiting network I/O) requires use of suspension-aware schedulability analyses [25], which are prone to substantial pessimism. Another alternative design wherein the periodic tasks are implemented without suspensions (*i.e.*, when tasks spin while waiting for I/O) is extremely inefficient in terms of CPU usage. Further, note that these alternatives pertain only to the modeling of the protocol implementation. An actual implementation can still realize all tasks (model entities) within a single sequential process (OS facility).

IV. Reliability Analysis

Software reliability in the presence of transient faults can be measured using different metrics [26]. We adopt the *failures-in-time* (FIT) rate, which is an industry-standard reliability metric that denotes the number of failures expected in one billion device operating hours [27]. Since in a real-time context the maximum frequency at which the protocol is invoked is known in advance, to bound a system's FIT rate, it is sufficient to **(i)** derive an upper bound on the failure probability of a single execution of the IC protocol, **(ii)** use this upper bound to compute a lower bound on the mean time to the first failed execution of the protocol, which is also known as its MTTF (see [26, Section 2.2] for a detailed discussion), and then **(iii)** derive an upper bound on the FIT rate as an inverse of the MTTF lower bound. Since steps (ii) and (iii) are trivial, our main contribution is addressing the first step, *i.e.*, the single-invocation failure probability problem.

A. Reliability Assumptions

Our fault model does not account for failures in the operating system and its scheduling mechanism, or the clock synchronization mechanism. In general, while analyzing the failure rate of the IC protocol, we assume that other system components are reliable, even though the protocol execution may directly depend on them. This assumption does not imply that the proposed analysis is not useful if a dependent component fails; rather it provides a FIT rate for the IC protocol, which can then be composed with the FIT rates of other dependent, dependee, or unrelated subsystems using a fault tree analysis [28]. In fact, fault tree analysis is a common way of decomposing the reliability analysis of a complex system into manageable subproblems. Extremely rare events like bit flips affecting the scheduler's priority bits occur with such low likelihood that they are best modeled as a separate, additive failure source and accounted for using a separate FIT analysis. In other words, in a fault-tree analysis of a complete system, orthogonal concerns (*e.g.*, a failing power supply vs. loss of network connectivity) are represented by separate branches of the fault tree, whereas tightly coupled components (*e.g.*, redundant messages that form a middleware transaction) form a single branch and must be analyzed jointly. The contribution of this paper is the first such joint analysis for a hard real-time instantiation of an IC protocol.

B. Protocol-Specific Message Errors

Recall from §II-A that, in each round of the IC protocol, the processes exchange one or more EIG nodes belonging to their respective EIG trees; and in the final round, they process the exchanged information locally to determine their respective decision vectors. Suppose that $M_{i,k}(\alpha)$ denotes the message sent by process Π_i to another process Π_k carrying information about the node labeled α, during round $r = |\alpha| + 1$.

Each message $M_{i,k}(\alpha)$ can be affected by crash or corruption errors. In particular, $M_{i,k}(\alpha)$ can be omitted if its sender host H_i crashes, if one of the switches through which the message is routed crashes, or if its receiver host H_k crashes. $M_{i,k}(\alpha)$ can also be omitted (or rather explicitly dropped) if the Ethernet frame carrying the message is corrupted during transmission and if Ethernet's checksum mechanism successfully detects this corruption. Finally, $M_{i,k}(\alpha)$ can also be corrupted if it was incorrectly prepared in the first place due to corruptions on the sender side, if the Ethernet frame carrying that message is corrupted during transmission but the corruptions are not detected by Ethernet's checksum mechanism, or if it is affected by corruptions on the receiver side just before being delivered. We denote these events as *protocol-specific message errors*.

However, unlike transient faults and basic errors, which are mutually independent, the protocol-specific message errors are not mutually independent. For example, all messages from Π_i to Π_k during round r are typically batched together into a single Ethernet frame; hence, they are simultaneously dropped if the frame gets corrupted and if the corruption is successfully detected. Similarly, if the common payload that is carried by all message frames originating from Π_i during round r is

TABLE I
MESSAGE ERROR EVENTS DUE TO TRANSIENT FAULTS.

Type	Error event description	Remark
1	*"round r msgs. omitted at source E_i"*	Omission
2	*"round r msgs. omitted at switch S_l"*	
3	*"round r msgs. omitted at dest. E_k"*	
4	*"round r frame from Π_i to Π_k omitted by NW"*	
5	*"round r msgs. corrupted at source E_i"*	Corruption
6	*"round r frame from Π_i to Π_k corrupted by NW"*	

corrupted during preparation, even before its checksum has been computed, the corruptions go undetected and are passed on to every message containing the payload.

Hence, for the purpose of this analysis, we conservatively model six types of error events that are defined at a coarser granularity in terms of sets of dependent messages (at the cost of slight pessimism). These are summarized in Table I. Events 1, 2, and 3 denote message omissions due to host and switch crashes during the r^{th} round's sending step of the IC protocol. Events 4 and 6 denote frame omissions and frame corruptions due to perceptible and imperceptible corruption during transmission, respectively. Event 5 denotes the message corruption due to corruption on the host. Unlike omission errors, we do not consider corruption errors at destinations since these are implicitly accounted for as corruption errors at the source of subsequently sent messages.

C. Upper-Bounding Message Error Probabilities

As a first step in the reliability analysis, we derive implementation-specific upper bounds on the probability of protocol-specific message errors (from §IV-B).

1) Event Types 1 and 2 in Table I: Using the Poisson arrival model (recall from §II-C), we first upper-bound the probability of omission errors due to host and switch crashes.

Let t_1 and t'_1 denote the release time of task T^r_s (responsible for the sending step in round r) on hosts E_i and E_k, respectively. Similarly, let $t_2 = t_1 + R^r_s + \Delta_{NW}$ and $t'_2 = t'_1 + R^r_s + \Delta_{NW}$ denote T^r_t's release time on hosts E_i and E_k, respectively. Since the IC protocol rounds on all hosts execute synchronously, and since the host clocks differ by at most Δ_{clock} time units, $|t'_1 - t_1| \leq \Delta_{clock}$ and $|t'_2 - t_2| \leq \Delta_{clock}$.

The sending step on host E_i may be omitted if node E_i is crashed at any time during task T^r_s's scheduling window $[t_1, t_1 + R^r_s)$ (as shown in Fig. 3). Thus, the event *"round r msgs. omitted at source E_i"* may occur if at least one crash occurs during interval $[t_1 - \Delta_{reboot}(E_i),\ t_1 + R^r_s)$, *i.e.*,

$$P(\textit{``round } r \textit{ msgs. omitted at source } E_i\textit{''}) \\ \leq 1 - \mathbb{P}(0,\ R^r_s + \Delta_{reboot}(E_i),\ \gamma_{crash}(E_i)).$$

The round r messages sent from Π_i to Π_k may arrive at Π_k any time during $[t_1, t_1 + R^r_s + \Delta_{NW})$. These messages are then used for updating the EIG tree on E_k any time during task T^r_t's scheduling window $[t'_2, t'_2 + R^r_t)$. Thus, the event *"round r msgs. omitted at dest. E_k"* may occur if at least one crash occurs during $[t_1, t'_2 + R^r_t)$. Since time t_2 and t'_2 may

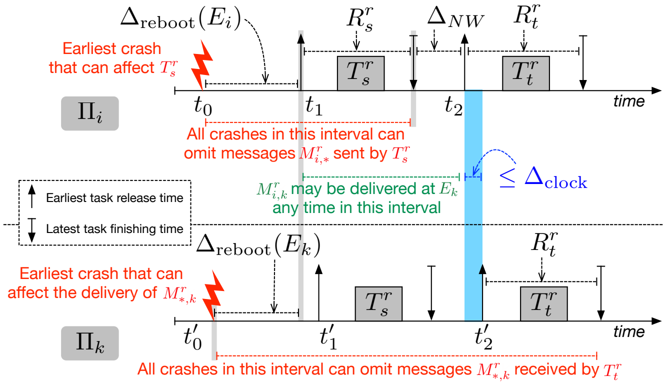

$\Delta_{\text{reboot}}(E_i)$
R^r_s
Δ_{NW}
R^r_t
Earliest crash that can affect T^r_s
T^r_s
T^r_t
Π_i
t_0
t_1
t_2
time
All crashes in this interval can omit messages $M^r_{i,*}$ sent by T^r_s
$\leq \Delta_{\text{clock}}$
$M^r_{i,k}$ may be delivered at E_k any time in this interval
Earliest task release time
Latest task finishing time
$\Delta_{\text{reboot}}(E_k)$
R^r_t
Earliest crash that can affect the delivery of $M^r_{*,k}$
T^r_s
T^r_t
Π_k
t'_0
t'_1
t'_2
time
All crashes in this interval can omit messages $M^r_{*,k}$ received by T^r_t

Fig. 3. Illustration to demonstrate when a crash on the sender side (above) or the receiver side (below) may result in the omission of messages that are to be sent from or delivered to that host, respectively.

differ by at most Δ_{clock} (as also shown in Fig. 3), the event *"round r msgs. omitted at dest. E_k"* may occur if at least one crash occurs during interval $[t_1, t_2 + \Delta_{clock} + R^r_t)$, *i.e.*,

$$P(\textit{``round } r \textit{ msgs. omitted at dest. } E_k\textit{''}) \\ \leq 1 - \mathbb{P}\left(0, \left(\begin{array}{c} R^r_s + \Delta_{NW} + R^r_t + \\ \Delta_{clock} + \Delta_{reboot}(E_k) \end{array}\right), \gamma_{crash}(E_k)\right).$$

2) Event Type 3 in Table I: Consider t_1 and t'_1 as defined above (§IV-C1). All round r messages sent from E_i that are routed through switch S_l can be omitted if switch S_l is crashed at any time during the interval $[t_1, t_1 + R^r_s + \Delta_{NW})$. Similarly, any round r message sent from E_k that is routed through switch S_l may be omitted if switch S_l is crashed at any time during time interval $[t'_1, t'_1 + R^r_s + \Delta_{NW})$. Since these two intervals are expected to be offset by at most Δ_{clock} time units, by generalizing across all round r messages that are routed through S_l, we get the following upper bound.

$$P(\textit{``round } r \textit{ msgs. omitted at switch } S_l\textit{''}) \\ \leq 1 - \mathbb{P}\left(0, \left(\begin{array}{c} R^r_s + \Delta_{NW} + \\ \Delta_{clock} + \Delta_{reboot}(S_l) \end{array}\right), \gamma_{crash}(S_l)\right).$$

The recovery interval of nodes from crashes is typically significantly larger than the task scheduling windows. Hence, for each of the omission errors whose probability is upper-bounded above (event types 1–3 in Table I), we conservatively assume that if a crash error occurs once during the protocol instance, it affects all subsequent tasks (and rounds) on that node in the remaining part of the protocol instance.

3) Event Type 5 in Table I: To upper-bound the probability of corruption errors due to corruption on the host, we need to argue about their exposure intervals. The entire message broadcast $M^r_{i,*} = \bigcup_{\Pi_k \in \Pi} M^r_{i,k}$ may be corrupted if the common payload is corrupted during preparation as part of the sending task T^r_s's execution. The payload corruption may even depend on state corruption during earlier rounds of the same protocol instance (*e.g.*, on the corruption of Π_i's EIG tree). However, recall from §II-C that, due to memory protection mechanisms, latent errors prior to the beginning of the protocol instance do not affect the protocol's execution. Hence, since T_s^r's release offset ϕ_s^r denotes the time since the start of the IC protocol instance and R_s^r denotes the maximum response time of T_s^r, event *"round r msgs. corrupted at source E_i"* may occur if at least one corruption error occurs during an interval of length $\phi_s^r + R_s^r$:

$$P(\text{"round } r \text{ msgs. corrupted at source } E_i\text{"}) \\ \leq 1 - \mathbb{P}(0,\, \phi_s^r + R_s^r,\, \lambda_{corrupt}(E_i)).$$

4) Event Types 4 and 6 in Table I: Next, we upper-bound the probability of frame omission or frame corruption by the network layer. This is non-trivial because the network itself is constituted of multiple components (links and switches), each of which may experience different rates of transient faults.

We assume that all messages sent from Π_i to Π_k during round r, denoted by message set $M_{i,k}^r$, are batched together and sent as a single payload, and that the resulting Ethernet frame does not exceed the Ethernet MTU of 1500 bytes (although this is not a fundamental limitation). The time to transfer the message frame corresponding to the batched frame $M_{i,k}^r$ over a single Ethernet link (which depends on the corresponding frame size and on the bit-transfer rate) is denoted $\Delta_{link}(M_{i,k}^r)$.

The standard 32-bit CRC used in Ethernet networks successfully detects every message corruption with three or fewer bit flips [29], whereas error detection becomes increasingly more difficult with larger numbers of bit-flips. Thus, if the message frame carrying $M_{i,k}^r$ suffers up to three bit-flips during transmission, the corruption is detected and the frame is dropped. (The omission considered here is different from the Type-2 errors analyzed earlier that correspond to involuntary omissions of correct messages due to switch crashes.) In contrast, if the message frame experiences more than three bit-flips, the corruption may remain undetected.

Hence, if events A_1 and A_2 hold, where A_1 denotes event *"$M_{i,k}^r$ suffers no corruption on any of the Ethernet links in $route_{i,k}$"* and A_2 denotes event *"$M_{i,k}^r$ suffers no corruption on any of the switches in $route_{i,k}$,"*, $M_{i,k}^r$ is guaranteed to not be omitted during transmission if:

$$P\left(\begin{array}{c}\text{"round } r \text{ frame from } \Pi_i \\ \text{to } \Pi_k \text{ omitted by NW"}\end{array}\right) \leq 1 - P(A_1)\cdot P(A_2).$$

Supposing that $route_{i,k} = \langle L_{l_1} S_{l_1} L_{l_2} S_{l_2} \ldots L_{l_{n-1}} S_{l_{n-1}} L_{l_n}\rangle$ consists of n hops, and using the independence assumption from §II-C, event probabilities $P(A_1)$ and $P(A_2)$ are defined as

$$P(A_1) = \prod_{1\leq x\leq n} \mathbb{P}\left(0, \Delta_{link}(M_{i,k}^r), \gamma_{corrupt}(L_{l_x})\right) \quad \text{and}$$
$$P(A_2) = \prod_{1\leq x< n} \mathbb{P}\left(0, R^+(M_{i,k}^r, S_{l_x}), \gamma_{corrupt}(S_{l_x})\right),$$

where $R^+(M_{i,k}^r, S_{l_x})$ denotes the maximum queuing delay experienced by message frame $M_{i,k}^r$ on switch S_{l_x}.

Frame $M_{i,k}^r$ is corrupted by the network only if it is undetectably corrupted (*i.e.*, with four or more bit-flips). To accurately upper-bound its probability, we must account for two factors: **(i)** if $M_{i,k}^r$ is undetectably corrupted once, any corruptions later on the network path need not be accounted for (as a worst case, we assume that more bit-flips later do not reverse previous bit-flips and thus do not render the corruption detectable); and **(ii)** before $M_{i,k}^r$ is undetectably corrupted for the first time, it does not suffer any detectable corruptions so as to cause its omission. Thus, we define the probability upper bound as a sum of the probabilities of events $C_{1,x}$ and $C_{2,y}$ for each $1 \leq x \leq n$ and $1 \leq y < n$, where $C_{1,x}$ denotes event *"the first undetectable corruption occurs on the x^{th} link in $route_{i,j}$"* and $C_{2,y}$ denotes event *"the first undetectable corruption occurs on the y^{th} switch in $route_{i,j}$," i.e.*,

$$P\left(\begin{array}{c}\text{"round } r \text{ frame} \\ \text{from } \Pi_i \text{ to } \Pi_k \\ \text{corrupted by NW"}\end{array}\right) \leq \sum_{1\leq x\leq n} P(C_{1,x}) + \sum_{1\leq y<n} P(C_{2,y}).$$

Like $P(A_1)$ and $P(A_2)$, each $P(C_{1,x})$ and $P(C_{1,y})$ can be defined using the independence assumption from §II-C as:

$$P(C_{1,x}) = \left(\begin{array}{l}\left(\prod_{1\leq y<x}\left(\mathbb{P}(0, L_{k_y})\mathbb{P}(0, S_{k_y})\right)\right) \\ \times\ \mathbb{P}(4^+, L_{k_x})\end{array}\right) \text{ and}$$
$$P(C_{2,y}) = \left(\begin{array}{l}\left(\prod_{1\leq z<y}\left(\mathbb{P}(0, L_{k_z})\mathbb{P}(0, S_{k_z})\right)\right) \\ \times\ \mathbb{P}(0, L_{k_y})\mathbb{P}(4^+, S_{k_y})\end{array}\right),$$

$$\begin{aligned}\text{where } \ \mathbb{P}(0, L_{k_y}) &= \mathbb{P}(0, \Delta_{link}(M_{i,k}^r), \gamma_{corrupt}(L_{k_y})), \\ \mathbb{P}(0, S_{k_y}) &= \mathbb{P}(0, R^+(M_{i,k}^r, S_{k_y}), \gamma_{corrupt}(S_{k_y})), \\ \mathbb{P}(4^+, L_{k_y}) &= \sum_{i\geq 4} \mathbb{P}(i, \Delta_{link}(M_{i,k}^r), \gamma_{corrupt}(L_{k_y})), \\ \mathbb{P}(4^+, S_{k_y}) &= \sum_{i\geq 4} \mathbb{P}(i, R^+(M_{i,k}^r, S_{k_y}), \gamma_{corrupt}(S_{k_y})).\end{aligned}$$

D. Estimating Protocol Failure Probability

Suppose that for each error event x (*i.e.*, belonging to one of the six error event types listed in Table I), its exact occurrence probability $P(x)$ is known in advance. In this section, we propose a recursive analysis that defines the IC protocol failure probability $P(\textit{IC failure})$ as a function of each $P(x)$, while taking into account all relevant scenarios. In the next section (§IV-E), we discuss how $P(\textit{IC failure})$'s definition can be updated to use the upper bound on each $P(x)$ (which were derived in §IV-C) instead of $P(x)$ (which are *unknown*).

By exhaustively enumerating all possible cases based on whether each protocol-specific message error event occurs or does not occur, the overall IC protocol failure probability can be derived. However, considering all rounds, hosts, switches, and message frames, altogether tens of errors events need to be evaluated. Furthermore, the total number of cases is exponential in the number of error events. Hence, for efficiency, we identify and prune certain scenarios that are not possible in practice, without compromising the analysis accuracy and safety. For example, if a message is omitted at its source, it cannot be omitted by the network. Therefore, the scenario corresponding to the omission of a message at source and also by the network can be safely excluded from the exhaustive enumeration. In the recursive analysis, we systematically account for such factors in order to reduce the number of cases to be analyzed.

1) Recursive Analysis Overview: The analysis pseudocode is provided in Algorithm 2. Let $\mathcal{M}$ denote the set of all messages exchanged between the processes in an error-free scenario, *i.e.*, $\mathcal{M} = \bigcup_{\Pi_i, \Pi_k \in \Pi, 1 \le r \le N_r} M^r_{i,k}$. To evaluate the probability of a failed protocol instance, we perform a recursive case analysis over all possible error combinations for each message in $\mathcal{M}$. We choose one message at a time from $\mathcal{M}$, consider all scenarios in which it may be affected by the error types listed in Table I, assign case probabilities for each of these scenarios, and recursively evaluate the error possibilities for the next message in $\mathcal{M}$. The recursion terminates when all messages in $\mathcal{M}$ (and hence all possible cases) have been accounted for. We consider messages from round one first, followed by messages from round two, and so on, because message errors during an earlier round may impact message transmissions during subsequent rounds. The ordering of messages from the same round is arbitrary since there is no causal relationship among message errors in the same round.

The analysis maintains the following message sets for bookkeeping. Message set $\mathcal{U}$ is initialized to $\mathcal{M}$. Messages are removed from $\mathcal{U}$ and analyzed one at a time. Every message that is omitted is inserted into message set $\mathcal{O}$. Similarly, every message that is not omitted (and hence delivered on time) but incorrectly computed is inserted into message set $\mathcal{C}$. If a message is neither omitted nor corrupted, it is still removed from $\mathcal{U}$ but inserted into message set $\mathcal{P}$, denoting that it is in pristine condition. Sets $\mathcal{O}$, $\mathcal{C}$, and $\mathcal{P}$ are eventually used during the terminating step of the recursion. We also maintain an event log $\mathcal{E}$ to keep track of the message error events that have already been accounted for in the earlier stages of the recursion, and that must not be accounted for again. Sets $\mathcal{O}$, $\mathcal{C}$, $\mathcal{P}$, and event log $\mathcal{E}$ are initially empty (Line 2).

2) Recursive Cases: The probability that an IC protocol instance fails is denoted $P(\textit{IC failure})$ and computed recursively by the function ProbAnalysisRec (Line 4). First, we obtain a message from $\mathcal{U}$ using GetEarliestMessage (Line 6), which returns messages from round one, followed by messages from round two, and so on. Let $M_{i,k}(\alpha)$ denote this message. Suppose that it belongs to round r, *i.e.*, $|\alpha| = r-1$. Probabilities $P_{\textit{fail}}$ and $P_{\textit{prefix}}$, which keep track of the cumulative failure probability and the case probability prefix (explained below), are then initialized to zero and one, respectively (Line 8).

Based on the error event types in Table I, we consider six cases in which $M_{i,k}(\alpha)$ is affected by errors and one case in which $M_{i,k}(\alpha)$ is transmitted error-free. Case 1 implies that $M_{i,k}(\alpha)$ experienced an error of type 1. Case 2 implies that $M_{i,k}(\alpha)$ did not experience an error of type 1, but experienced an error of type 2, Case 3 implies that $M_{i,k}(\alpha)$ did not experience errors of type 1 and 2, but experienced an error of type 3, and so on. In other words, our analysis explicitly ignores all scenarios that do not adhere to this rule. As a result, all omission errors (event types 1–4) are analyzed first, which is sound since message corruption probabilities contribute to the failure probability only if the message is not omitted. Similarly, an omission at the source (event type 1) is considered first since that determines whether the message is even exposed to omissions by the network (event type 3).

Finally, the case that corresponds to an error-free transmission of message $M_{i,k}(\alpha)$ is evaluated last.

Algorithm 2 Probabilistic analysis of an IC protocol instance.

```
1:  procedure ProbAnalysisInit
2:    $P(\textit{IC failure}) \leftarrow$ ProbAnalysisRec($\mathcal{M}, \emptyset, \emptyset, \emptyset, \emptyset$)
3:
4:  procedure ProbAnalysisRec($\mathcal{U}, \mathcal{O}, \mathcal{C}, \mathcal{P}, \mathcal{E}$)
5:    if $\mathcal{U} = \emptyset$ then return $P(\textit{IC failure} \mid \mathcal{O}, \mathcal{C}, \mathcal{P})$    ▷ termination case
6:    $M_{i,k}(\alpha) \leftarrow$ GetEarliestMessage($\mathcal{U}$)    ▷ the msg. to be analyzed
7:    $r \leftarrow |\alpha| + 1$    ▷ compute the IC protocol round
8:    $P_{\textit{fail}} \leftarrow 0$ , $P_{\textit{prefix}} \leftarrow 1$    ▷ initialize probabilities
9:    $X \leftarrow \langle\rangle$    ▷ an empty FIFO-ordered sequence
10:   $X.enqueue($"round $r$ msgs. omitted at source $E_i$"$)$    ▷ Case 1
11:   for all $S_l \in route_{i,k}$ do    ▷ Case 2
12:     $X.enqueue($"round $r$ msgs. omitted at switch $S_l$"$)$
13:   $X.enqueue($"frame $M^r_{i,k}$ omitted by NW"$)$    ▷ Case 3
14:   $X.enqueue($"round $r$ msgs. omitted at destination $E_i$"$)$    ▷ Case 4
15:   $P_{\textit{fail}}, P_{\textit{prefix}}, \mathcal{E} \leftarrow$ OmissionCases($\mathcal{U}, \mathcal{O}, \mathcal{C}, \mathcal{P}, \mathcal{E}, X, P_{\textit{fail}}, P_{\textit{prefix}}$)
16:   $X \leftarrow \langle\rangle$    ▷ an empty FIFO-ordered sequence
17:   $X.enqueue($"round $r$ msgs. corrupted at source $E_i$"$)$    ▷ Case 5
18:   $X.enqueue($"frame $M^r_{i,k}$ corrupted by NW"$)$    ▷ Case 6
19:   $P_{\textit{fail}}, P_{\textit{prefix}}, \mathcal{E} \leftarrow$ CorruptionCases($\mathcal{U}, \mathcal{O}, \mathcal{C}, \mathcal{P}, \mathcal{E}, X, P_{\textit{fail}}, P_{\textit{prefix}}$)
20:   $P_{\textit{fail}} \leftarrow$ ErrorFreeCase($\mathcal{U}, \mathcal{O}, \mathcal{C}, \mathcal{P}, \mathcal{E}, P_{\textit{fail}}, P_{\textit{prefix}}$)    ▷ Case 7
21:   return $P_{\textit{fail}}$
22:
23: procedure OmissionCases($\mathcal{U}, \mathcal{O}, \mathcal{C}, \mathcal{P}, \mathcal{E}, X, P_{\textit{fail}}, P_{\textit{prefix}}$)
24:   while $X$ is not empty do
25:     $x \leftarrow X.dequeue()$
26:     if $x \notin \mathcal{E}$ then    ▷ analyze event $x$ if not analyzed before
27:       $\mathcal{E} \leftarrow \mathcal{E} \cup \{x\}$    ▷ update $\mathcal{E}$ to prevent repeated analysis
28:       $P_{\textit{case}} \leftarrow P_{\textit{prefix}} \times P(x)$    ▷ compute case probability
29:       $P_{\textit{prefix}} \leftarrow P_{\textit{prefix}} \times \overline{P(x)}$    ▷ update prefix for later cases
30:       $\mathcal{S}_o \leftarrow$ OmittedMessagesGiven($x$)    ▷ dependent messages
31:                          ▷ compute conditional probability using the recursive call
32:       $P_{\textit{cond}} \leftarrow$ ProbAnalysisRec($\mathcal{U} \setminus \mathcal{S}_o, \mathcal{O} \cup \mathcal{S}_o, \mathcal{C}, \mathcal{P}, \mathcal{E}$)
33:       $P_{\textit{fail}} \leftarrow P_{\textit{fail}} + P_{\textit{case}} \times P_{\textit{cond}}$    ▷ update failure probability
34:   return $P_{\textit{fail}}, P_{\textit{prefix}}, \mathcal{E}$    ▷ return params needed in the later cases
35:
36: procedure CorruptionCases($\mathcal{U}, \mathcal{O}, \mathcal{C}, \mathcal{P}, \mathcal{E}, X, P_{\textit{fail}}, P_{\textit{prefix}}$)
37:   while $X$ is not empty do    ▷ similar to OmissionCases ...
38:     $x \leftarrow X.dequeue()$
39:     if $x \notin \mathcal{E}$ then
40:       $\mathcal{E} \leftarrow \mathcal{E} \cup \{x\}$
41:       $P_{\textit{case}} \leftarrow P_{\textit{prefix}} \times P(x)$
42:       $P_{\textit{prefix}} \leftarrow P_{\textit{prefix}} \times \overline{P(x)}$
43:       $\mathcal{S}_c \leftarrow \{M_{i,k}(\alpha)\}$    ▷ ... except this step
44:       $P_{\textit{cond}} \leftarrow$ ProbAnalysisRec($\mathcal{U} \setminus \mathcal{S}_c, \mathcal{O}, \mathcal{C} \cup \mathcal{S}_c, \mathcal{P}, \mathcal{E}$)
45:       $P_{\textit{fail}} \leftarrow P_{\textit{fail}} + P_{\textit{case}} \times P_{\textit{cond}}$
46:   return $P_{\textit{fail}}, P_{\textit{prefix}}, \mathcal{E}$
47:
48: procedure ErrorFreeCase($\mathcal{U}, \mathcal{O}, \mathcal{C}, \mathcal{P}, \mathcal{E}, P_{\textit{fail}}, P_{\textit{prefix}}$)
49:   $\mathcal{S} \leftarrow \{M_{i,k}(\alpha)\}$
50:   $P_{\textit{cond}} \leftarrow$ ProbAnalysisRec($\mathcal{U} \setminus \mathcal{S}, \mathcal{O}, \mathcal{C}, \mathcal{P} \cup \mathcal{S}, \mathcal{E}$)
51:   $P_{\textit{case}} \leftarrow P_{\textit{prefix}}$
52:   $P_{\textit{fail}} \leftarrow P_{\textit{fail}} + P_{\textit{case}} \times P_{\textit{cond}}$
53:   return $P_{\textit{fail}}$
```

3) Cases 1–4: These cases are evaluated by calling the OmissionCases procedure (Line 15). Since an error event may affect multiple messages, it is possible that an error event that might affect $M_{i,k}(\alpha)$ has already been accounted for while analyzing another message in an earlier recursion stage. Thus, each case is evaluated only if the corresponding error event has not already been evaluated before, in which case, it is not in the event log $\mathcal{E}$ (Line 26). If the event is indeed being evaluated for the first time, it is first inserted into $\mathcal{E}$ (Line 27). The case analysis is then executed as follows.

First, the case probability is computed as the product of the probability that prior cases do not occur (given by the latest value of P_{prefix}) and probability $P(x)$ with which the analyzed case occurs (Line 28). Probability P_{prefix} is then updated to account for the negation of the analyzed case, so that it can be reused during the analysis of subsequent cases (Line 29). All messages in M that are omitted either directly or indirectly due to X are computed as $\mathcal{S}_o = \textsc{OmittedMessagesGiven}(X)$ (Line 30). The conditional failure probability is then computed using a recursive call to ProbAnalysisRec with the updated values of $\mathcal{U}$ and $\mathcal{O}$, where the set of omitted messages $\mathcal{S}_o$ is excluded from $\mathcal{U}$ and added to $\mathcal{O}$ (Line 32). In the end, the conditional probability is multiplied with the case probability, and added to the cumulative failure probability (Line 33).

4) Cases 5–7: Cases 5 and 6 are evaluated by calling the CorruptionCases procedure (Line 19), and their analysis is similar to the analysis of Cases 1–4 except for the computation of the conditional failure probability. That is, unlike Cases 1–4, the corrupted message $M_{i,k}(\alpha)$ is removed from $\mathcal{U}$ and added to $\mathcal{C}$ while invoking the recursive call to ProbAnalysisRec (Line 44). Case 7 corresponds to the scenario where $M_{i,k}(\alpha)$ is transmitted error-free (Line 20). In this case, $M_{i,k}(\alpha)$ is removed from $\mathcal{U}$ and inserted into set $\mathcal{P}$ that consists of all pristine messages (Line 50). The case probability for the last case is simply the probability that Cases 1–6 do not occur, given by the latest value of P_{prefix} (Line 51).

5) Terminating Case: The recursion terminates when $\mathcal{U}$ is empty, since each message has been assigned to either $\mathcal{O}$, $\mathcal{C}$, or $\mathcal{P}$ based on whether it is affected by any fault-induced error in this case. What remains is a computation of the conditional probability given $\mathcal{O}$, $\mathcal{C}$, and $\mathcal{P}$ that the IC protocol instance fails, denoted as $P(\mathit{IC\ failure} \mid \mathcal{O}, \mathcal{C}, \mathcal{P})$ (Line 5).

Since it is impossible to estimate $P(\mathit{IC\ failure} \mid \mathcal{O}, \mathcal{C}, \mathcal{P})$ without knowing the exact contents of the corrupted messages, we derive an upper bound on it through worst-case analysis. In a nutshell, since all messages in $\mathcal{M}$ are already partitioned into sets $\mathcal{O}$, $\mathcal{C}$, and $\mathcal{P}$, we can deterministically apply the reduction procedure in the IC protocol to these messages and map the conditional failure probability for the termination case to either zero or one. We assume as a worst-case scenario that all faulty messages are identically corrupted.

6) Last-Mile Errors: A protocol instance may also fail if, at the last moment, say, just after the reduction step, the decision vectors are corrupted or the host crashes. Since the proposed analysis is based on the analysis of message errors, to account for such *last-mile errors*, we use dummy messages that are sent back to the same host. To avoid clutter, Algorithm 2 does not discuss dummy messages; it can be updated as follows. **(i)** The dummy messages are denoted using our regular notation $M_{i,k}(\alpha)$, but with $i = k$ (the value of α is irrelevant for these dummy messages); **(ii)** they are incorporated into the recursive analysis by adding them to message set $\mathcal{M}$ during initialization; **(iii)** function GetEarliestMessage (Line 6) is modified to return one of these dummy messages only if all other regular messages have been analysed; and finally, **(iv)** cases 4 and 6 corresponding to network errors are not applied to the dummy messages (since these messages are local to each host).

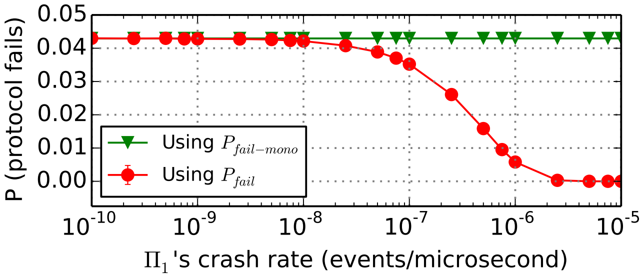


Fig. 4. Given a message corruption rate of 10^{-5} *events/µs*, the failure probability estimated using P_{fail} decreases when Π_1's crash rate is increased from 10^{-10} to 10^{-5} *events/µs*, whereas the failure probability estimated using $P_{fail\text{-}mono}$ always bounds the worst-case failure probability.

E. Reliability Anomalies and their Elimination

Suppose we seek to estimate $P(\mathit{IC\ failure})$ when three processes Π_1, Π_2, and Π_3 execute a two-round IC protocol. Suppose that Π_1 is susceptible to crashes and message corruptions, whereas Π_2 and Π_3 execute error-free. While keeping the message corruption rate on Π_1 constant at $10^{-05}\,events/\mu s$, if Π_1's crash rate is reduced from $10^{-05}\,events/\mu s$ to $10^{-10}\,events/\mu s$, the value of $P(\mathit{IC\ failure})$ as estimated using Algorithm 2 actually increases (as indicated by the curve labeled P_{fail} in Fig. 4). This can be explained by the fact that crash-induced message omissions at Π_1 prevent the transmission of possibly faulty messages, thereby preventing other processes from making a wrong decision, and as a result making the overall protocol execution more reliable. However, the presence of such *reliability anomalies* poses a significant problem: exact values of $P(x)$ are unknown; but computing failure probability using upper bounds on each $P(x)$ (*i.e.*, at peak error rates) does not necessarily yield a safe upper bound on $P(\mathit{IC\ failure})$, and it is infeasible to exhaustively evaluate $P(\mathit{IC\ failure})$ for all possible values of $P(x)$. In the following, we provide an analytical approach to eliminate such anomalies.

We attribute the root cause of reliability anomalies to the non-monotonicity of P_{fail} (returned at the end of function ProbAnalysisRec$(\mathcal{U}, \mathcal{O}, \mathcal{C}, \mathcal{P}, \mathcal{E})$ in Algorithm 2) with respect to each error probability $P(x)$. In particular, since P_{fail} depends on complements of the exact probabilities (see the use of $\overline{P(x)}$ in Lines 29 and 42), it may not be monotonically increasing in each $P(x)$. As a result, replacing each $P(x)$ with its upper bound can yield an *unsafe* estimate of $P(\mathit{IC\ failure})$ that is lower than its exact value. To solve this problem, we derive non-negative correction terms that are added to the analysis to ensure monotonicity, as shown below for a simplified scenario.

Suppose that message $M_{i,k}(\alpha)$ can experience only two types of errors, $E_1 =$ *"round r msgs. omitted at source E_i"* and $E_2 =$ *"frame $M^r_{i,k}$ corrupted by NW"* (where $r = |\alpha| + 1$). Let P_1 and P_2 denote the event probabilities for E_1 and E_2 (respectively), C_1 and C_2 denote the conditional failure probability bounds for cases corresponding to events E_1 and E_2 (respectively), and let C_3 denote the conditional failure probability bound for the case that $M_{i,k}(\alpha)$ experiences neither E_1 nor E_2. C_1, C_2, and C_3 correspond to the recursive calls in Algorithm 2, and since our analysis never evaluates an event twice, they do not depend on P_1 and P_2.

As explained in §IV-D, $M_{i,k}(\alpha)$ can experience E_2 only if it did not experience E_1 first, which is accounted for by the order in which these events are analyzed. Thus, the failure probability P_{fail} returned at the end of the recursive step concerned with the analysis of message $M_{i,k}(\alpha)$ reduces to:

$$\begin{aligned} P_{fail} &= P_1C_1 + \overline{P_1}P_2C_2 + \overline{P_1}\,\overline{P_2}C_3 \\ &= C_3 + P_1(C_1 - C_3) + P_2(C_2 - C_3) - P_1P_2(C_2 - C_3). \end{aligned}$$

The above expression is monotonic in P_1 and P_2 only if C_1 and C_2 are greater than or equal to C_3. However, even though C_3 corresponds to the scenario where $M_{i,k}(\alpha)$ does not experience errors E_1 or E_2, C_1 and C_2 can be smaller than C_3 because of the anomaly that omission errors can possibly reduce the overall failure chances. Thus, we determine whether $C_1 \geq C_3$ and $C_2 \geq C_3$ at analysis time. We define boolean values B_1 and B_2 such that, for each $i \in \{1, 2\}$, if $C_i \geq C_3$ then $B_i = 1$ otherwise $B_i = 0$. Using these, $C_i - C_3$ can be expressed as $B_i \cdot |C_i - C_3| - (1 - B_i) \cdot |C_i - C_3|$, and P_{fail} can be split into two terms $P_{fail,pos}$ and $P_{fail,neg}$, which are guaranteed to be non-negative and non-positive (respectively):

$$\begin{aligned} &P_{fail} = P_{fail,pos} + P_{fail,neg}, \ \textit{where} \\ &P_{fail,pos} = \left(\begin{array}{l} C_3 + P_1B_1|C_1 - C_3| + P_2B_2|C_2 - C_3| \\ + P_1P_2(1 - B_2)|C_2 - C_3| \end{array} \right), \\ &P_{fail,neg} = \left(\begin{array}{l} P_1(1 - B_1)|C_1 - C_3| + \\ P_2(1 - B_2)|C_2 - C_3| + \ P_1P_2B_2|C_2 - C_3| \end{array} \right). \end{aligned}$$

It is now trivial to obtain a safe upper bound on P_{fail} (denoted $P_{\textit{fail-mono}}$) that is guaranteed to be monotonic in P_1 and P_2 by negating all negative terms, *i.e.*, $P_{\textit{fail-mono}} = P_{fail} - P_{fail,neg}$. Consecutively, using $P_{\textit{fail-mono}}$ in place of P_{fail} in Algorithm 2 always yields a safe upper bound on $P(\textit{IC failure})$ (see the green curve in Fig. 4).

In general, the decomposition of P_{fail} into $P_{fail,pos}$ and $P_{fail,neg}$ explained in the example above can be accomplished analogously for all scenarios, *i.e.*, for all instances of the recursive function where one or more cases may not be analyzed (since they were analyzed before), and can also be easily generalized to different network topologies. We provide the full details in the Appendix.

In summary, to ensure safety, it is important to eliminate reliability anomalies. Actual fault rates may vary unpredictably over time, and are possibly correlated across different components (*e.g.*, during fault bursts). Since we address this unpredictability by analyzing *peak* fault rates for all components (recall our fault model from §II-C), thereby implicitly accounting for any variations or correlations in *actual* fault rates, there must not be any reliability anomalies. To this end, our approach eliminates reliability anomalies by adding non-negative correction terms. Conceptually, such terms may increase pessimism, but we show in our evaluation that this increase is negligible (see §V-A).

F. Analysis Complexity

The proposed analysis is conducted entirely offline. Its complexity is exponential in the number of messages, which is proportional to the number of hosts and the number of switches. While this can lead to intractable analysis runtimes for configurations with many replicas and switches, in practice two factors help to keep analysis times acceptable. First, in practical deployments, there are typically only between two to four replicas. Thus, even if the total number of hosts in the system is large, the analysis must consider only up to four hosts and the traversed switches. Second, as mentioned in §IV-D, we prune impossible scenarios (like a message being omitted by both the source and a switch) to speed up the analysis.

V. Evaluation

We implemented our analysis in C++ using the GNU MPFR library [30] to ensure that all analysis computations were carried out at a precision of 200 decimal places. Upper bounds on the message scheduling delays on switches (recall the use of $R^+(M^r_{i,k}, S_{l_x})$ from §IV-C4) were computed using the Compositional Performance Analysis (CPA) [31, 32]. In particular, we used the open-source pyCPA framework [33]. All experiments were carried out on Intel Xeon E7-8857 v2 machines (48 cores, $1.5\,\text{TB}$ of memory) clocked at $3\,\text{GHz}$.

The analyzed workload consisted of up to four processes on four different hosts periodically executing a hard real-time IC protocol instance every $P = 100\,ms$. The global worst-case response time for each IC protocol task was set to $1\,ms$. The hosts were assumed to be connected via a single switch (*star* topology) or via multiple switches arranged in either a *line* or a *ring* topology (Fig. 5). We used a transfer rate of $100\,Mbps$ for each port and a wire delay of $330\,ns$ for each link. The hosts also periodically exchanged PTP messages for clock synchronization. Based on PTPd version 2.3.2, these messages have a payload of $76\,bytes$ each and a period of $500\,ms$. We assumed periodic exchanges of maximum-sized frames between hosts to model lower-priority traffic, which results in worst-case blocking delay for the IC protocol messages.

The crash recovery times were set to $1\,s$. Error rates are reported as the *mean number of errors per microsecond*. Unless mentioned otherwise, experiments assume the strong correctness criterion defined in §II-B1.

A. Analysis vs. Simulation

We compared `Unsafe-Analysis` (with reliability anomalies) and `Mono-Analysis` (without reliability anomalies) with simulation baselines `Sim-v1` and `Sim-v2` to evaluate the pessimism incurred due to reliability anomalies elimination.

`Sim-v1` knows in advance the message error probabilities for each IC protocol message. Thus, for every error type, `Sim-v1` draws a number uniformly at random from the range $[0, 1]$, compares it with the respective error probability to decide whether the error is encountered or not, and if the error is encountered, simulates the corresponding error scenario.

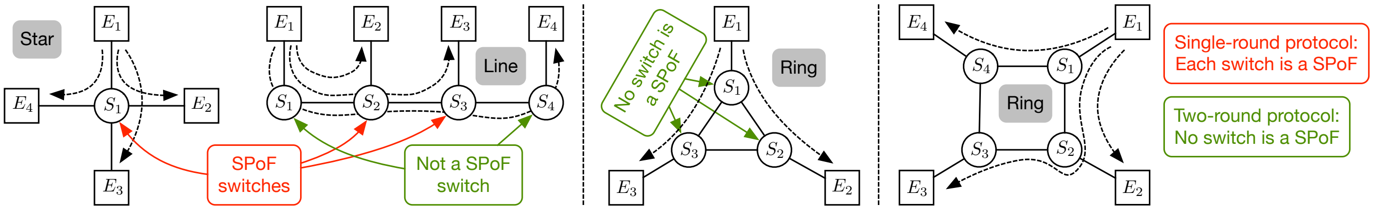


Fig. 5. Network topologies with static routes (dotted arrows) from E_1 to other hosts. SPoF denotes "Single Point of Failure".

`Sim-v1` thus helps to isolate the pessimism incurred (if any) in our recursive analysis procedure. In contrast, `Sim-v2` simulates FIFO priority queues at the network layer and uses Poisson processes to generate the respective fault events on each host and on the network. These events may manifest as message errors if they coincide with the message's lifetime, *e.g.*, as an incorrect computation error if they coincide with the message's exposure interval. `Sim-v2` evaluates the pessimism incurred when upper-bounding the message error probabilities as a function of the raw transient fault rates using the Poisson model.

Both `Sim-v1` and `Sim-v2` make the worst-case assumption that any two faulty message copies are identical, as in the analysis. The simulations were run as a discrete event simulation for 100 000 iterations each to ensure that the 99th percentile confidence intervals were of negligible magnitude relative to the absolute values.

We compared `Unsafe-Analysis`, `Mono-Analysis`, `Sim-v1`, and `Sim-v2` for different topologies, host crash error rates (0 or 10^{-8}), switch crash error rates (0 or 10^{-8}), host corruption rates (0 or 10^{-5}), switch corruption rates (0 or 10^{-5}), and link corruption rates (0 or 0.001). We used higher error rates than can be realistically expected as otherwise the simulations would be extremely time-consuming. The objectives of these comparisons are threefold. First, we compare `Mono-Analysis` and `Unsafe-Analysis` to understand the magnitude of pessimism due to correction terms. Second, we compare `Mono-Analysis` and `Unsafe-Analysis` with `Sim-v1` as a baseline that yields results similar in magnitude to what a probabilistic model checker would obtain. Third, comparisons with `Sim-v2` illuminate the magnitude of pessimism incurred due to the message error analysis in §IV-C.

In Fig. 6(a), we illustrate the results for $N_p = 3$ and $N_r = 2$. `Unsafe-Analysis` exactly tracks `Sim-v1`, which indicates that the recursive analysis presented in §IV-D incurs no substantial pessimism. `Mono-Analysis` also closely tracks `Unsafe-Analysis` and `Sim-v1`, (*i.e.*, it does not exhibit notable pessimism), which we attribute to the anomaly correction terms having negligibly small magnitudes when not needed. In contrast, the analysis results do not closely track `Sim-v2` for some configurations (with the maximum observed relative error of about 2.1×). Higher pessimism results from the analysis to upper-bound host corruption errors (§IV-C1), since the exposure intervals for different protocol tasks overlap, *i.e.*, the exposure interval of each task includes the time since the start of the IC protocol instance.

B. Case Studies

We conducted the following experiments to understand the benefits (if any) of using the weak correctness criterion (whenever the application permits), and the effects of non-uniform fault rates and different network topologies on the protocol reliability. The error rates used are much smaller than those used in the previous section, since the analysis runtime (unlike simulations) does not depend on the magnitude of error rates. In particular, we use realistic error rates derived from prior studies on transient fault rates [34, 35].

We evaluated FIT bounds for the strong and weak correctness criteria for six configurations with $N_p \in \{2, 3, 4\}$ and $N_r \in \{1, 2\}$. We only considered crash errors in this experiment (each host has a crash rate of 10^{-15}).

The results in Fig. 6(b) show that the FIT bounds for the strong criterion are orders of magnitude higher than the FIT bounds for the weak criterion, which indicates that the protocol is much more likely to violate the strong criterion (as expected). Therefore, an effective reliability analysis must account for the weak criterion whenever it suffices for an application, to obtain more accurate failure rates.

In addition, when the number of processes is increased from two to three, while the FIT bounds for the weak criterion decrease, the FIT bounds for the strong criterion remain the same. This observation corroborates the findings from classical BFT theory (which is modeled by the strong correctness criterion) that going from an odd number of replicas to an even number of replicas does not yield any reliability benefits.

Surprinsingly, the results in Fig. 6(b) indicate that additional rounds seemingly never help. This is a consequence of crash errors, which dominate in these scenarios, since a crash is likely to keep the node unavailable for all rounds of the protocol. We repeated a similar experiment for $N_p = 3$ while considering only network corruption errors. The resulting FIT bounds for $N_r = 1$ and $N_r = 2$ were 3.623×10^{-5} and 6.993×10^{-14} (respectively), clearly indicating the benefit of multiple rounds when the dominant error sources affect different rounds independently.

Next, we sought to understand the impact of different network topologies as well as non-uniform error rates on the evaluated FIT analysis. Therefore, we considered only switch crash errors in this experiment, assigned a crash error rate of 10^{-15} to switches S_1 and S_2 (see Fig. 5 for reference), whereas other switches were assumed to execute error-free. Assuming the strong correctness criterion, we computed FIT bounds for

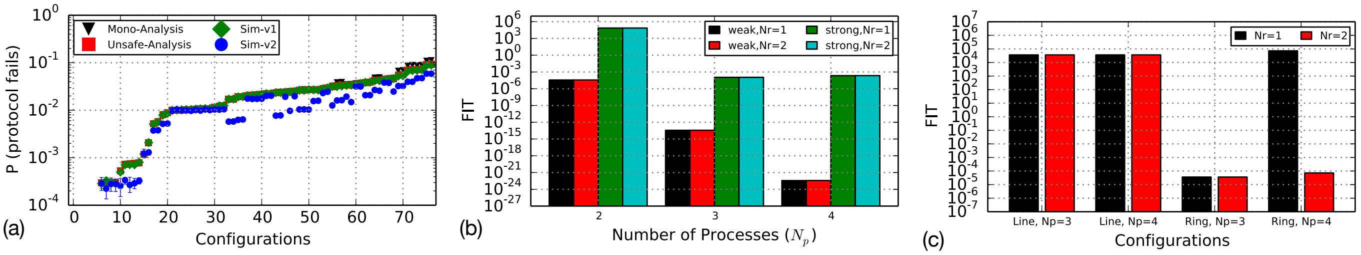


Fig. 6. **(a)** Failure probabilities estimated by `Unsafe-Analysis`, `Mono-Analysis`, `Sim-v1`, and `Sim-v2` (in increasing order of `Unsafe-Analysis` results). **(b)** FIT bounds estimated in the presence of host crash errors. **(c)** FIT bounds estimated in the presence of switch crash errors.

eight different configurations: line and ring topology, $N_p \in \{3, 4\}$, and $N_r \in \{1, 2\}$. The results are shown in Fig. 6(c).

We observe that all configurations with line topology have very high FIT bounds with negligible differences. This is because switch S_2 is a single point of failure (SPoF) in a line topology with three or four hosts (two hosts cannot form a quorum if $N_p = 4$). In contrast, if three hosts are arranged in a ring topology, the FIT bounds are low since no single switch is a SPoF (failure results only if both S_1 and S_2 crash). Interestingly, four hosts benefit from the ring topology only if $N_r = 2$. We attribute this to a combination of two factors: static routing and asymmetric IC protocol rounds. Static routing prevents switches from immediately moving to an alternate route. Thus, every single switch becomes a SPoF for $N_r = 1$. However, for $N_r = 2$, if Π_3 misses a message from Π_1 in the first round owing to S_2's crash, it still gets a chance to receive Π_1's private value from Π_4 in the second round, since messages from Π_4 and Π_1 are not routed through S_2.

To conclude, the proposed analysis enables a systematic exploration of the parameter space with respect to protocol reliability, which allows any weak links, or even the presence of any "redundant redundancies," to become apparent.

VI. Related Work

Reliability analyses of real-time systems have been widely studied in the context of field buses [10, 36, 37]. A common theme across these works is modeling of transient faults either probabilistically (as in this paper) or using periodic models with bursts, and then leveraging the timing properties of the field bus to bound the likelihood of transient faults affecting a given message. Recently, similar results have also been obtained for Ethernet-based real-time systems [38]. However, these prior works primarily focus on low-level properties, *e.g.*, Broster *et al.*'s analysis [10] upper-bounds the probability of an individual message's timely transmission over CAN despite fault-induced retransmissions. In contrast, our contribution is to leverage such message-level analyses to evaluate the failure rate of a complex, higher-level, multi-round distributed agreement protocol.

Prior work on Byzantine fault tolerance has not investigated BFT protocols from the perspective of hard real-time predictability and transient faults. For instance, classic high-level Byzantine safety guarantees (*e.g.*, $3f + 1$ processes can tolerate up to f Byzantine faults) are oblivious to non-uniform fault rates across different components of the system that arise due to environmental disturbances. In contrast, our analysis evaluates a hard real-time CPS-friendly implementation of a BFT protocol, considers timing delays in our correctness definitions, and explicitly models PE nodes, network switches, and network links.

Ryan *et al.* [39] presented the design and implementation of an IC protocol suitable for real-time applications based on time-triggered CAN (TTCAN). In contrast, our goal is to realize such protocols over point-to-point Ethernet topologies based on message-passing hard real-time periodic tasks. In future work, it would be interesting to transfer our analysis methodology to quantify the reliability of Ryan *et al.*'s TTCAN-based design.

Simulations and probabilistic model checking are alternate techniques to solve the reliability analysis problem, but they do not account for reliability anomalies. These alternatives also suffer from scalability issues when the error probabilities are very small. Simulations must be run for excessively long durations to estimate the failure rate with high confidence, and probabilistic model checkers such as PRISM [40] need to fall back on exact model checking to avoid incorrect results due to floating-point noise. For example, evaluting the reliability of even a very basic distributed system [41] using PRISM takes up to a few hours when exact representations are used (whereas otherwise, it takes only a few seconds).

VII. Conclusion

In this work, we presented the first quantitative reliability analysis of real-time systems connected by point-to-point networks (such as Ethernet) in the presence of environmentally induced Byzantine errors. Our analysis explicitly models hosts, network switches, and network links, and considers the effect of transient faults in any of them. Importantly, our analysis is free from reliability anomalies, *i.e.*, when a non-maximal fault rate in some component can counter-intuitively result in an increase of the system's overall failure rate. In fact, to the best of our knowledge, this is the first work to formalize the concept of reliability anomalies, and to propose techniques to eliminate such anomalies in a hard real-time setting. In future work, it would be interesting to evaluate a practical prototype of the analyzed protocol, and to incorporate recent advances in real-time Ethernet standards related to flow integrity, such as different stream reservation and path control protocols, into our quantitative reliability analysis framework.

References

[1] F. Mathur, “On Reliability Modeling and Analysis of Ultra-reliable Fault-Tolerant Digital Systems,” *IEEE Transactions on Computers*, vol. C-20, no. 11, pp. 1376–1382, Nov. 1971.

[2] A. Hopkins, T. Smith, and J. Lala, “FTMP—A highly reliable fault-tolerant multiprocess for aircraft,” *Proceedings of the IEEE*, vol. 66, no. 10, pp. 1221–1239, 1978.

[3] S. Arthasartsri and H. Ren, “Validation and verification methodologies in A380 aircraft reliability program,” in *2009 8th International Conference on Reliability, Maintainability and Safety*. Chengdu, China: IEEE, Jul. 2009, pp. 1356–1363.

[4] S. S. Banerjee, S. Jha, J. Cyriac, Z. T. Kalbarczyk, and R. K. Iyer, “Hands Off the Wheel in Autonomous Vehicles?: A Systems Perspective on over a Million Miles of Field Data,” in *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. Luxembourg City: IEEE, Jun. 2018, pp. 586–597.

[5] A. Nasrallah, A. S. Thyagaturu, Z. Alharbi, C. Wang, X. Shao, M. Reisslein, and H. ElBakoury, “Ultra-Low Latency (ULL) Networks: The IEEE TSN and IETF DetNet Standards and Related 5G ULL Research,” *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 88–145, 2019.

[6] “Automotive Ethernet: The Future of In-Car Networking?” *Electronic Design*, Apr. 2018.

[7] J. M. Gitlin, “General Motors designs a new “brain and nervous system” for its vehicles,” *Ars Technica*, May 2019.

[8] K. Driscoll, B. Hall, H. Sivencrona, and P. Zumsteg, “Byzantine Fault Tolerance, from Theory to Reality,” in *Computer Safety, Reliability, and Security*, G. Goos, J. Hartmanis, J. van Leeuwen, S. Anderson, M. Felici, and B. Littlewood, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, vol. 2788, pp. 235–248.

[9] J. B. Dugan and R. Van Buren, “Reliability evaluation of fly-by-wire computer systems,” *Journal of Systems and Software*, vol. 25, no. 1, pp. 109–120, Apr. 1994.

[10] I. Broster, A. Burns, and G. Rodriguez-Navas, “Timing Analysis of Real-Time Communication Under Electromagnetic Interference,” *Real-Time Systems*, vol. 30, no. 1-2, pp. 55–81, May 2005.

[11] J. Song, J. Wittrock, and G. Parmer, “Predictable, Efficient System-Level Fault Tolerance in Cˆ3,” in *2013 IEEE 34th Real-Time Systems Symposium*. Vancouver, BC, Canada: IEEE, Dec. 2013, pp. 21–32.

[12] Y. Amir, B. Coan, J. Kirsch, and J. Lane, “Prime: Byzantine Replication under Attack,” *IEEE Transactions on Dependable and Secure Computing*, vol. 8, no. 4, pp. 564–577, Jul. 2011.

[13] M. Pease, R. Shostak, and L. Lamport, “Reaching Agreement in the Presence of Faults,” *Journal of the ACM*, vol. 27, no. 2, pp. 228–234, Apr. 1980.

[14] P. Brisset, A. Drouin, M. Gorraz, P.-S. Huard, and J. Tyler, “The Paparazzi Solution,” in *MAV 2006, 2nd US-European Competition and Workshop on Micro Air Vehicles*, Sandestin, United States, Oct. 2006, hAL ID: hal-01004157.

[15] C. L. Liu and J. W. Layland, “Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment,” *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, Jan. 1973.

[16] Y. Lu, “Probabilistic verification of satellite systems for mission critical applications,” PhD, University of Glasgow, 2016.

[17] F. Borran and A. Schiper, “A Leader-Free Byzantine Consensus Algorithm,” in *Distributed Computing and Networking*, K. Kant, S. V. Pemmaraju, K. M. Sivalingam, and J. Wu, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 67–78.

[18] N. Nakka, G. P. Saggese, Z. Kalbarczyk, and R. K. Iyer, “An Architectural Framework for Detecting Process Hangs/Crashes,” in *Dependable Computing - EDCC 5*, D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, M. Dal Cin, M. Kaâniche, and A. Pataricza, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, vol. 3463, pp. 103–121.

[19] C. Slayman, “Cache and memory error detection, correction, and reduction techniques for terrestrial servers and workstations,” *IEEE Transactions on Device and Materials Reliability*, vol. 5, no. 3, pp. 397–404, Sep. 2005.

[20] A. Gujarati and B. B. Brandenburg, “When Is CAN the Weakest Link? A Bound on Failures-in-Time in CAN-Based Real-Time Systems,” in *RTSS 2015*, 2015.

[21] X. Li, S. V. Adve, P. Bose, and J. A. Rivers, “Architecture-Level Soft Error Analysis: Examining the Limits of Common Assumptions,” in *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN’07)*. Edinburgh, UK: IEEE, Jun. 2007, pp. 266–275.

[22] S. Mukherjee, C. Weaver, J. Emer, S. Reinhardt, and T. Austin, “A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor,” in *22nd Digital Avionics Systems Conference. Proceedings (Cat. No.03CH37449)*. San Diego, CA, USA: IEEE Comput. Soc, 2003, pp. 29–40.

[23] D. Fontanelli, D. Macii, P. Wolfrum, D. Obradovic, and G. Steindl, “A clock state estimator for PTP time synchronization in harsh environmental conditions,” in *2011 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*. Munich, Germany: IEEE, Sep. 2011, pp. 99–104.

[24] R. I. Davis and A. Burns, “A survey of hard real-time scheduling for multiprocessor systems,” *ACM Computing Surveys*, vol. 43, no. 4, pp. 1–44, Oct. 2011.

[25] J.-J. Chen, G. Nelissen, W.-H. Huang, M. Yang, B. B. Brandenburg, K. Bletsas, C. Liu, P. Richard, F. Ridouard, N. Audsley, R. Rajkumar, D. de Niz, and G. von der Brüggen, “Many suspensions, many problems: a review of self-suspending tasks in real-time systems,” *Real-Time Systems*, Sep. 2018.

[26] W. Kuo and M. J. Zuo, *Optimal reliability modeling: principles and applications*. Hoboken, N.J: John Wiley & Sons, 2003.

[27] S. Stanley, “MTBF, MTTR, MTTF & FIT Explanation of Terms.”

[28] E. Ruijters and M. Stoelinga, “Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools,” *Computer Science Review*, vol. 15-16, pp. 29–62, Feb. 2015.

[29] P. Koopman, “32-bit cyclic redundancy codes for Internet applications,” in *Proceedings International Conference on Dependable Systems and Networks*. Washington, DC, USA: IEEE Comput. Soc, 2002, pp. 459–468.

[30] “The GNU MPFR Library.”

[31] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst, “System level performance analysis – the SymTA/S approach,” *IEE Proceedings - Computers and Digital Techniques*, vol. 152, no. 2, p. 148, 2005.

[32] J. Diemer, D. Thiele, and R. Ernst, “Formal worst-case timing analysis of Ethernet topologies with strict-priority and AVB switching,” in *7th IEEE International Symposium on Industrial Embedded Systems (SIES’12)*. Karlsruhe, Germany: IEEE, Jun. 2012, pp. 1–10.

[33] “Welcome — pyCPA current documentation.”

[34] A. A. Atallah, G. Bany Hamad, and O. Ait Mohamed, “Reliability-Aware Routing of AVB Streams in TSN Networks,” in *Recent Trends and Future Technology in Applied Intelligence*, M. Mouhoub, S. Sadaoui, O. Ait Mohamed, and M. Ali, Eds. Cham: Springer International Publishing, 2018, vol. 10868, pp. 697–708.

[35] P. Hazucha and C. Svensson, “Impact of CMOS technology scaling on the atmospheric neutron soft error rate,” *IEEE Transactions on Nuclear Science*, vol. 47, no. 6, pp. 2586–2594, Dec. 2000.

[36] R. I. Davis, A. Burns, R. J. Bril, and J. J. Lukkien, "Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised," *Real-Time Systems*, vol. 35, no. 3, pp. 239–272, Feb. 2007.

[37] M. Sebastian, P. Axer, and R. Ernst, "Utilizing Hidden Markov Models for Formal Reliability Analysis of Real-Time Communication Systems with Errors," in *2011 IEEE 17th Pacific Rim International Symposium on Dependable Computing*. Pasadena, CA, USA: IEEE, Dec. 2011, pp. 79–88.

[38] F. Smirnov, M. Glaß, F. Reimann, and J. Teich, "Formal reliability analysis of switched ethernet automotive networks under transient transmission errors," in *Proceedings of the 53rd Annual Design Automation Conference on - DAC '16*. Austin, Texas: ACM Press, 2016, pp. 1–6.

[39] C. Ryan, D. Heffernan, and G. Leen, "Interactive Consistency on a Time-Triggered Real-Time Control Network," *IEEE Transactions on Industrial Informatics*, vol. 2, no. 4, pp. 242–254, Nov. 2006.

[40] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM 4.0: Verification of Probabilistic Real-Time Systems," in *Computer Aided Verification*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, vol. 6806, pp. 585–591.

[41] ——, "Controller dependability analysis by probabilistic model checking," *Control Engineering Practice*, vol. 15, no. 11, pp. 1427–1434, Nov. 2007.

Appendix

Recall the recursive analysis from §IV-D. We eliminate reliability anomalies when using Algorithm 2 by replacing P_{fail} with $P_{fail\text{-}mono}$ in the algorithm. We defined $P_{fail\text{-}mono}$ for a simplified scenario in §IV-E. In this section, we show how $P_{fail\text{-}mono}$ can be defined for a generic scenario, *i.e.*, for different instances of the recursive function in Algorithm 2 and for different network topologies.

For brevity, we first introduce a shorthand notation (see Table II) to denote the exact error probabilities and the conditional failure probabilities used in Algorithm 2. For each P_i in Table II, we let $\overline{P_i} = 1 - P_i$. Note that the shorthand notation is defined with respect to the specific iteration of the recursive analysis, *i.e.*, pertaining to the analysis of message $M_{i,k}(\alpha)$ specifically. To start with, we assume that message $M_{i,k}(\alpha)$ is routed through a single switch S_l. Later, we show how the results can be extended to more general topologies.

Using the shorthand notation, P_{fail} returned at the end of function ProbAnalysisRecSM$(\mathcal{U}, \mathcal{O}, \mathcal{C}, \mathcal{P}, \mathcal{E})$ is defined as follows. If all events in X are being analyzed for the first time during the recursive analysis, *i.e.*, the condition $x \in \mathcal{E}$ (in Line 26) evaluates to *false* for each $x \in X$, then

$$P_{fail} = \begin{pmatrix} P_1 \cdot C_1 \\ +\overline{P_1} \cdot P_2 \cdot C_2 \\ +\overline{P_1} \cdot \overline{P_2} \cdot P_3 \cdot C_3 \\ +\overline{P_1} \cdot \overline{P_2} \cdot \overline{P_3} \cdot P_4 \cdot C_4 \\ +\overline{P_1} \cdot \overline{P_2} \cdot \overline{P_3} \cdot \overline{P_4} \cdot P_5 \cdot C_5 \\ +\overline{P_1} \cdot \overline{P_2} \cdot \overline{P_3} \cdot \overline{P_4} \cdot \overline{P_5} \cdot P_6 \cdot C_6 \\ +\overline{P_1} \cdot \overline{P_2} \cdot \overline{P_3} \cdot \overline{P_4} \cdot \overline{P_5} \cdot \overline{P_6} \cdot C_7 \end{pmatrix}. \quad (1)$$

In case an event in X has already been analyzed during an earlier stage of the recursion, the corresponding case analysis is skipped, since $x \in \mathcal{E}$ would evaluate to *true* (see Line 26). In this case, if, say, event *"all round r msgs. omitted at source E_i"* has already been analyzed, P_{fail} for this recursion step is defined by setting probabilities P_1 and C_1 to zero in Eq. (1). Thus, in the following, we assume P_{fail} is defined generically as in Eq. (1). The results can be extrapolated to all scenarios (*i.e.*, to all instances of the recursive function) by setting the appropriate probabilities to zero.

Clearly, since P_{fail} relies on complementary probability terms $\overline{P_i}$'s, it is not apparent if P_{fail} is monotonic in all P_i's, and if not, how can P_{fail}'s definition be enhanced (while being safe) to ensure monotonicity. Thus, to assist with the derivation of the correction terms (and in a way that can also be generalized to any number of switches), we express P_{fail} in a canonical form consisting of only P_i's, *i.e.*, where all $\overline{P_i}$'s in Eq. (1) are replaced with $1 - P_i$, *i.e.*,

$$\begin{aligned} P_{fail} &= T_1 + T_2 + T_3 + T_4 + T_5 + T_6 + T_7, \quad (2) \\ \textit{where}\ \ T_1 &= C_7, \\ T_2 &= \sum_{i=1}^{6} P_i(C_i - C_7), \\ T_3 &= -\sum_{i=1}^{5} \sum_{j=i+1}^{6} P_i P_j (C_j - C_7), \\ T_4 &= \sum_{i=1}^{4} \sum_{j=i+1}^{5} \sum_{k=j+1}^{6} P_i P_j P_k (C_k - C_7), \\ T_5 &= -\sum_{i=1}^{3} \sum_{j=i+1}^{4} \sum_{k=j+1}^{5} \sum_{l=k+1}^{6} P_i P_j P_k P_l (C_l - C_7), \\ T_6 &= \sum_{i=1}^{2} \sum_{j=i+1}^{3} \sum_{k=j+1}^{4} \sum_{l=k+1}^{5} \sum_{m=l+1}^{6} \begin{pmatrix} P_i P_j P_k P_l P_m \\ (C_m - C_7) \end{pmatrix}, \\ \textit{and}\qquad T_7 &= -P_1 P_2 P_3 P_4 P_5 P_6 (C_6 - C_7). \end{aligned}$$

In Eq. (2), if each C_i is greater than or equal to C_7, P_{fail} is monotonic in each P_i. However, even though C_7 corresponds to the conditional failure probability in an error-free scenario whereas each C_i corresponds to a conditional failure probability in an error scenario, C_i can be smaller than C_7 because of the anomaly that omission errors can sometimes reduce the failure chances. Instead, for each C_i, we rely on a boolean value B_i that can be evaluated at analysis runtime to denote whether $C_i \geq C_7$, based on the following definition:

$$\textit{if}\ C_i \geq C_7\ \textit{then}\ B_i = 1\ \textit{else}\ B_i = 0. \quad (3)$$

Using Eq. (3), we rewrite each term $C_i - C_7$ in Eq. (2) as

$$C_i - C_7 = B_i \cdot |C_i - C_7| - (1 - B_i) \cdot |C_i - C_7|. \quad (4)$$

Next, using Eq. (4), and relying on the fact that all probabilities (*i.e.*, each P_i and C_i) are positive, we split P_{fail} (Eq. (2)) into two terms $P_{fail,pos}$ and $P_{fail,neg}$, such that $P_{fail,pos}$ is guaranteed to be non-negative and $P_{fail,neg}$ is

TABLE II

SHORTHAND NOTATION FOR THE EXACT MESSAGE ERROR PROBABILITIES AND INTERMEDIATE CONDITIONAL FAILURE PROBABILITIES USED IN ALGORITHM 2. MESSAGE $M_{i,k}(\alpha)$ IS ASSUMED TO BE ROUTED THROUGH A SINGLE SWITCH S_l IN THIS CASE.

Label	Error event x in Line 26	Error event probability $P(x)$	Conditional probability P_{cond}
X_1	$x =$ *"all round r msgs. omitted at source E_i"*	P_1	C_1 (Line 32)
X_2	$x =$ *"all round r msgs. omitted at switch S_l"*	P_2	C_2 (Line 32)
X_3	$x =$ *"all round r msgs. omitted at dest. E_k"*	P_3	C_3 (Line 32)
X_4	$x =$ *"round r frame from Π_i to Π_k omitted by NW"*	P_4	C_4 (Line 32)
X_5	$x =$ *"all round r msgs. corrupted at source E_i"*	P_5	C_5 (Line 44)
X_6	$x =$ *"round r frame from Π_i to Π_k corrupted by NW"*	P_6	C_6 (Line 44)
-	$x =$ *"message $M_{i,k}(\alpha)$ is transmitted error-free"*	-	C_7 (Line 50)

TABLE III

DEFINITION OF EACH $T_{i,neg}$ AND $T_{i,neg}$ USED IN EQ. (5).

i	$T_{i,pos}$	$T_{i,neg}$
1	C_7	0
2	$\sum_{i=1}^{6} P_i B_i \lvert C_i - C_7 \rvert$	$-\sum_{i=1}^{6} P_i (1 - B_i)(\lvert C_i - C_7 \rvert)$
3	$\sum_{i=1}^{5} \sum_{j=i+1}^{6} P_i P_j (1 - B_j)(\lvert C_j - C_7 \rvert)$	$-\sum_{i=1}^{5} \sum_{j=i+1}^{6} P_i P_j B_j (\lvert C_j - C_7 \rvert)$
4	$\sum_{i=1}^{4} \sum_{j=i+1}^{5} \sum_{k=j+1}^{6} P_i P_j P_k B_k (\lvert C_k - C_7 \rvert)$	$-\sum_{i=1}^{4} \sum_{j=i+1}^{5} \sum_{k=j+1}^{6} P_i P_j P_k (1 - B_k)(\lvert C_k - C_7 \rvert)$
5	$\sum_{i=1}^{3} \sum_{j=i+1}^{4} \sum_{k=j+1}^{5} \sum_{l=k+1}^{6} P_i P_j P_k P_l (1 - B_l)(\lvert C_l - C_7 \rvert)$	$-\sum_{i=1}^{3} \sum_{j=i+1}^{4} \sum_{k=j+1}^{5} \sum_{l=k+1}^{6} P_i P_j P_k P_l B_l (\lvert C_l - C_7 \rvert)$
6	$\sum_{i=1}^{2} \sum_{j=i+1}^{3} \sum_{k=j+1}^{4} \sum_{l=k+1}^{5} \sum_{m=l+1}^{6} P_i P_j P_k P_l P_m B_m (\lvert C_m - C_7 \rvert)$	$-\sum_{i=1}^{2} \sum_{j=i+1}^{3} \sum_{k=j+1}^{4} \sum_{l=k+1}^{5} \sum_{m=l+1}^{6} P_i P_j P_k P_l P_m (1 - B_m)(\lvert C_m - C_7 \rvert)$
7	$P_1 P_2 P_3 P_4 P_5 P_6 (1 - B_6)(\lvert C_6 - C_7 \rvert)$	$-P_1 P_2 P_3 P_4 P_5 P_6 B_6 (\lvert C_6 - C_7 \rvert)$

guaranteed to be non-positive. That is,

$$\begin{aligned} &P_{fail} = P_{fail,pos} + P_{fail,neg}, \ \textit{where} \\ &P_{fail,pos} = \sum_{i=1}^{7} T_{i,pos}, \ \ P_{fail,neg} = \sum_{i=1}^{7} T_{i,neg}, \\ &\textit{and } T_{i,neg}, \ T_{i,neg} \textit{ are defined as in Table III.} \end{aligned} \tag{5}$$

Given Eq. (5), it is trivial to come up with an over-estimation of P_{fail} (since under-approximation is unsafe) that is also monotonic in each P_i by negating all the negative terms, *i.e.*,

$$P_{fail\text{-}mono} = P_{fail} - \sum_{i=1}^{7} T_{i,neg} = \sum_{i=1}^{7} T_{i,pos}. \tag{6}$$

The aforementioned procedure can be generalized for more number of switches, which is the case for *line* and *ring* topologies. In particular, with every extra switch in the route from Π_i to Π_k (recall that $M_{i,k}(\alpha)$ is the message being analyzed), we need to deal with one extra error probability term, and thus the definition of P_{fail} (Eq. (2)) would be updated accordingly. For example, with one additional switch, P_{fail} would be defined as follows:

$$\begin{aligned} &P_{fail} = T_1 + T_2 + T_3 + T_4 + T_5 + T_6 + T_7 + T_8, \\ &\textit{where } \ T_1 = C_8, \\ &T_2 = \sum_{i=1}^{7} P_i (C_i - C_8), \\ &T_3 = -\sum_{i=1}^{6} \sum_{j=i+1}^{7} P_i P_j (C_j - C_8), \\ &T_4 = \sum_{i=1}^{5} \sum_{j=i+1}^{6} \sum_{k=j+1}^{7} P_i P_j P_k (C_k - C_8), \\ &T_5 = -\sum_{i=1}^{4} \sum_{j=i+1}^{5} \sum_{k=j+1}^{6} \sum_{l=k+1}^{7} P_i P_j P_k P_l (C_l - C_8), \\ &T_6 = \sum_{i=1}^{3} \sum_{j=i+1}^{4} \sum_{k=j+1}^{5} \sum_{l=k+1}^{6} \sum_{m=l+1}^{7} P_i P_j P_k P_l P_m (C_m - C_8), \\ &T_7 = -\sum_{i=1}^{2} \sum_{j=i+1}^{3} \sum_{k=j+1}^{4} \sum_{l=k+1}^{5} \sum_{m=l+1}^{6} \sum_{n=m+1}^{7} \left(\begin{array}{c} P_i P_j P_k P_l P_m P_n \\ (C_m - C_8) \end{array} \right), \\ &\textit{and } \ T_8 = P_1 P_2 P_3 P_4 P_5 P_6 P_7 (C_7 - C_8). \end{aligned} \tag{7}$$