

Multi-level Fault Tolerance Approach for IoT Systems

Mário Melo[1(✉)] and Gibeon Aquino[2]

[1] Federal Institute of Rio Grande do Norte, Natal, Brazil
mario.melo@ifrn.edu.br
[2] Federal University of Rio Grande do Norte, Natal, Brazil
gibeon@dimap.ufrn.br

Abstract. The presence of faults is inevitable in the Internet of Things (IoT) systems. However, fault Tolerance in IoT systems is a challenge to overcome due to their size, complexity, dynamicity, and level of heterogeneity, especially error detection and error recovery and continued service. Thus, it is impractical to consider a unified fault tolerance technique for an entire system. The purpose of this study is to propose a multi-level fault tolerance approach granting interconnection among IoT system levels allowing information exchange and collaboration in order to attain the dependability property. Therefore, we define an event-driven framework called FaTEMa (Fault Tolerance Event Manager), creating a dedicated fault-related communication channel to propagate events among system levels. The implemented framework acted to assist error detection and continued service. Also, it offers extension points to support heterogeneous communication protocols and evolves to new capabilities. Our empirical results show that the FaTEMa introduction has improved error detection and error resolution time, consequently improving system availability. In addition, the use of FaTEMa demonstrates a reliability improvement and a reduction in the number of failures produced.

Keywords: Fault tolerance · Dependability · IoT · Multi-level

1 Introduction

Internet of Things has emerged as a framework of technologies and paradigms for the future of the internet [1]. The term Internet of Things (IoT) has attracted attention by projecting the vision of a global network of physical objects, enabling connectivity anytime, anyplace, for anything and anyone [10,11,16]. IoT made everyday life more pervasive, personal and intimate through technology [17]. However, this came with a price: failures on those systems can cause fatal accidents, undermining their trustworthiness in the public eye [19]. For this reason, the dependability of IoT systems has become crucial in many contexts, particularly in critical domains [4,20]. Although much progress has been made in the systems, hardware, approaches, and networks, several challenges still need to be appropriately addressed [18,23].

© Springer Nature Switzerland AG 2021
O. Gervasi et al. (Eds.): ICCSA 2021, LNCS 12957, pp. 421–436, 2021.
https://doi.org/10.1007/978-3-030-87013-3_32

Current implementations of FT mechanisms in IoT are static, tightly coupled, and inflexible. For example: (1) they are designed for a specific architecture and application [5,22]; (2) they do not scale beyond small (decentralized) solutions [12]; and (3) they provide solutions to specific faults, such as link failures [8], device failures [9]. These concerns are problematic because IoT systems are expected to continuously evolve to support new services, capabilities, and devices and add at runtime. Thus, FT mechanisms should evolve along with the system needs. Also, IoT FT mechanisms are concerned with handling faults at a single level point of view or do not raise concerns about heterogeneity and collaboration [2,3,5]. However, it is hard to adopt a unified, effective fault tolerance mechanism in complex and heterogeneous systems such as IoT systems.

The objective of the present study is to propose a novel mechanism that supports multi-level fault tolerance. This approach allows information exchange between different fault tolerance strategies, especially for error detection, error recovery, and continued service. The term "multi-level fault tolerance" indicates that two or more system abstraction levels pass information or are adapted to each other (at design time or runtime) to reduce error occurrence and error propagation through system levels. Thus, the goal is to minimize the effort in terms of hardware, software, and network resources through this information exchange and adaption compared to the case where the levels are optimized without being aware of fault tolerance events or applied means at the others levels.

We implemented our approach in the framework called FaTEMa (Fault Tolerance Event Manager). This framework uses an event-driven architecture providing extension points that enable ways to meet each level's inherent needs and enable heterogeneous intercommunication. In addition, it acts as a dedicated channel for the transversal communication of information regarding failures and their implications. Using an event-based architecture limits the system intrusion to send and receive information through events. Therefore, it does not act directly in the tolerance of the failures in the levels. To evaluate the FaTEMa feasibility, especially efficiency and effectiveness, we design an experiment to empirically compare a system implementation with a well-known fault tolerance technique using the multi-level fault tolerance approach and not using it.

Our experiment simulates a smart home system injecting two faults to measure and analyze each scenario regarding error detection and resolution, availability, failure occurrence, and error detection effectiveness. Our results indicate that using the multi-level fault tolerance approach has better efficiency and reached a greater degree of effectiveness. Consequently, there was an improvement in the system availability, which can improve the overall system dependability.

The contributions of this paper are summarized as follows:

- We introduce a novel mechanism that supports multi-level fault tolerance that supports information exchange between different fault tolerance strategies, especially for error detection, error recovery, and continued service.
- We implemented a novel framework called FaTEMa. This framework uses an event-driven architecture providing extension points to provide ways to meet

the inherent needs of each level and enable heterogeneous intercommunication.
– We report on an empirical study comparing a smart home system under two scenarios using and not using a multi-level fault tolerance approach.

The remainder of the article is organized as follows: in Sect. 2, we revise related works pointing its weakness and strengths. In Sect. 3, we present the novel multi-level fault tolerance approach and the FaTEMa framework in detail. Section 4, presents the results and discussion of the empirical evaluation of the proposed approach by comparing two smart home system scenarios, using the FaTEMa framework and not using it. Finally, we discuss the paper findings presenting future research gaps and concluding the paper in Sect. 5.

2 Related Work

IoT systems are used in a wide variety of domains. In some of these domains, the occurrence of failures is acceptable or causes only discomfort. In others, on the contrary, the damage can be disastrous. Thus, several works have contributed to the area, including those reported below directly related to this work.

In Su et al. [21], a fault tolerance mechanism for IoT middleware is developed. Their work aims to provide a way to overcome component failures by reaching the tolerance levels defined by the developers. The proposed mechanism can detect, recover from failures, and reconfigure the system autonomously. The mechanism uses each node to replicate several components to guarantee these characteristics and uses a monitoring chain through heartbeat messages from each node. Thus, when a failure occurs, the node that perceives the failure informs the others so that everyone keeps their replicas consistent and ends by reconfiguring the nodes and removing dead nodes from monitoring.

Woo et al. [22] proposes a layered IoT system for personal health devices based on the oneM2M communication protocol. To guarantee the reliability of the proposed solution, the authors define an algorithm for fault tolerance. This algorithm acts on the gateways present in each layer of the system. The same layer's gateways form a chain, and each of them has a backup copy of the previous gateway in the chain, forming a fault tolerance strategy in each layer. Furthermore, the upper layer gateways are connected to the lower layer, thus managing the lower layer gateways.

The proposal by Li et al. [14] aims to solve the problem of complexity in the management of end-to-end communication failures in IoT. To this end, the authors propose a flexible, feasible, and layered approach to perform fault detection, localization, and correction efficiently. Fault management is divided into two layers. The upper layer is responsible for adaptively detecting and locating faults using observers at different transmission networks. These observers collect information about traffic and abnormal events, so prediction models are introduced (cognitive map fuzzy and semi-Markov). The lower layer executes the failure recovery strategy. This strategy varies according to each type of network used in the system.

In the work of Celesti et al. [2], a watchdog service for IoT monitoring is proposed, which allows checking the status of microservice applications, detecting faults, and repairing them. This service operates on each IoT device, synchronizing with each other using rsync and Network Time Protocol (NTP). Henceforth, microservices have replicas spread over network nodes when a watchdog detects a failure, the instance that is failing is removed, and a new replica is started on another device. Data from each device and microservice are stored on a shared database in the cloud. This database is available for all resources.

Javed et al. [7] propose an architecture for the edge and cloud designed to tolerate failures, called CEFIoT. This architecture's main objective is to enable fault tolerance at the edge nodes, especially when processing operations are moved from the cloud to the edge. Thus, there must be local strategies to preserve the system's state, especially in scenarios of node failure or connectivity problems. The proposed architecture is divided into three layers. The first layer is application isolation, where individual containerized processes in a single application from source to destination. The data is transported through a publish/subscribe framework in the second layer, whose data is replicated through the architecture nodes. At last, the cluster management layer responsible for monitoring and managing the operations of the previous two layers. Thus, to guarantee fault tolerance, data replication is used to process this information in several edge devices, reconfiguring whenever a failure occurs.

3 Multi-level Fault Tolerance Approach

The multi-level fault tolerance approach provides a communication channel that allows the exchange of events related to dependability and fault tolerance aspects. This dedicated channel should not interfere with the functioning of the system, assisting and allowing the collaboration of the different fault tolerance strategies to obtain an increase in the dependability of the systems. To provide this intercommunication channel, we develop a framework called FaTEMa (Fault Tolerance Event Manager) that consists of a set of extensible components to handle external fault tolerance events and propagate them among levels of an IoT system. It acts as a mediator, enabling communication between IoT system levels to assist fault tolerance mechanisms, aiming for failure avoidance. Thus, it facilitates information exchange between local or distributed techniques providing interoperability between them to improve systems' dependability.

By general means, fault tolerance can be implemented to prevent faults from leading to system failures. Fault tolerance has four constituent phases: error detection, damage confinement and assessment, error recovery, fault treatment, and continued system service [13]. The FaTEMa focused on assisting error detection and fault treatment techniques and continued system service, especially resuming a system to normal service. The success of any fault tolerance technique depends on the effectiveness with which it can detect errors. Thus, the initial stage of any technique is the error detection step. Therefore, the more errors are detected, the better system dependability will be. Once the errors are

detected, the appropriate techniques can be applied to prevent the system from failing. For this reason, the use of FaTEMa allows the propagation of information regarding error detection allowing different levels and techniques to act in order to recover from those errors, which could eventually avoid a system failure.

Once an error is detected and recovered, it is necessary to return a system to its normal operation. Consequently, components could be resumed or restart, others can be reconfigured, and some can be terminated. Thus, FaTEMa acts to assist continued service techniques to communicate to other entities and levels measures to the resumption of an IoT system's normal operation.

However, FaTEMa, by itself, does not contain any fault tolerance mechanism. The FaTEMa framework's vision is to enable intercommunication between the various levels of abstraction, enabling efficient and effective error detection and continued service, improving an IoT system's dependability. Also, the proposed framework aims to be a technology, hardware, or operating system agnostic, facilitating the development of new solutions for fault tolerance.

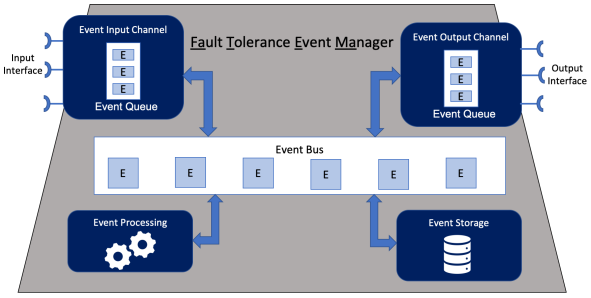


Fig. 1. Fault Tolerance Event Manager (FaTEMa) architecture

3.1 Architecture

Figure 1 depicts FaTEMa framework architecture. This framework has five components: Event Input Channel, Event Processing, Event Storage, Event Output Channel, and Event Bus. Each component acts independently, separating responsibilities. The communication between components occurs through the use of asynchronous messages posted and received at the EventBus. This interaction between the components and the EventBus occurs using the Publish-Subscribe messaging pattern. Thus, the components subscribe on the bus to certain types of events. Once these events are detected, the component is notified and receives the specific event. Also, each component can send events to the EventBus. However, there are two special cases: EventInput Channel and EventOutput Channel. The first one can only post external events received at the entrance of the framework in the bus. The second is only allowed to receive events posted on the bus and make them available externally. These events are triggered and consumed by the local fault tolerance strategies.

Additionally, each component has an interface that makes it possible to implement new behaviors. The framework uses the Observer pattern allowing the extensions to indicate in what are the events of interest. Once the component receives an event, it verifies if there is a valid observer extension point. If it is true, the event is passed to the extension point.

Event Input Channel: Acts as an event broker, providing an external communication interface to receive events. In order to support heterogeneous communication, the component provides an extensible interface allowing the implementation of different communication protocols. However, due to the heterogeneity of technologies in IoT, this component defines a standard event ensuring interoperability between protocols.

Event Processing: This component acts by providing event processing capabilities. It supports extensions that register interest in specific events to perform its computation. These processing rules must avoid using specific systems processing rules since the FaTEMa is a distributed component by nature. Thus, rules regarding system-specific fault tolerance (error detection and recovery) should not be placed at FaTEMa components. Some examples of processing are device registration, system metrics, system state, FaTEMa fault tolerance, among others.

Event Storage: It is responsible for store e recover data from and for the event processing. The data can be stored locally or distributed. The size should be set according to the domain or system need. However, it is not a mandatory component once certain devices could have processing and storage limitations.

Event Output Channel: It is responsible for providing events to the external environment. Like the Event Input Channel component, this component provides an extensible interface to support different communication protocols. The data sent follows the event structured, and the receiver must treat them according. Thus, support heterogeneous technologies and provides interoperability between them, creating a multi-level communication channel for fault tolerance.

3.2 FaTEMa Operation

The proposed framework creates a "virtual" communication channel between the different levels of abstraction, as depicted in Fig. 2. This channel acts by facilitating information exchange through local fault tolerance mechanisms (Local FT) present in system levels. Each level is subjected to specific types of uncertainty and different types of faults that have to be detected and recovered, ideally in a coordinated manner. Local fault tolerance mechanisms are responsible for handle faults in a specific level or scenario without shares a global system vision. Additionally, as IoT systems grow in size, complexity, speed, and heterogeneity level, it becomes impractical to assume that a single fault tolerance mechanism can access the entire system's information. Therefore, the FaTEMa integrates system levels enabling information exchange between fault tolerance approaches.

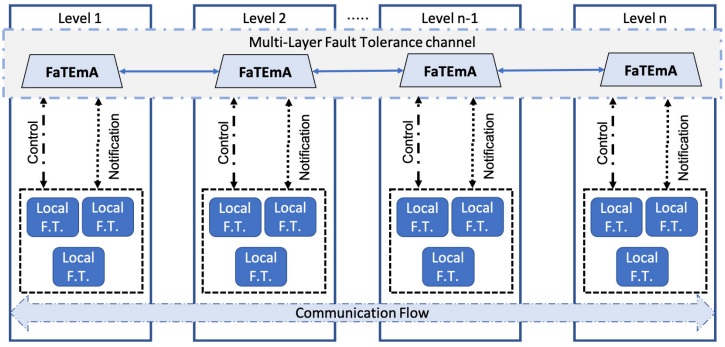


Fig. 2. Generic IoT system using the FaTEMa and multi-level fault tolerance channel

The framework acts to reduce the complexity in the implementation of multi-level fault tolerance techniques. Although the complexity introduced by these techniques could decrease system dependability, the FaTEMa is designed does not reduce, rather than increase, the dependability. The framework does not directly interfere in the functioning of fault tolerance techniques. It enables distributed intercommunication between fault tolerance techniques through its communication channel. On this channel, there are two types of generic events: Control and Notification. Control Event messages are related to fault-tolerance measures and treatments. Notification Event messages carry information about the system's current state, system administrative information, or metrics regarding system utilization information.

The flow of communication occurs in both directions. Higher-level fault tolerance techniques can communicate with lower levels and vice versa. Also, FT techniques can use the framework to collaborate in the measures against errors at the same level. However, to enable this intercommunication, there must be at least one FaTEMa instance at each level. FaTEMa allows instance replication in order to assure availability and avoid single failure points. Each instance acts isolated. However, it is possible to configure a distributed database to replicate the stored information between instances. These functionalities are available by default. Furthermore, all internal or external communication carried out by FaTEMa is done asynchronously. Thus, to send or receive FaTEMa events, each local FT must register in the framework's input and output channels.

4 Experiment Evaluation

Aiming to evaluate the framework's efficiency and effectiveness to assist error detection and continued service, we implemented a smart home system prototype. This system was designed to involve monitoring various devices in a home, such as security cameras, temperature, pressure, humidity sensors, various equipment (such as TVs, microwaves, refrigerators).

The system architecture consists of four levels of abstraction. The perception level is directly related to the device monitoring and actuation in the smart home. This level contains a set of appliances connected to the monitored devices making it possible to manage them. All information generated or obtained by the house devices flows through a gateway that concentrates the information of a certain set of houses, connecting them to the upper levels. The service level provides a cloud middleware that enables the treatment, processing, and storage of a high volume of information. Finally, a mobile app that interacts with the devices managing them is available at the application level.

The smart home contains several devices (D) sensing and acting over the environment in order to facilitate a user life experience. These devices are linked to an appliance (AP) responsible for monitoring and connection distribution. The appliances are responsible for having a physical connection with devices providing efficient communication with the rest of the system. The access to devices and appliances must happen through a gateway (GW) responsible for centralizing the information of several instances of houses and transferring them to the upper levels. In addition, it receives requests directed to devices and identifies which paths the requests will follow. The Cloud middleware (CM) has a high computational power to process and store all the information transferred. Furthermore, it implements fault tolerance mechanisms in order to prevent system failures. The mobile application (MA) interacts with the CM, sending and receiving information to the devices.

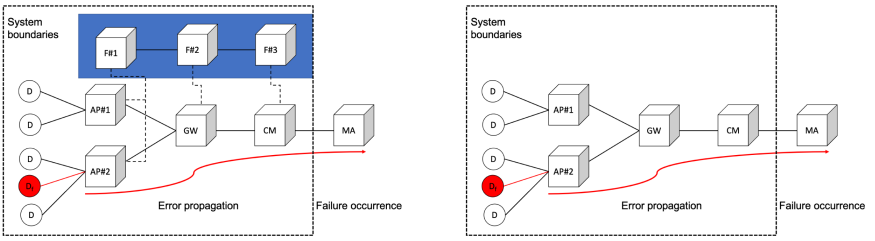
In this execution, an omission fault was injected into a device (temperature sensor), which results in an error when activated. This experiment evaluates the efficiency of detecting error occurrences and their efficiency in decreasing failure occurrences using the proposed approach. Also, it is essential to assess the impact that the use of the framework will have on the system's general use. To attain these objectives, we define two questions to answer.

RQ1 – An IoT system using the multi-level fault tolerance approach is more efficient in detecting an error and resume the system to its normal operation than a system using fault tolerance techniques at each level?
RQ2 – An IoT system using the multi-level fault tolerance approach is more effective than a system using fault tolerance techniques at each level?

Efficiency in this context is measured by error detection time and error resolution time. The smaller the time to detect and resolve errors are, the more efficient it is considered. Effectiveness is measured by the rate of failures (Failure per second), Handled Error Rate (Total of Errors per Total of Failures), and Detected Error Rate (Total of Detected Errors per Total Errors). The effectiveness successfully achieves the desired results, reduces the number of failures, and leads to improved dependability attributes.

4.1 Experiment Design

In order to carry out this evaluation, we compared the use of the system from two scenarios, Scenario #1 (S#1) - using FaTEMa and the multi-level fault tolerance



(a) Scenario #1 – Using FaTEMa (b) Scenario #2 – Not using FaTEMa

Fig. 3. Experiment scenarios

channel and Scenario #2 (S#2) - not using FaTEMa. Both scenarios use the exact implementation and fault tolerance strategies. The fault tolerance (FT) mechanism is activated when they obtain the information in response to a request to an unavailable device (Df). The implemented FT mechanism was the Bounded Retries [6]. This mechanism deals with transient errors by restricting the number of retries to a faulty entity. It typically takes an exponential backoff function to control the retrying process to avoid massives requests to fail. Then, the CM starts an asynchronous check using a retry strategy with exponential backoff, checking if the unavailable device is active again. While the device is unavailable, incoming requests to the faulty device are intercepted and returned, informing its unavailability in order to avoid failure occurrence. When the asynchronous check detects the faulty device has resumed its continuous operation, the retry process is interrupted, and CM stops intercepting requests to the unavailable device (Fig. 3).

In both scenarios, the APs are responsible for identifying that the device has failed. Therefore, they perform device monitoring in order to identify possible faults. If a fault is identified in a given device, the subsequent request is returned to the sender with the device's unavailability information. Once the APs identify that the device has returned to continued service, the following response is carried out normally. In scenario S#1, in addition to the described fault notification, when an AP identifies that a device is not working correctly, a control event is sent to FaTEMa. Likewise, if a device that was not in regular operation resumes service continuation, a control event is sent to FaTEMA.

4.2 Experiment Execution

In order to answer the research questions, this experiment injects two omission faults in the system at specifics intervals. The faults occur in a device (temperature sensor) connected to AP#2, occurring according to the following pair (start, duration): Fault Activation 1 (15,15) and Fault Activation 2 (40,30). The start time means the instant of fault activation, initiating the omission errors. The duration is the amount of time the error occurrence lasts (both expressed in seconds). The values of request timeout and the maximum throughput were

200 ms and 20 req/s, respectively. At the beginning of the test, a setup phase was executed to minimize the resource initialization effects. This procedure took approximately 5 s, and only after this, the simulated mobile application started to send requests for 70 s. Each scenario was executed 12 times, and in each one, all requests are randomly generated. Thus, measures obtained and analyzed are the average of these executions.

4.3 Efficiency Results

The detection of an error occurs when the manifestation of a fault is detected [13]. In this study, an erroneous state is considered detected when is detected by the CM. The same principle is applied to error resolution that a system will be considered error-free when the CM enables the system to continue providing its services. Table 1 presents measurements obtained regarding error detection and error resolution time. The average error detection time in scenario 1 resulted in over 30% higher than scenario 2. It is possible to notice a significant difference in the detection time, greater than 25%, for the two fault injections.

Table 1. Error detection time and error resolution time measurements

		Metrics	
		Error detection (ms)	Error resolution (ms)
S#1	Fault injection 1	667.42	637.83
	Fault injection 2	648.08	601.25
	Average	657.75	619.54
S#2	Fault injection 1	727.50	3573.63
	Fault injection 2	691.58	8804.72
	Average	709.54	6189.18

Similarly, the average error resolution time in scenario 1 obtained a result 90% lower than scenario 2. In this case, the system returns to its regular operation 90% faster in scenario 1 when compared to scenario 2. These results indicate a better efficiency in detect errors and resuming the system's continuous operation in scenario 1. This assertion is mainly supported since the communication channel used by the FaTEMas for propagating information does not require active CM interaction. Once the recovery event is launched on the Multi-Level Fault Tolerance Channel, all interested parties, regardless of level, will receive the information passively. Thus, in scenario 1, the faulty level actively propagates F.T. information, contrary to scenario 2, where the C.M. actively requests the appliances for error detection and error resolution.

The availability of a system can be defined as the fraction of time this system is in a functioning condition [15]. In this study, we measure the Df availability adopted the ratio TA/TE, where TA means the amount of time the device is

available and operating according to its specifications, and TE the duration of the experiment. Independent of the scenarios, the Df availability was fixed based on the experiment configuration (TA = 70 − 10 − 20, TE = 70, and consequently, availability = 57.14%). However, CM availability is influenced by the F.T. mechanism applied. Thus, it is necessary to consider error detection time and error resolution time to calculate system availability. Thus, we adopted the ratio TAs = (TA − TD − TR)/Te, where TAs is the system availability, TD is the sum of the errors detection time, Tr = sum of the errors resolution time. Table 2 shows the system availability in each scenario. In S#1, the system availability is 86,12% higher than in S#2. Hence, it indicates that the use of FaTEMa provided a superior effective system availability, demonstrating that S#1 performs more efficiently compared to S#2. The error detection time and error resolution time metrics also support this assertion.

Table 2. Measurements related to device and system availability for each scenario

Scenario	Metrics					
	TE (s)	TA (s)	DA	TD (s)	TR (s)	TSa
S#1	70	40	57.14%	1.31	1.23	53.51%
S#2				1.41	12.37	37.46%

4.4 Effectiveness Results

To assess the effectiveness of using FaTEMa mechanisms combined with the fault tolerance strategies, we collect measurements regarding the requests received at the C.M. and classifies them according to the produced response. In other words, the effectiveness of using FaTEMa as a dedicated channel for fault tolerance information was assessed under the C.M. perspective. Thus, if a request sent to the C.M. does not produce one of the expected responses, a failure occurs. The C.M. is the component responsible for protecting the system from failing, implementing fault tolerance strategies. Moreover, produce responses that indicate that the system cannot respond to the request at the moment, informing that the selected resource is unavailable or successfully provides the selected resource.

Table 3 presents the collected measurement for the C.M. responses from the mobile application requests. The average of total requests sent to C.M. resulting in a difference of about 1.20%. This difference was expected given the experiment's random nature, and it is unlikely that the same amount of requests were produced and responded. However, this metric points out that despite FaTEMa introduction at each system level in S#1, that there was a low impact on the overall system functioning. Nevertheless, S#1 successful responses were 77.41% higher than S#2. Also, it produces about 31% fewer failure responses. As for the total of handled requests, S#2 produces more than 36% responses compared to S#1. This difference is explained due to the higher error resolution time measured in S#2 compared to S#1.

Figure 4 represents the average of requests responded by the C.M. over experiment time. There is a considerable difference in the error detection time and the number of failures produced until the system responds adequately to that failure when the scenarios are compared, shown in Fig. 4b. Also, the two fault injections produce similar results, the number of failures produced and error detection time for both scenarios. Thus, this result indicates that the adoption of FaTEMa provided a more efficient response to the injection of a fault, reducing the error detection, leading to a reduction in the number of failures produced by the system. Similar behavior is identified in the handled requests as shown in Fig. 4c, in which S#1 identifies error recovery occurrence has been corrected firstly compared to S#2. This difference between scenarios was produced because instead of waiting for the tolerance strategy identifies that the requested resource is available again, the use of FaTEMa in S#1 allowed a proactive sharing of fault tolerance and recovery events, allowing without a need for new requests to handle failures or continued service. This situation directly impacts the successful responses (Fig. 4a) because while the CM does not deactivate the FT mechanisms, the responses produced will continue to indicate that there is still an error in progress.

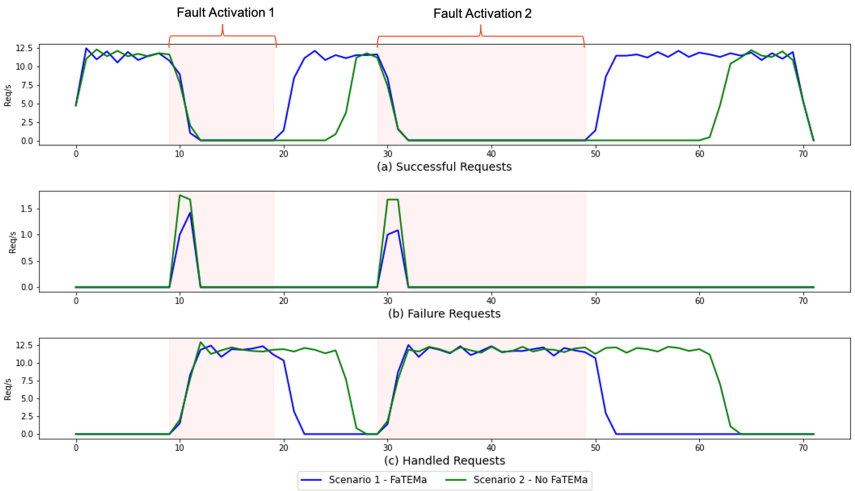


Fig. 4. C.M. responses over time

The impact of FaTEMa usage requires the use of metrics related to dependability property. We establish the measures of Failure Rate defined by the total of failures over second, Handled Error Rate is defined as the total of failures over a total of errors, and Detection Error Rate which is adopted as the total of detected errors over a total of errors, which are typical metrics to assess reliability and fault tolerance effectiveness. Regarding the Failure Rate metric, S#1

demonstrates an inferior occurrence of failures per unit of time than S#2, having a rate of 31,52% fewer failures than S#2. In other words, 93,88% of error requests produces failures. Consequently, the use of FaTEMa demonstrates a positive effect on the system reliability, and consequently, its dependability. The Handled Error Rate and Detection Error Rate metrics are complementary and demonstrate how effective a fault tolerance strategy concerns the total of failures and detected errors. The use of FaTEMa (S#1) provided a decrease of about 14% in the number of failures produced and increased the Detection Error Rate by more than 20% compared to S#2.

Table 3. Measurements related to the responses sent by the C.M. to the mobile application during the experiment for all scenarios.

Metric	Scenarios	
	S#1	S#2
Total	895.83	906.75
Successful	497.33	280.33
Handled	393.25	618.75
Error	6.58	8.17
Failures	5.25	7.67
Detected error	1.33	0.50
Failure rate	0.08	0.11
Handled error rate	79.75%	93.88%
Detected error rate	25.40%	6.52%

4.5 Discussion

The FaTEMa introduction allowed the information propagation related to fault through the levels of the system detailed in the experiment. In addition, the usage of FaTEMa demonstrates to assist in identifying the occurrence of an error and returning the system to the continued service. A highlighted aspect perceived by the results is that the introduction of FaTEMa made it possible to use proactive communication in error detection and resolution stages without adding additional complexity to the system. Thus, the reactive communication model for detecting and resolving errors remained active but could be combined with that one. Therefore, the evidence collected and presented in Tables 2 and 3 reveals that with the multi-level fault tolerance strategy, it was possible to reduce the time to detect the fault activation by 7.8% and reduce the time taken to propagate an error resolved 93.28%. Consequently, the use of FaTEMa improved system availability by about 86%. In just 3.63% of the time that the error was happening, the system was responding erroneously. This time represents about 2.5 s of the total experiment to propagate FT information regarding error detection and error resolution. Thus, the FaTEMa introduction contributed to the

error detection and resolution to operate quickly and efficiently in an organized way, increasing system dependability.

Related to the effectiveness of using a multi-level fault-tolerance approach is possible to verify a reduction of more than 31% in the number of failures in S#1 when compared to the use of the fault tolerance strategy with only local treatment (S#2). In addition, we found that the average error detection rate was 2.5× higher in S#1, which demonstrates the importance of error detection occurring as promptly as possible. Because lower detection time tends to be, it could reduce the number of errors and failures. This fact is directly linked to the rate of failures, which is a well-established metric used to assess the reliability of a system. Thus, S#1 reduced the failure rate by about 27%, leading to an improvement in reliability and dependability. In addition, S#1 revealed that the use of FaTEMa in S#1 improved the rate of errors handled, meaning that they did not become failures at a rate greater than 14% compared to S#2. In the same way, in S#1, there was a substantial increase, more than 4x, the rate of errors detected concerning the total errors produced by the system. Therefore, the use of FaTEMa is considered more effective because they successfully achieved an increase in system reliability, reducing the failure rate and increasing error detection.

Although the use of the communication channel for multi-level fault tolerance introduces an increase in CPU processing and network bandwidth, no negative impact was perceived on the system's overall functioning. On the contrary, the average difference in the total number of requests answered between scenarios is 1.2%. This difference can be explained due to the difference of more than 2x responding successful requests by S#1. Successful requests require more time between source-destination-source as they need to go through the entire system. Also, the results presented in this experiment demonstrate the efficiency and effectiveness of using a dedicated channel for transmitting events related to fault tolerance, only two scenarios were considered. However, it was evidenced that the exchange of collaborative messages about tolerance between the levels of the IoT system can help and improve dependability attributes. By creating a channel that permeates all levels, solutions can collaborate, promoting end-to-end fault tolerance within an IoT system.

5 Conclusion and Future Work

This paper proposes a framework to enable the propagation of fault tolerance events between system levels. It acts as a mediator, enabling communication to collaborate between levels and its local fault tolerance aiming to improve the steps of error detection and continued service. Thus, the FaTEMa focused on assisting error detection and fault treatment and continued service, especially to resuming a system to normal service. Because the success of any fault tolerance technique depends on its effectiveness, it can detect errors. This multi-layer approach provides an end-to-end communication channel that facilitates FT event

exchange to minimize fault occurrence and propagation effects. Also, the framework facilitates system levels interoperability by providing extensible interfaces for heterogeneous communication protocols.

The empirical evaluation was accomplished by conducting an experiment to compare two scenarios: a system using FaTEMa and the multi-level fault tolerance approach; and a system not using FaTEMa. The implemented system used in the experiment uses a well-known fault tolerance approach and uses the exact implementation in both scenarios. Overall, the results show that the scenario using the multi-level fault tolerance had a better efficiency in error detection and error resolution. Also, it was observed that in the scenario using FaTEMa, the system had a greater availability when compared to the other scenario leading to an increase in dependability attributes. This behavior was expected because once an error is detected as soon as possible, it will be conceivable to activate measures to tolerate and recover faults. Equally important is the time to resume system operation once a fault is recovered. The use of FaTEMa demonstrated a high capacity to propagate the recovery event between levels, enabling a more efficient resumption of the system's normal functioning, leading to improved availability.

As future work, we plan to extend the empirical evaluation to a real-world scenario. Furthermore, the present study investigated the applicability e effectivity of fault tolerance multi-level in one IoT System. However, IoT is an ecosystem, and thus, we plan to extend the study to investigate the feasibility of applying a multi-systems fault tolerance approach. Also, we plan to expand FaTEMa event processing to handle complex events and contextual information.

References

1. Atzori, L., Iera, A., Morabito, G.: The internet of things: a survey. Comput. Netw. **54**(15), 2787–2805 (2010)
2. Celesti, A., Carnevale, L., Galletta, A., Fazio, M., Villari, M.: A watchdog service making container-based micro-services reliable in IoT clouds. In: 2017 IEEE 5th International Conference on Future Internet of Things and Cloud (FiCloud), pp. 372–378. IEEE (2017)
3. Choi, J., Jeoung, H., Kim, J., Ko, Y., Jung, W., Kim, H., Kim, J.: Detecting and identifying faulty IoT devices in smart home with context extraction. In: 2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pp. 610–621. IEEE (2018)
4. Ciccozzi, F., Crnkovic, I., Di Ruscio, D., Malavolta, I., Pelliccione, P., Spalazzese, R.: Model-driven engineering for mission-critical IoT systems. IEEE Softw. **34**(1), 46–53 (2017)
5. Gia, T.N., Rahmani, A.M., Westerlund, T., Liljeberg, P., Tenhunen, H.: Fault tolerant and scalable IoT-based architecture for health monitoring. In: 2015 IEEE Sensors Applications Symposium (SAS), pp. 1–6. IEEE (2015)
6. Homer, A., Sharp, J., Brader, L., Narumoto, M., Swanson, T.: Cloud Design Patterns: Prescriptive Architecture Guidance for Cloud Applications. Microsoft Patterns & Practices, CreateSpace, Scotts Valley (2014)

7. Javed, A., Heljanko, K., Buda, A., Främling, K.: CEFIoT: a fault-tolerant IoT architecture for edge and cloud. In: 2018 IEEE 4th World Forum on Internet of Things (WF-IoT), pp. 813–818. IEEE (2018)
8. Karthikeya, S.A., Vijeth, J., Murthy, C.S.R.: Leveraging solution-specific gateways for cost-effective and fault-tolerant IoT networking. In: 2016 IEEE Wireless Communications and Networking Conference, pp. 1–6. IEEE (2016)
9. Khan, F.I., Hameed, S.: Understanding security requirements and challenges in Internet of Things (IoTs): a review. arXiv preprint arXiv:1808.10529 (2018)
10. Kosmatos, E.A., Tselikas, N.D., Boucouvalas, A.C.: Integrating RFIDs and smart objects into a unified internet of things architecture. Adv. Internet Things **1**(01), 5 (2011)
11. Kubler, S., Främling, K., Buda, A.: A standardized approach to deal with firewall and mobility policies in the IoT. Pervasive Mob. Comput. **20**, 100–114 (2015)
12. La Marra, A., Martinelli, F., Mori, P., Saracino, A.: Implementing usage control in internet of things: a smart home use case. In: 2017 IEEE Trustcom/BigDataSE/ICESS, pp. 1056–1063. IEEE (2017)
13. Lee, P.A., Anderson, T.: Fault tolerance. In: Fault Tolerance. Dependable Computing and Fault-Tolerant Systems, vol. 3, pp. 51–77. Springer, Vienna (1990). https://doi.org/10.1007/978-3-7091-8990-0_3
14. Li, X., Ji, H., Li, Y.: Layered fault management scheme for end-to-end transmission in internet of things. Mob. Netw. Appl. **18**(2), 195–205 (2013)
15. Lyu, M.R.: Software reliability engineering: a roadmap. In: Future of Software Engineering (FOSE 2007), pp. 153–170. IEEE (2007)
16. Madakam, S., Ramaswamy, R., Tripathi, S.: Internet of things (IoT): a literature review. J. Comput. Commun. **3**(05), 10 (2015)
17. Miraz, M.H., Ali, M., Excell, P.S., Picking, R.: A review on Internet of Things (IoT), Internet of Everything (IoE) and Internet of Nano Things (IoNT). In: 2015 Internet Technologies and Applications (ITA), pp. 219–224 (2015). https://doi.org/10.1109/ITechA.2015.7317398
18. Ojie, E., Pereira, E.: Exploring dependability issues in IoT applications. In: Proceedings of the Second International Conference on Internet of things, Data and Cloud Computing, p. 123. ACM (2017)
19. Ratasich, D., Khalid, F., Geißler, F., Grosu, R., Shafique, M., Bartocci, E.: A roadmap toward the resilient internet of things for cyber-physical systems. IEEE Access **7**, 13260–13283 (2019). https://doi.org/10.1109/ACCESS.2019.2891969
20. Ştefan, V.K., Otto, P., Alexandrina, P.M.: Considerations regarding the dependability of internet of things. In: 2017 14th International Conference on Engineering of Modern Electric Systems (EMES), pp. 145–148. IEEE (2017)
21. Su, P.H., Shih, C.S., Hsu, J.Y.J., Lin, K.J., Wang, Y.C.: Decentralized fault tolerance mechanism for intelligent IoT/M2M middleware. In: 2014 IEEE World Forum on Internet of Things (WF-IoT), pp. 45–50. IEEE (2014)
22. Woo, M.W., Lee, J., Park, K.: A reliable IoT system for personal healthcare devices. Futur. Gener. Comput. Syst. **78**, 626–640 (2018)
23. Xing, L., Zhao, G., Wang, Y., Mandava, L.: Competing failure analysis in IoT systems with cascading functional dependence. In: 2018 Annual Reliability and Maintainability Symposium (RAMS), pp. 1–6. IEEE (2018)