A Microservices Architecture for Reactive and Proactive Fault Tolerance in IoT Systems

Alexander Power and Gerald Kotonya
School of Computing and Communications
Lancaster University
Lancaster, United Kingdom
{a.power3, g.kotonya}@lancaster.ac.uk

***Abstract*—Providing fault-tolerance (FT) support to Internet of Things (IoT) systems is an open challenge, with many implementations providing static, tightly coupled FT support that does not adapt and evolve like IoT systems do. This paper proposes a pluggable framework based on a microservices architecture that implements FT support as two complementary microservices: one that uses complex event processing for real-time FT detection, and another that uses online machine learning to detect fault patterns and pre-emptively mitigate faults before they are activated. We provide an early evaluation of how our framework can handle a real-world scenario.**

***Index Terms*—internet of things, fault tolerance, microservices, complex event processing, machine learning**

I. Introduction

The Internet of Things (IoT) is a vision in which the Internet extends to everyday objects as a means of bridging the gap between the physical and virtual worlds [1]. This opens up the Internet to a vast network of interconnected "smart" objects that not only harvest information from their environment and interact with the physical world, but also adopt existing Internet standards to provide services for information transfer, analytics, and communications [2].

Fundamentally, IoT is driven by *data*, whether exchanged between devices or services across the Internet. Therefore, providing a dependable infrastructure for the billions of expected IoT devices is an important challenge [3], [4]. For an IoT system to be dependable, it must deliver service that can justifiably be trusted, encompassing attributes such as availability, reliability, safety, and maintainability, where *reliability* is a high-priority goal to address for IoT solutions concerned with quality of service (QoS) [5].

Dependability is threatened by faults and errors that contribute to the occurrence of service failures, where a system can no longer provide its service as intended [6]. Mitigating faults can be accomplished by three approaches, namely [7]:

- **Fault avoidance**: avoiding error introduction at the design and programming stage to minimize the number of faults introduced into the system.
- **Fault detection and correction**: using verification and validation to remove faults before deployment.
- **Fault tolerance**: designing the system to detect and recover from faults during runtime to prevent failures.

978-1-5386-4725-7/18/$31.00 ©2018 Crown

While these approaches are necessary for developing resilient systems, *fault tolerance* (FT) is especially challenging in the IoT domain for several reasons.

Firstly, as IoT systems are distributed, they suffer from failures similar to other distributed systems, namely: (1) *crash* failures, where the server halts and requires a restart; (2) *omission* failures, where the server stops sending and receiving messages; (3) *timing* failures, where a server's response is too early or too late; (4) *response* failures, where the server's response value or state transition that takes place is incorrect; and (5) *arbitrary* failures, where the root cause is unclear [8].

Secondly, the above failures are exacerbated because IoT devices are typically constrained (in terms of energy, computing power, and resources) and rely upon wireless communication. This limits their ability to survive 'in the wild' or perform complex recovery strategies when faults manifest, meaning that FT is typically delegated to some external and more reliable entities, such as the fog or cloud [9].

Thirdly, IoT systems are expected to continuously evolve in order to handle new services, features, and devices that had not been anticipated when the system was first designed. A *monolithic* service-oriented architecture (SOA), where all software is within a single application, would limit an IoT system's ability to scale and evolve as changes would require a complete restart of the system [10]. However, a *microservices* architecture breaks down the monolithic structure into small applications with individual responsibilities that can be deployed, scaled, and tested independently [11]. They act as stand-alone subunits that interconnect through message-passing, making them lightweight and easy to update in scenarios where one can not fully anticipate functionalities in advance [12].

As IoT systems are heavily dependent upon their location and context, FT support should *react* to faults that occur in real time to isolate faults as quickly as possible. With sufficient analysis of the system's state, data, and historical faults over time, there is the potential to *predict* faults that often occur under specific patterns of system usage.

A. Contribution

We propose an FT framework based on a microservices architecture to provide a scalable means of applying real-time and predictive FT support to IoT systems. Our framework is

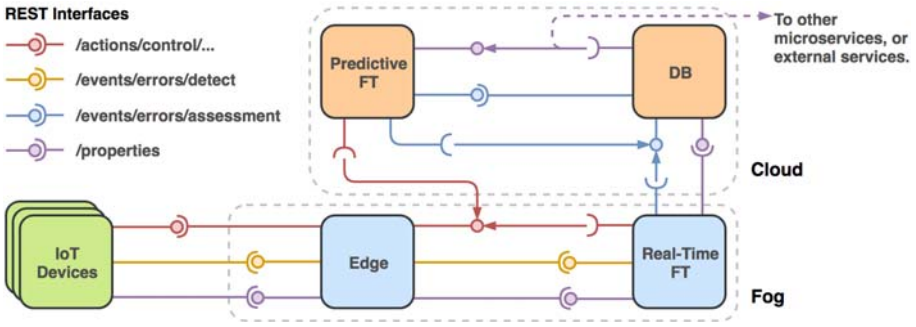

Fig. 1. The interfaces between IoT devices and the four microservices defined in Section III-A.

designed to provide a live and continuous 'health check' [13], whereby our FT microservices constantly assess system state, data, and errors to detect and mitigate faults.

The data extracted from system monitoring will help to provide *fault patterning*, whereby faults are assessed w.r.t. the system context so that the system can learn to identify when fault activations are likely to occur due to similar prior experiences. By identifying correlations between faults, the system can proactively handle them before they are activated.

Concisely, our contributions are as follows:

1) To proprose a microservices architecture where FT is plugged in as a service, exploring its design, interfaces, and potential to scale.
2) To consider a fault scenario that might occur and how reactive and proactive FT can handle it.

The rest of the paper is organized as follows. Section II discusses previous work about FT and microservices in IoT. Section III describes our proposed framework and its architecture. Section IV demonstrates our framework with a real-world scenario. Section V summarizes our work.

II. Related Work

A. Fault Tolerance in IoT

Choubey et al. [14] present a smart home architecture where sensors are first analyzed for correlations so that, if some sensor data can be predicted by others, a neural network can be trained to predict the data. This provides redundancy for failed devices as data can be predicted on lost data until repairs are made. Tests show predicted temperatures deviate from actual values by ± 2°C. Zhou et al. [15] consider using sensors of different *modalities* (i.e. that are not directly compatible) to provide hardware redundancy for failed sensors. They identify compatibility between sensors via regression analysis and combine the data. Their experiment shows that light sensors distant from a failed sensor perform badly by themselves but achieve great results when combined. Karthikeya et al. [16] propose the NewIoTGateway-Select algorithm for smart cities to determine the minimum number of necessary gateways, to reduce deployment costs and provide redundancy. The algorithm considers gateway and link failures by ensuring at least k routes exist between them. Simulations show that the total number of gateways is much lower than what would be used without the algorithm. Su et al. [17] prefer a decentralized solution where devices are in a ring topology and services are delegated to devices, where the failure of a device shifts responsibility to a redundant device to provide the lost service.

B. Microservices in IoT

Sun et al. [18] propose an IoT framework based on microservices that decomposes the system into nine units that communicate over a REST interface. A *core* microservice coordinates the eight others which provide security, storage, big data analytics, etc. They use microservices as it allows the framework *to easily extend, evolve, and integrate third-party applications to support interoperability and scalability*, with a greater ability to deploy FT at scale. Celesti et al. [19] consider a watchdog service for containerized microservices deployed as middleware on IoT devices. Microservice failure prompts a repair or replacement with a replica. Their solution shows acceptable overhead during recovery. Krylovskiy et al. [20] present the DIMMER platform for smart cities to enable stakeholders to increase the energy efficienty of a city at the district level. They use microservices for its decentralized governance, meaning microservices can use their own technologies, free of standards and platform homogeneity.

III. Framework Architecture

A. System Design

Our proposed framework is the integration of four microservices, where two provide FT support in complementary ways: the first provides real-time data stream analysis using *complex event processing* (CEP) for reactive FT, and the second uses *machine learning* (ML) to provide proactive FT.

Reactive FT is where the system initiates an error recovery strategy *after* an error has been detected. This requires fast detection and decision making with a low-latency connection to the hardware/software at fault. The fog can provide cloud-like services to the network edge for low-latency data analytics, making it an ideal candidate for analyzing stream data [21].

Proactive FT is where the system initiates an error recovery strategy *before* an error has been detected. The concept is designed to prevent failures from impacting an application by preemptively migrating parts of a system away from any soon-to-fail hardware/software [22].

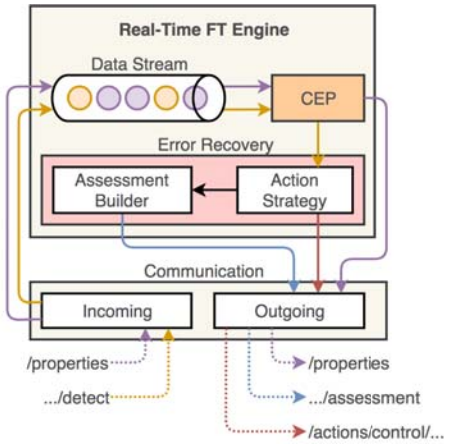

Fig. 2. The architecture of the Real-Time FT microservice. Purple arrows indicate the flow of properties data, yellow for detected errors, blue for error assessments, red for error recovery actions, and black for internal data.

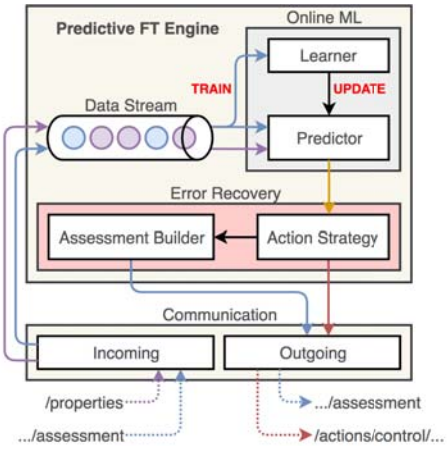

Fig. 3. The architecture of the Predictive FT microservice. Arrow colors are the same as in Figure 2.

As with many microservice architectures, communication between our services is conducted via a RESTful architecture style where data is exchanged using the JSON format. Other protocols suitable for IoT include CoAP, MQTT, and XMPP, however the advantage of REST is that almost all cloud platforms support it [23], making it the ideal choice for encouraging interoperability across IoT systems.

The *Web Thing API* [24] provides a standard approach for describing physical devices and is designed to allow access to device properties, request the execution of actions and subscribe to events that occur within the device. As shown in Figure 1, we base the interface on three overarching categories defined in the API, as follows:

- **/properties**: describes attributes of an IoT 'thing' (e.g. sensory information, such as temperature) as well as internal information about system devices and microservices.
- **/events**: describes events that occur within the system and on IoT devices. The **/events/errors/detect** interface is where devices can POST errors that they detect or receive. The **/events/errors/assessment** interface enables microservices to annunciate that they have provided some form of recovery to handle an error. Errors are further discussed in Section III-B.
- **/actions**: describes system functionality to be performed by a microservice. The **/actions/control/...** interface enables a microservice to control the functionality of another. Actions are further discussed in Section III-C.

When a microservice registers itself with the service broker, it defines which categories (i.e. properties, events, actions) it wishes to subscribe to. Other microservices, after registration, emit data to the subscribers. Our proposed architecture centers around four microservices, discussed next.

1) Edge: This microservice provides an entry point for IoT devices to pass its properties (Table I) to the wider system and for microservices to interact with devices. It provides REST interfaces for devices to submit their properties and errors and to receive commands for controlling IoT device actuators. Edge relays all of its received data to other microservices that are subscribed to it, and any actions that are sent to Edge are either executed internally or relayed to an IoT device.

2) Real-Time FT: As shown in Figure 1, this microservice is situated between Edge and DB (Section III-A3). It acts as a 'firewall' that only permits *reasonable* properties from reaching DB once it has been passed through the Real-Time FT Engine (Figure 2), whereby the stream of properties and detected errors are fed into a CEP system. This analyzes streams of 'primitive events' (i.e. properties, errors) then combines them to define and detect a number high-level, complex situations (i.e. new errors) [25].

CEP provides an intelligent way to handle errors because it can enable one to define recovery strategies based on many errors rather than just one. For example, if five IoT devices fail within three seconds, the CEP system might consider that the gateway to the devices has failed, rather than the devices themselves. Errors can also be produced based upon properties alone. For example, if a property's value spikes and deviates from the average by some margin, the system can tell the Edge to isolate the device producing the property. Additionally, errors can be produced by combining properties and errors for more intelligent error analysis and recovery.

The benefit of microservices is that, if one crashes, it does not bring down the entire application. However, other failure types (Section I) have the potential to cascade into other areas of the system because of data (or a lack of it). Error detection relies upon constant system checks that can be placed into the following classifications [26]: replication, timing, reversal, coding, reasonableness, structural, and diagnostic checks.

Checking data *reasonableness* is a challenge in IoT because what constitutes 'reasonable' data is highly context dependent. For example, high temperatures in an office might be reasonable at 1pm, but not at 1am. An example of our framework approaching data reasonableness is provided in Section IV.

3) DB: This microservice is a back-end database service that receives data to store for (authorized) services to subscribe

TABLE I
THE DATA TRANSMITTED WHEN SENDING A PROPERTY USING THE /PROPERTIES INTERFACE.

Field	Description	Example
sourceAddr	The IP address of the property source.	192.168.1.2
entryAddr	The IP address of the microservice that first received the property.	192.168.1.3
name	The name of the property.	Temperature
type	The type of the property.	Number
unit	The unit for the type.	Celsius
value	The value of the property.	20.5
timestamp	When the property was created, in epoch milliseconds.	1520168977817

to. Predictive FT (Section III-A4) subscribes to DB to consume its properties and error assessments.

4) Predictive FT: In IoT, data is constantly flowing from source(s) to sink(s), capturing the latest state of the system and its physical environment(s). To pre-empt faults, predictions must be made using this live continuous data. For this, we consider an *online learning* (OL) approach. In OL, a sequence of hypotheses $f = (f_1, ..., f_{m+1})$ are produced over time, where f_1 is an arbitrary initial hypothesis and f_i for $i > 1$ is the hypothesis of the $(i-1)$th example [27].

Unlike with *batch learning* (BL), where a single predictor is generated based upon an entire dataset, OL is trained incrementally with data that arrives in a continuous stream and the algorithm updates and adapts on the fly [28], making it ideal for IoT systems. BL systems can adapt to change if the training and launching of each new algorithm is automated. However, continually training a BL algorithm from scratch on *all* current and prior data is a computationally expensive process that requires far more storage space for this ever-expanding dataset; OL can discard data once it has used it.

Current OL techniques exist as extensions of established algorithms (e.g. Support Vector Machines, Bayes), ensemble learning variants (e.g. Online Random Forests), and algorithms that are online by design (e.g. K-Nearest Neighbor) [29], [30].

Predictive FT receives error assessments and properties (Figure 3) which are then fed into Learner to train the algorithm to: (1) *identify* errors and the system state(s) that led to them; (2) *learn* how the system attempts to recover from errors; and (3) *evaluate* the effectiveness of the recovery strategies, so that only effective strategies are learned from. Using this knowledge, *fault patterns* can be generated and, using subsequent system data, help to probabilistically infer whether errors are likely to happen in the future.

B. Errors

1) Error Detection: The *Eight-Ingredient* (8I) framework was developed as a systematic way of conducting vulnerability analysis on both internal and external aspects of a system. It identifies reliability and security as being vital for continuous system operation and the key infrastructure upon which all other critical infrastructures depend [31].

It defines eight *ingredients* that identify different vulnerability types that can manifest, namely: (1) *human*, (un)intentional behaviors, physical limitations, etc; (2) *policy*, agreements, standards, policies, etc; (3) *hardware*, electronic and physical components; (4) *software*, creating, maintaining, and protecting code; (5) *networks*, node configuration, synchronization, redundancy; (6) *payload*, information transported across the infrastructure; (7) *environment*, harsh conditions where hardware is exposed to weather conditions, etc; and (8) *power*, internal power infrastructure, batteries, cabling, etc.

TABLE II
THE DATA TRANSMITTED WHEN SENDING AN ERROR DETECTION EVENT USING THE /EVENTS/ERRORS/DETECT INTERFACE.

Field	Description	Example
sourceAddr	The IP address of the error source.	192.168.1.2
scope	Whether the error occurred due to internal or external factors.	Internal
ingredient	The ingredient that applies to the error (Section III-B1).	Hardware
category	The error category w.r.t. the ingredient.	FRU
scenario	The type of failure w.r.t. the category.	Sensor Failure
fault	The (believed) cause of the error.	PIR Sensor
persistence	Whether it is a transient, intermittent, or permanent fault.	Permanent
description	A human-readable description of the error.	"Cannot activate PIR sensor."
timestamp	When the error was detected, in epoch milliseconds.	1520168977817

TABLE III
THE DATA TRANSMITTED WHEN SENDING AN ERROR ASSESSMENT EVENT USING THE /EVENTS/ERRORS/ASSESSMENT INTERFACE.

Field	Description	Example
pattern	The properties and errors that caused the detected error.	Table I & II
error	The detected error.	Table II
actions	The actions taken to recover from the detected error.	Section III-C
approach	Whether a reactive or proactive recovery approach was used.	Reactive
timestamp	When the assessment was created, in epoch milliseconds.	1520168977817

As shown in Table II, we begin building error detection events with the error source, followed by many categories that make a top-down analysis of the error, down to affected hardware/software that triggered it (i.e. the *fault*). The format is based upon the approach by Bauer [32], where ingredients are mapped to error categories and scenarios to achieve a systematic approach of defining test cases. Bauer [32] also provides error categories, namely: field-replaceable units (FRUs), programming, data inconsistency, redundancy, system power, network, application protocol, and procedural errors.

2) Error Assessment: The error assessment event is the product of the Real-Time FT microservice (Section III-A2) handling error events and inferring errors based upon prior data. Shown in Table III, it comprises a list of errors and properties that are used to detect a new error, as well as actions taken to try to recover from the new error.

The chosen actions are based upon the assumed fault that potentially caused the error. It is *assumed* because FT support

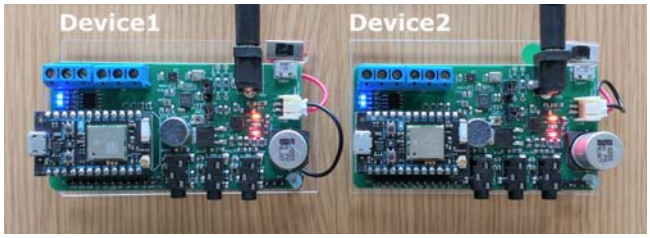

Fig. 4. Device1 and Device2 from our early evaluation (Section IV).

can only infer the root cause of errors through probabilistic methods, which CEP and ML aim to do. The fault classifications can be those introduced in Section I, namely: crash, omission, timing, response, and arbitrary [8].

C. Actions

When recovering from faults, systems can employ *backward* and *forward* error recovery mechanisms, where the former tries to restore a previous system state, and the latter tries to move into a new, error-free state [33]. In IoT, data, and the services that rely upon it, help to create virtual entities that resemble physical entities in the real world by monitoring their states with sensors and actuators [34]. Therefore, forward error recovery is the ideal option to keep the system focused upon the latest states and data.

Erroneous data from low-level IoT devices can hinder the performance of our framework. If DB stores bad data, it can harm other services that rely upon its data. If Predictive FT consumes bad data, it could suffer from a *concept drift*, where the input distribution with which the OL algorithm is trained changes and the algorithm's accuracy lowers over time [29].

To combat this, the **/actions/control/block** interface on Edge is called by Real-Time FT to block data from, and interactions with, IoT devices. If the CEP system (Figure 2) flags a property as erroneous, then its *sourceAddr* and *name* (Table I) are sent to the Edge at *entryAddr* to block it and prevent further bad data from propagating through the system.

D. Scalability

Butzin et al. [13] identify the fog as an enabling technology for containerization in IoT, which is a key tool for deploying microservices. In our architecture, microservices are distributed across the fog and cloud (Figure 1). The fog is important because it provides a network with a gateway to (a subset of) services without long-range connections to the cloud, enabling low latency and rapid response times for Real-Time FT. As Predictive FT is in the cloud, it can be a shared service for all system clients, where error assessments are 'crowdsourced' to improve the ML algorithm's predictions.

IV. Early Evaluation

To demonstrate how Real-Time FT and Predictive FT work together, we generated live data using two multi-sensor boards, *Device1* and *Device2* (Figure 4), that have infrared, ultraviolet (UV), and visible light sensors, and a microphone for sound detection. We performed two experiments where we left the devices running for two weeks in both experiments. They were given constant power so that we can observe errors and faults that were not power related.

In our first experiment (Figure 5), visible light dropped to 0 between 21:43 and 21:44 on Device1 before returning to normal. Then, a stuck-at fault occurs on all light sensors, where their values exceed 100 and remain constant until the end of the experiment. This pattern occurs *in both experiments on Device1*, where a drop in visible light occurs minutes before a major stuck-at fault in all light sensors.

In this scenario, our framework can perform the following:

1) The CEP system (Figure 2) identifies Device1's visible light drop to 0 and flags it as erroneous data, because the value deviates from the last ten seconds of data by a significant margin. An error assessment targets Device1's visible light sensor as the cause and considers it a transient fault. The action taken is to drop the data, as the values return to normal immediately afterwards.
2) When the major stuck-at fault occurs, the assessment identifies the three light sensors as the cause. Real-Time FT contacts Edge via */actions/control/block* to block the three light properties from Device1. The sound property is still accepted as it is not producing erroneous data.
3) Each time the last two steps occur, the two error assessments are produced. Predictive FT receives these assessments each time and identifies the fault pattern that the first error is often followed minutes later by a stuck-at fault on Device1.
4) When the drop to 0 occurs, Predictive FT identifies a high probability of the stuck-at fault occurring and notifies Edge to activate a redundant replica to produce these properties so that, if/when Device1 has the predicted stuck-at fault, there is another sensor to take over. Otherwise, Edge can just block the properties.

V. Conclusion

Providing FT support to IoT systems is an open challenge, with many implementations providing static, tightly coupled FT support that does not adapt and evolve as IoT systems do. We have proposed a framework based on a microservices architecture that provides reactive and proactive FT support with two microservices: Real-Time FT, that uses complex event processing to analyze stream data for rapid error recovery; and Predictive FT, that uses machine learning to learn fault patterns and mitigate future faults before they occur. We have presented the necessary interfaces and evaluated the framework against a real-world scenario.

References

[1] F. Mattern and C. Floerkemeier, *From the Internet of Computers to the Internet of Things*. Springer Berlin Heidelberg, 2010, pp. 242–259.
[2] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (iot): A vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645–1660, 2013.
[3] M. A. Razzaque, M. Milojevic-Jevric, A. Palade, and S. Clarke, "Middleware for internet of things: a survey," *IEEE Internet of Things Journal*, vol. 3, no. 1, pp. 70–95, 2016.

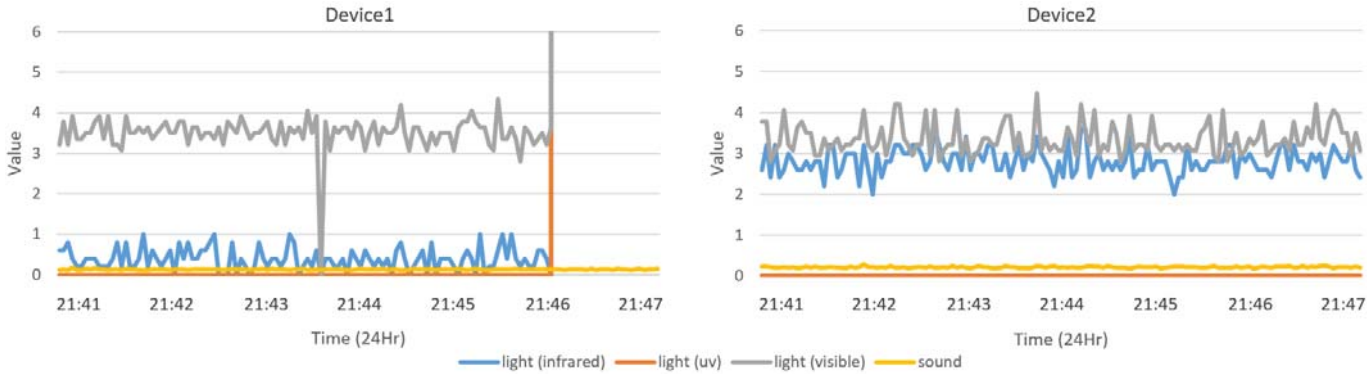

Fig. 5. The data from our first experiment, showing four properties over a six minute period on Device1 (left) and Device2 (right). Device1 experiences a stuck-at fault from 21:46 onwards, whereas Device2 is running as normal.

[4] G. Choudhary and A. Jain, "Internet of things: A survey on architecture, technologies, protocols and challenges," in *2016 International Conference on Recent Advances and Innovations in Engineering (ICRAIE)*. IEEE, 2016, pp. 1–8.

[5] G. White, V. Nallur, and S. Clarke, "Quality of service approaches in iot: A systematic mapping," *Journal of Systems and Software*, vol. 132, pp. 186–203, 2017.

[6] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, 2004.

[7] I. Sommerville, *Software Engineering, Global Edition*, 10th ed. Pearson Education Limited, 2016.

[8] F. Cristian, "Understanding fault-tolerant distributed systems," *Commun. ACM*, vol. 34, no. 2, pp. 56–78, Feb 1991.

[9] A. Botta, W. De Donato, V. Persico, and A. Pescapé, "Integration of cloud computing and internet of things: a survey," *Future Generation Computer Systems*, vol. 56, pp. 684–700, 2016.

[10] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, *Microservices: Yesterday, Today, and Tomorrow*. Cham: Springer International Publishing, 2017, pp. 195–216.

[11] J. Thönes, "Microservices," *IEEE Software*, vol. 32, no. 1, pp. 116–116, 2015.

[12] M. Fazio, A. Celesti, R. Ranjan, C. Liu, L. Chen, and M. Villari, "Open issues in scheduling microservices in the cloud," *IEEE Cloud Computing*, vol. 3, no. 5, pp. 81–88, 2016.

[13] B. Butzin, F. Golatowski, and D. Timmermann, "Microservices approach for the internet of things," in *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, Sept 2016, pp. 1–6.

[14] P. K. Choubey, S. Pateria, A. Saxena, V. P. C. SB, K. K. Jha, and S. B. PM, "Power efficient, bandwidth optimized and fault tolerant sensor management for iot in smart home," in *2015 IEEE International Advance Computing Conference (IACC)*. Bangalore: IEEE, 2015, pp. 366–370.

[15] S. Zhou, K.-J. Lin, J. Na, C.-C. Chuang, and C.-S. Shih, "Supporting service adaptation in fault tolerant internet of things," in *2015 IEEE 8th International Conference on Service-Oriented Computing and Applications (SOCA)*. IEEE, 2015, pp. 65–72.

[16] S. A. Karthikeya, J. K. Vijeth, and C. S. R. Murthy, "Leveraging solution-specific gateways for cost-effective and fault-tolerant iot networking," in *2016 IEEE Wireless Communications and Networking Conference (WCNC)*. Doha: IEEE, 2016.

[17] P. H. Su, C.-S. Shih, J. Y.-J. Hsu, K.-J. Lin, and Y.-C. Wang, "Decentralized fault tolerance mechanism for intelligent iot/m2m middleware," in *2014 IEEE World Forum on Internet of Things (WF-IoT)*. Seoul: IEEE, 2014, pp. 45–50.

[18] L. Sun, Y. Li, and R. A. Memon, "An open iot framework based on microservices architecture," *China Communications*, vol. 14, no. 2, pp. 154–162, 2017.

[19] A. Celesti, L. Carnevale, A. Galletta, M. Fazio, and M. Villari, "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)*. IEEE, 2017, pp. 372–378.

[20] A. Krylovskiy, M. Jahn, and E. Patti, "Designing a smart city internet of things platform with microservice architecture," in *Future Internet of Things and Cloud (FiCloud), 2015 3rd International Conference on*. IEEE, 2015, pp. 25–30.

[21] A. V. Dastjerdi and R. Buyya, "Fog computing: Helping the internet of things realize its potential," *Computer*, vol. 49, no. 8, pp. 112–116, 2016.

[22] C. Engelmann, G. R. Vallee, T. Naughton, and S. L. Scott, "Proactive fault tolerance using preemptive migration," in *2009 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing*. IEEE, 2009, pp. 252–257.

[23] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of things: A survey on enabling technologies, protocols, and applications," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.

[24] Mozilla, "Web thing api," 2018, accessed: 2018-03-01. [Online]. Available: https://iot.mozilla.org/wot/

[25] G. Cugola and A. Margara, *The Complex Event Processing Paradigm*. Cham: Springer International Publishing, 2015, pp. 113–133.

[26] P. A. Lee and T. Anderson, *Fault Tolerance: Principles and Practice*, ser. Dependable Computing and Fault-Tolerant Systems. Springer Vienna, 2012.

[27] J. Kivinen, A. J. Smola, and R. C. Williamson, "Online learning with kernels," *IEEE Transactions on Signal Processing*, vol. 52, no. 8, pp. 2165–2176, 2004.

[28] Ó. Fontenla-Romero, B. Guijarro-Berdiñas, D. Martinez-Rego, B. Pérez-Sánchez, and D. Peteiro-Barral, "Online machine learning," *Efficiency and Scalability Methods for Computational Intellect*, vol. 27, 2013.

[29] A. Gepperth and B. Hammer, "Incremental learning algorithms and applications," in *European Symposium on Artificial Neural Networks (ESANN)*, Bruges, Belgium, 2016.

[30] B. Krawczyk, L. L. Minku, J. Gama, J. Stefanowski, and M. Woźniak, "Ensemble learning for data stream analysis: A survey," *Information Fusion*, vol. 37, pp. 132–156, 2017.

[31] K. F. Rauscher, R. E. Krock, and J. P. Runyon, "Eight ingredients of communications infrastructure: A systematic and comprehensive framework for enhancing network reliability and security," *Bell Labs Technical Journal*, vol. 11, no. 3, pp. 73–81, 2006.

[32] E. Bauer, *Design for Reliability: Information and Computer-Based Systems*. Wiley, 2011.

[33] L. L. Pullum, *Software Fault Tolerance Techniques and Implementation*, ser. Artech House computing library. Artech House, 2001.

[34] A. Bassi, M. Bauer, M. Fiedler, T. Kramp, R. v. Kranenburg, S. Lange, and S. Meissner, Eds., *Enabling Things to Talk*. Springer Berlin Heidelberg, 2013.