Axo: Tolerating Delay Faults in Real-Time Systems

Maaz Mohiuddin, Wajeb Saab, Simon Bliudze, Jean-Yves Le Boudec
School of Computer Science and Communication Systems
École Polytechnique Fédérale de Lausanne, Switzerland
{firstname.lastname}@epfl.ch

***Abstract*—We address delay faults: faults that cause a software component to take more time for completing an action than a given deadline. Such faults are particularly of interest in real-time mission-critical control applications that use general-purpose computing platforms to compute setpoints. A violation of real-time constraints associated with setpoints can result in failure. Existing benign and Byzantine fault-tolerance architectures do not tolerate delay faults. We discuss the challenges involved in tolerating such faults. Then, we list the requirements on the real-time systems that pave the way for our solution: Axo. We describe how Axo masks delay faults, and we conclude with open issues.**

I. Introduction

Real-time control applications (RTCAs) for electrical grids [1], [2], autonomous vehicles [3], and manufacturing processes [4] are among the many examples that use software-based controllers such as cRIO (from National Instruments), DAP server (from Alstom), and MGC600 (from ABB). Many of these applications are mission-critical: their failure can lead to serious damage.

In addition to the controller, the RTCA has sensors that report the state of the controlled process to the controller. The controller uses these *measurements* to compute *setpoints*. The computed setpoints are received by the process agents (PAs) that implement them through the actuators.

The setpoints are subject to strict real-time constraints. That is, they must be implemented within a deadline that, henceforth referred to as a *validity horizon* (τ), depends on the specifics of the RTCA and varies from tens of milliseconds [1] to a few hundred milliseconds [2]. Hence, a sufficiently large delay incurred by the controller due to software and/or hardware faults, or a delay in the transmission of the setpoint due to the network, can result in failure of the RTCA. Such faults are termed *delay faults*.

Tolerating delay faults involves masking any delays in the controller or the network from the PA, so that the PA continues to receive setpoints within a bounded delay. Delay faults fall under the category of Byzantine faults and are thus not tolerated by existing benign fault-tolerance architectures.

Also, all existing Byzantine fault-tolerance (BFT) architectures consider only the correctness of the setpoint, not the timing aspects. Furthermore, they hold a consensus between the replicas, which cannot guarantee termination within bounded delay in asynchronous systems, as shown by [5]. Therefore, though a BFT implementation can guarantee safety, it will be at the expense of availability. Therefore, safety needs to be provided without consensus. Lastly, BFT architectures require at least $2f + 1$ replicas for f faults [6].

Below, in Section II, we identify the prerequisites on an RTCA for tolerating delay faults. In Section III, we introduce Axo, a fault-tolerance architecture addressing delay faults, and describe the fault masking mechanism of Axo and the associated assumptions on the RTCA. In Sections IV and V, we present early results and open issues, respectively.

II. Prerequisites to tolerating delay faults

To mask delay faults, i.e., faults causing the setpoint to be delayed by more than a validity horizon (τ), the first requirement is for the RTCA to provide the knowledge of τ to the fault-tolerance layer. This can be statically configured with a prior understanding of the application in question.

Evaluating the delay at the PAs requires the controller and the PAs to have a common notion of time. This is achieved through GPS- or network-based time synchronization (e.g., PTP, NTP) that most real-time systems already have.

Evaluating the delay pre-supposes knowledge of the first time at which a setpoint was valid (t_0). Setpoints received at $t > t_0 + \tau$ are considered invalid. However, obtaining t_0 is non-trivial: if recording t_0 is a part of the setpoint computation, then delays in the computation cannot be masked.

III. Axo

Axo uses active replication of the controller in order to mask delay faults and requires $f+1$ replicas to mask f faults. Besides requiring fewer replicas than BFT, Axo also has the advantage of minimal delay being incurred by a correct replica.

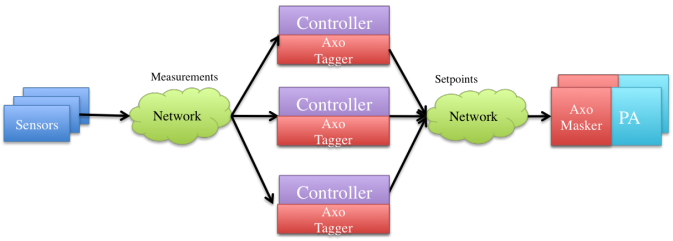

Fig. 1: Replication with Axo

As discussed in Section II, the RTCA is expected to have the notion of a validity horizon τ and to adopt a time synchronization protocol with a known accuracy δ_s.

In addition to the aforementioned prerequisites, Axo requires the PAs to handle duplicate setpoints received from controller replicas. The ability to handle duplicates is usually the case for RTCAs that use absolute, rather than differential, setpoints. For example, an electric-grid controller would instruct a battery agent that was injecting 8 kW to *set the injected power to 10 kW* rather than to *increase it by 2 kW*.

Last, we assume that the RTCA controller can be instrumented so as to record a timestamp (t_c) immediately before the computation of a setpoint, in a way that can be described by Algorithm 1.

Algorithm 1: Model of a controller to which Axo applies

```
1 while true do
2 |   t_c ← current time;
3 |   if ready to compute then
4 |   |   send t_c to Axo;
5 |   |   compute and issue setpoint(s);
6 |   end
7 end
```

As mentioned in Section II, t_0 is needed to evaluate the delay, and obtaining t_0 is non-trivial in the presence of delay faults. In Algorithm 1, lines 2 and 4 are added to the controller in order to obtain a $t_c < t_0$. Note that t_c is recorded before the condition *ready to compute* is verified. As we show in the full version of the work, obtaining t_c in this way allows Axo to mask all delay faults.

Axo is implemented as a separate fault-tolerance layer, with a *tagger* at each of the controller replicas and a *masker* at each PA, as shown in Figure 1. The tagger receives timestamps (t_c) from the controller and intercepts any setpoints issued by the controller to a PA. The tagger then sends to the masker of the destination PA a message consisting of both the setpoint and t_c, along with other information required to recreate the original packet. The masker receives these messages from the tagger and forwards the setpoint to its PA only if the current time is less than or equal to $t_c + \tau - \delta_s$.

IV. Early Results

We implemented a prototype of Axo for delay-fault masking. We tested its safety and availability with three controller replicas: C_1, C_2 (both virtual machines) and C_3 (physical machine). We also had a PA that received the setpoints and logged the delay. Our test-RTCA received measurements from a sensor every 100 ms, and slept to simulate computation time for a uniform random time between 2-7 ms when non-faulty and 7-15 ms when faulty. The faults were bursty with a rate of 1%. Here, the validity horizon is $\tau = 11$ ms.

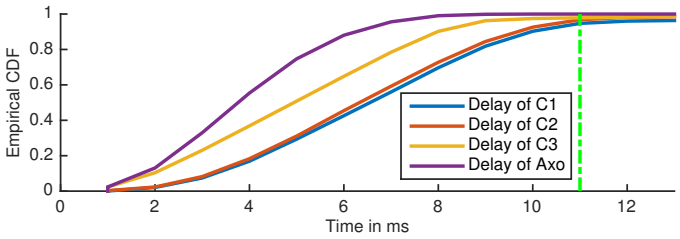

Fig. 2: Safety with Axo

Figure 2 shows the CDF of delays of C1, C2 and C3 at the masker and of the net delay of Axo at the PA. The delay at the PA is less than or equal to τ for all setpoints, thereby demonstrating the safety of Axo. We also observe the added benefit of active replication and Axo, as the delay of Axo is the minimum of the delays of C1, C2, and C3, thereby improving the real-time characteristics of the RTCA.

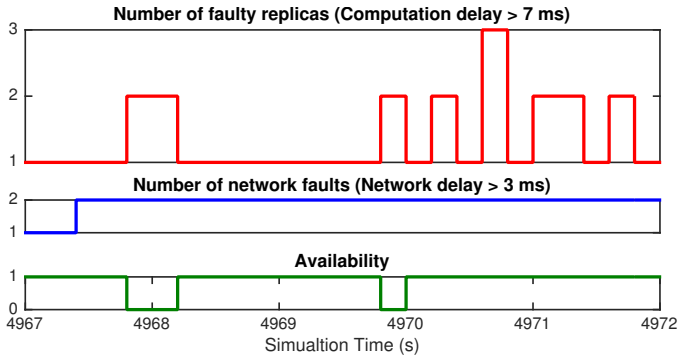

Fig. 3: Availability with Axo

Figure 3 shows the availability for different setpoints as a relation to the number of faults in the replicas and in the network, for an extract of the simulation scenario. In this experiment, a replica is considered faulty when its computation delay exceeds 7 ms, and a network fault is said to occur when the latency exceeds 3 ms. In the full version, we prove that Axo guarantees availability when the sum of the number of faulty replicas (f) and the number of network faults (n) is less than the number of replicas ($g = 3$), i.e., $f + n < g$, which is indeed verified here (and more, we see that Axo is available at certain instances when $f + n \geq 3$).

V. Conclusion and Future Work

We have discussed delay faults, a special class of Byzantine faults, primarily of interest to real-time mission-critical applications. We have presented the mechanism for masking such faults with Axo. In the full version of the work, we add the mechanisms for fault-detection and recovery. We also formalize the delay-fault model, under which we prove safety, availability and bounds on latency in masking, detection and recovery. Lastly, we will validate the fault-tolerance properties of Axo by testing it with Commelec [1], an RTCA for control of electrical grids.

References

[1] Andrey Bernstein, Lorenzo Reyes-Chamorro, Jean-Yves Le Boudec, and Mario Paolone. A Composable Method for Real-Time Control of Active Distribution Networks with Explicit Power Setpoints. Part I: Framework. *Electric Power Systems Research*, 125:254–264, 2015.

[2] Konstantina Christakou, D-C Tomozei, J-Y Le Boudec, and Mario Paolone. GECN: Primary Voltage Control for Active Distribution Networks via Real-Time Demand-Response. *Smart Grid, IEEE Transactions on*, 5(2):622–631, 2014.

[3] Tan Yew Teck, Mandar Chitre, and Prahlad Vadakkepat. Hierarchical Agent-Based Command and Control System for Autonomous Underwater Vehicles. In *Autonomous and Intelligent Systems (AIS), 2010 International Conference on*, pages 1–6. IEEE, 2010.

[4] Paulo Leitão. Agent-Based Distributed Manufacturing Control: A State-of-the-Art Survey. *Engineering Applications of Artificial Intelligence*, 22(7):979–991, 2009.

[5] Michael J Fischer, Nancy A Lynch, and Michael S Paterson. Impossibility of Distributed Consensus with One Faulty Process. *Journal of the ACM (JACM)*, 32(2):374–382, 1985.

[6] Rüdiger Kapitza, Johannes Behl, Christian Cachin, Tobias Distler, Simon Kuhnle, Seyed Vahid Mohammadi, Wolfgang Schröder-Preikschat, and Klaus Stengel. CheapBFT: Resource-Efficient Byzantine Fault Tolerance. In *Proceedings of the 7th ACM european conference on Computer Systems*, pages 295–308. ACM, 2012.