An MQTT-based Resource Management Framework for Edge Computing Systems

Conference Paper · August 2020

DOI: 10.1109/INISTA49547.2020.9194690

3 authors:

Sasa Pesic
Arizona State University
20 PUBLICATIONS **151** CITATIONS
SEE PROFILE

Milos Radovanovic
University of Novi Sad
97 PUBLICATIONS **1,840** CITATIONS
SEE PROFILE

Mirjana Ivanovic
University of Novi Sad
450 PUBLICATIONS **4,661** CITATIONS
SEE PROFILE

An MQTT-based Resource Management Framework for Edge Computing Systems

Saša Pešić*, Miloš Radovanović*, Mirjana Ivanović,*
*University of Novi Sad, Faculty of Sciences. *sasa.pesic,radacha,mira@dmi.uns.ac.rs*

***Abstract*—The complexity of IoT systems and tasks that are put before them require shifts in the way resources and service provisioning are managed. The concept of edge computing is introduced to enhance IoT systems' scalability, reactivity, efficiency, and privacy. In this paper, we present an edge computing solution for resource management of context-aware decision-making processes distributed between IoT gateways. The solution performs decision-making process management for smart actuation, based on analysis of sensory data streams, and context-informed edge computing resource and service provisioning management based on topology and operational changes. Our architectural solution showcases the first version of a Resource Management Framework – a generic framework for software resource orchestration best-suited to IoT platforms with event-driven, publish-subscribe communication mechanisms. Proof of concept experiments that are executed in a simulated edge computing testbed validate our solution's performance in improving the resilience and responsiveness of the edge computing system when there are operational and topology changes. Furthermore, the framework addresses the recovery of failed decision-making processes, impacting the overall health of the underlying IoT system.**

***Index Terms*—edge computing, IoT, resource management, MQTT**

I. Introduction

The term IoT refers to all devices of everyday life that are connected to the Internet and that have some kind of intelligence. It is applicable in domestics, all kinds of appliances that communicate with the user, biomedical control systems, trends in consumer use, automobile industry, etc. Smart cities are great incubators for IoT systems [1] that make urban living more enticing, such as agile and convenient transportation systems, eco-friendly heat, and electricity management [2] and energy-efficient buildings [3]. IoT and advances in AI are making it possible for devices, sensors, and actuators to operate autonomously and communicate directly without human intervention.

Much of the data in the IoT is processed using cloud computing resources. However, data provides the most value when it is interacted with in real-time. Cloud computing is good for offline analysis of large data sets which provides the basis for proactive management of processes and resources. However, apart from this strategic nature of data analysis, most systems and processes in the IoT domain require real-time actions and cannot risk delays introduced by transferring all the data to a cloud platform for analysis. That is why data processing functionalities are being moved to the edge, which represents a computing paradigm better known as edge computing (ECP). This leads to providing real-time actionable insights that transform into major business benefits.

978-1-7281-6799-2/20/$31.00 ©2020 IEEE

The complexity of IoT systems and tasks that are put before them require shifts in the way resources and service provisioning are managed. Network management approaches like software-defined networking and advanced routing protocols provide a certain level of robustness. If the network of IoT gateways (GWs) performs only data formatting and bridging to cloud resources, then these network management solutions are very effective in overcoming topology disturbances. However, when IoT GWs are responsible for reactive decision-making (DM) directly influencing the operational environment of the system these network management solutions are ineffective in keeping the system functionalities *alive* (i.e. address failed DM/actuation processes). One solution to this problem would be to provide a fail-over mechanism towards cloud resources. However, that is not always a possibility nor is desirable for ECP systems since they rely on data privacy and low latency. Therefore, ECP systems need a solution that allows participating nodes to reinstate not only failed/missing data streams from their peers but also data-analytics (DA) and DM processes. In this paper, we are proposing a Resource Management Framework (RMF) for ECP systems that provides a decentralized solution to this challenge. The RMF is tested in a simulated environment.

The rest of the paper is structured as follows: Section II presents the state-of-the-art approaches and identifies existing research gaps; Section III provides a description and high-level functional architecture of the RMF; Section IV presents groups of operational workflows inside RMF; Section V describes fail-recover scenarios resolved by the RMF; Section VI presents the simulation experiments and results, and offers a discussion of advantages and problems with the current implementation of the RMF; Finally, Section VII provides conclusive remarks and future work directions.

II. State-of-the-art and Research Gaps

As state-of-the-art goes, research papers addressing resource management can be divided into two groups: (1) addressing a lower-level (specific) solution, and (2) addressing a high-level architectural approach. Papers from group (1) can be further categorized into resource management techniques focusing on (a) storage, (b) network, (c) computational resources, and (d) software components. Our paper describes an RMF from a higher-level standpoint (2), focusing on a re-distribution

approach for software components of the underlying IoT/ECP architecture (d), while approaches we use for (a), (b) and (c) are not discussed in this paper.

Papers addressing solutions from group (2) include container-based allocation of resources [4], infrastructure health audit [5], etc. Solutions for resource management (1) in IoT/ECP, in categories (a), (b), and (c) are not lacking. Kim et al. describe an efficient XML-based classification scheme for data storing/processing of heterogeneous data for IoT [6]. Sehgal et al. proposed a set of IP-based network management protocols implemented on resource-constrained devices [7]. Many research papers are addressing computational resource allocation and offloading management for IoT/ECP systems to improve their performance [8]–[10]. Hong et al. presented a thorough survey of such approaches [11].

On the other hand, solutions managing failing software components and logical components of ECP architectures are under-researched. In their study, Alreshidi et al. underlined that future research on architecting logical components (software) of IoT platforms will be focused on architectural patterns that support automation and reusability to dynamically adapt IoT software [12]. As a movement towards solving a part of that challenge, Leiba et al. use blockchain, smart contracts, and Zero-knowledge proof to build a decentralized network for the propagation of IoT software update [13]. However, secure software updates are a small part of the bigger problem that is fail-rescue strategies of processes inside an IoT platform.

Taking into account the above-mentioned research, in our paper, we focus on architecting a high-level resource management framework for re-distribution of running DA and DM processes throughout an event-driven, publish-subscribe communication mechanism-based IoT platform. To the best of our knowledge, this is the first solution focusing exclusively on the Message Queuing Telemetry Transport (MQTT) protocol. Thus, our main contributions with addressing such a framework rest in: (1) providing a higher-level RMF for the IoT based on a lightweight communication protocol (MQTT), as contrary to the above-mentioned papers that are focused on a specific resource management task; (2) providing step-by-step detailed description for resolving sophisticated failure scenarios; (3) discussing recover scenarios not only for failure but also in workload re-distribution to achieve optimal operational capacity.

III. RMF Functional Architecture and Workflows

The high-level architecture of the proposed resource management framework for recovering ECP systems is displayed in Figure 1.

The actors of the RMF can be categorized into physical and software components. Among the physical components, there are typical IoT platform-connected hardware: sensors, actuators, and devices. Sensors and Actuators can be connected to multiple Devices by wire and/or wirelessly – e.g. Actuators 1 and 2 are connected wirelessly to Device 1 and connected by a wire to Device 2. A Device has hardware and a software stack – hardware stack describes the set of wireless technologies the Device is capable of communicating on and the software stack describes the set of software components (IoT platform and RMF related) deployed on the Device.

The software stack for each Device consists of two groups of software modules: (1) providing IoT platform functionalities (multiple soft sensors, the MQTT broker), and (2) providing RMF functionalities (Controller and the Soft Sensors Controller Module (SSCM)).

In the observed IoT platform, the soft sensors approach is used to enable scalable and hierarchically distributed DA and DM processes across the platform, but the RMF as well [14], [15]. Soft sensors represent self-contained software modules performing simple analytical processes and exposing services and/or data via MQTT. They are hierarchically deployed which allows the creation of analytical chains where each step in the DA chain provides deeper insight into the context of the managed IoT system. 1st layer soft sensors are deployed right on top of physical sensors and actuators, 2nd layer soft sensors read, preprocess, and selectively forward data to interested parties, 3rd layer soft sensors perform aggregation, and finally 4th layer soft sensors perform DM.

Publish/subscribe mechanisms are a suitable choice for machine-to-machine correspondence due to loose coupling and simple communication architecture [16]. One of the widely accepted standards is the MQTT protocol which passes messages between multiple clients through a central broker. It has a strong foothold in IoT and fog computing systems [17], [18]. In the observed IoT platform and RMF, all communication is MQTT-based (soft sensors-soft sensor, soft sensor-device, device-device). In the RMF there are 5 critical topics:

- **Active soft sensors** – used to propagate information about soft sensors failure/recovery to the SSCM
- **Commands** – used to propagate commands to and from RMF devices
- **$SYS** – used to propagate information about device and communications operation (device CPU, RAM, storage, MQTT broker status, etc.)
- **MAR i.e. Manual Action Required** – used to propagate information to the platform administrator that the RMF did not resolve a failure issue itself and manual action is required
- **SSCM** – used to distribute the SCT from RMF leader to other devices

A Controller module is a critical functioning part of the RMF. The soft sensors approach in deploying DA and DM mechanisms ensures high independence of the ECP system. To separate the RMF management functionalities from the underlining IoT platform, the Controller module handles all RMF-related communication: failed processes notification, command propagation, RMF leader voting, etc.

The SSCM is in charge of three operational tasks: (1) Building the system's control table, (2) Running continuous health checks, and (3) handle processes failure, redistribution, and ensuring fail-safe system operation. Throughout the operation, if the IoT platform, the Control Table Builder (1) maintains the

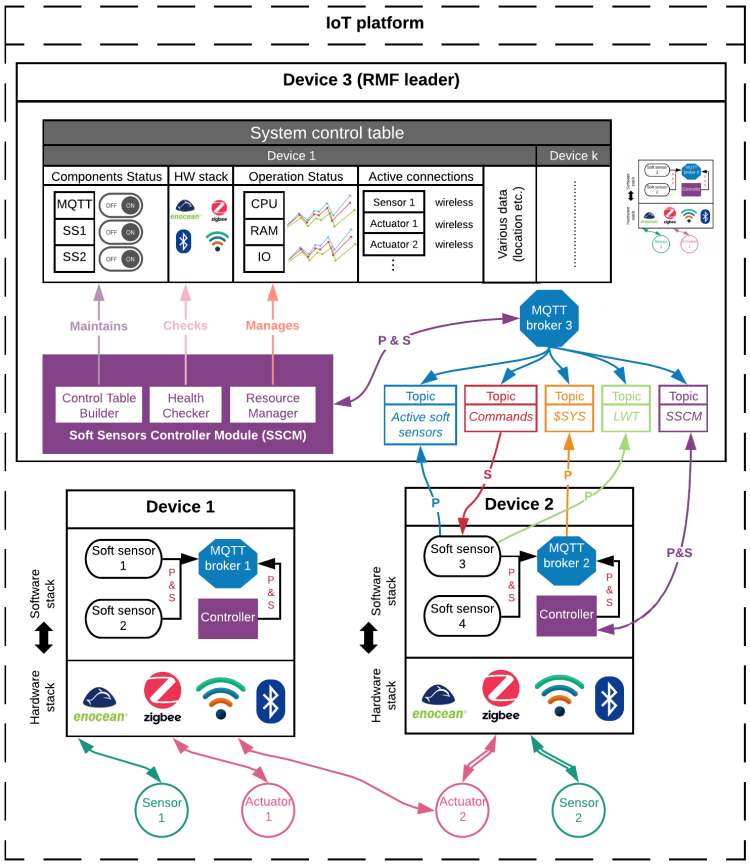

Fig. 1. Resource Management Framework high-level workflow.

system's control table – persisting and continuously updating information about the status of the entire IoT platform per device: running physical components and soft sensors, MQTT broker status, etc. Health Checker (2) updates the same control table with resource consumption information on the one hand, and on the other runs regular workload balance checks and high-risk analysis of potentially failing devices. The Resource Manager (3) does not perform DA but serves as a proxy to issue two types of commands: (a) actions based on the results of the analysis from (2) Health Checker and (b) reactions to failed soft sensors information that reaches the SSCM from the IoT platform belonging devices.

IV. RMF: Standard Operational Workflows

RMF operational workflows are categorized into 5 groups, depending on their role in the RMF. These groups of workflows are: (1) SCT update and propagation, (2) RMF Leader voting and role switch, (3) System health status data collection, (4) System health and workload distribution analysis and (5) Fail-recover scenarios handling. These groups and belonging processes are displayed in Figure 2. It is important to highlight that every process inside RMF runs as a soft sensor. Thus, RMF processes that fail are also recovered with other RMF-established workflows. In the rest of this Section, we will elaborate on the RMF-specific workflows.

1. SCT update and propagation – Processes from this group are in charge of manipulating the SCT, which includes freezing the table (table does not take in updates), updating the table's contents, publishing SCT updates via MQTT to IoT Controllers and a process to gather initial SCT information once the RMF is setup. SCT is distributed to all devices every k seconds;

2. RMF Leader role switch – RMF leadership role is awarded to devices of the RMF in a round-robin fashion. The Controller script of the RMF leader issues a command for a role switch every k seconds. The success of that command is evaluated through analyzing SCT updates (whether RMF role status was updated).

3. System health status data collection – A set of processes enabling context-building inside the RMF through collecting operational status information about running communications (MQTT brokers and bridges), device operation status (CPU/RAM/Storage usage), physical component status (active sensor connections, communication channels, etc.)

4. System health and workload distribution analysis – Processes in this group are in charge of ensuring the health and optimal operation of the underlying edge computing platform. This is achieved through constant cross-referencing of current health parameters with defined optimal ones (by platform administrator) and reacting when a deviation is detected. WBA,

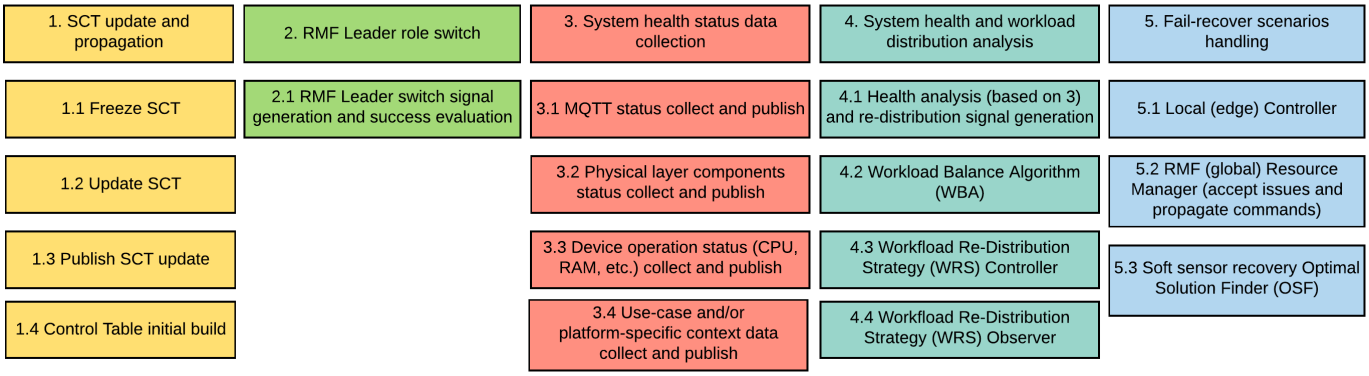

Fig. 2. RMF standard operational workflow grouping.

WRS Controller, and WRS Observer handle activities related to executing and evaluating the success of re-distribution strategies. Health Checker performs regular health checks of all system components every k seconds;

5. Fail-recover scenarios handling – A set of processes covering activities for recovering failed components of the underlying edge computing platform. For enabling recovery, all Devices host all soft sensors that they are capable of running (technology and access compatibility can be a problem for 1st layer soft sensors). All soft sensors send keep-alive pings to the Controller every k seconds. Controller (local) and Resource Manager (global) immediately react to all Publish/Subscribe actions triggering other RMF functionalities.

The RMF operates as an integral part of an IoT platform. However, on top of RMF a Platform administrator management portal is envisioned to enable IoT platform administrators to issue manual commands through the RMF when needed. Also, there is a time limit for all fail-rescue scenarios predefined by platform administrators. If these limits are exceeded the Manual Action Requirement event is triggered and the platform administrator is notified.

V. RMF: Fail-recover Sample Scenarios Description and Handling

To fully understand the RMF and its inner workings this section will present a detailed workflow from the standpoint of how RMF handles recovery of failed processes and devices and workload re-distribution. Our proposed RMF addresses 9 fail-recover scenarios: T1. RMF leader switch; T2. Soft sensor failure; T3. MQTT broker failure; T4/T5/T6. CPU/RAM/Storage peak; T7. RMF leader failure; T8. Controller failure; T9. Device failure. To visualize how RMF handles these scenarios, three scenarios will be explained and visualized. Steps taken from the moment the scenarios are triggered up to when they are resolved are displayed in Figure 3. Software specification for all other triggers/scenarios (T1–T9) is available in the RMF as well, but will not be further discussed in this paper.

Scenario 1 covers the failure of a soft sensor of any hierarchical layer (T2. Soft sensor failure). As soon as the Controller stops receiving keep-alive pings from a failed soft sensor (step 2) a *FAIL* message is posted to the *Active soft sensors* topic on a local MQTT broker (step 3). This message is immediately transferred to all remote MQTT brokers bridged to this MQTT broker. The message is received by the Control Table Builder (CTB) which updates the SCT. The Resource Manager of the Device that is currently the RMF leader will receive this message as well (step 4) and will search for the optimal solution for recovering the failed soft sensor (step 5). The optimal solution is found in 2 steps: (a) find all devices capable of running failed soft sensor; (b) launch the failed soft sensors on the device with lowest average workload (CPU, storage, and RAM utilization). Once the solution has been obtained, the SCT is frozen and no other topology updates are received at this point (step 6). The Resource Manager will propagate the solution in a form of a system command, e.g. 'start SoftSensor^M on Device^N', to the *Commands* topic (step 7). This information is published to all subscribed Controllers through MQTT bridges. The Controller of Device^N is the only one that will react to this message and execute the system command (step 8). The execution of the command will be repeated after failure. If the command executed successfully the Controller posts an *ACTIVE* message to the *Active soft sensors* topic referencing SoftSensor^M (step 9). The Control Table Builder of the RMF leader will update the SCT (step 10) and this update is published to the *SSCM* topic (step 11). All devices are subscribed to the SSCM topic and this update is propagated to their local copies of SCT.

Scenario 2 covers the failure of a Device in the observed infrastructure (T9. Device failure). This scenario is not shown in Figure 3 since it is a repetition of the Scenario 1 workflow per device. In this version of the RMF, we have envisioned a brute-force approach for recovering of soft sensors – the next version of the RMF will have to consider workload balancing at this point.

Scenario 3 is triggered when the Health Checker (HC) of the RMF leader device detects a deviation in the behavior of a device (step 1), e.g. abnormal CPU usage on Device^N (T4/T5. CPU/RAM peak). This and similar behavior deviations can happen due to many reasons: malfunction, physical tampering, malicious attacks, etc. However, the RMF adopts a strict policy to react first and notify later. This is done so that the soft

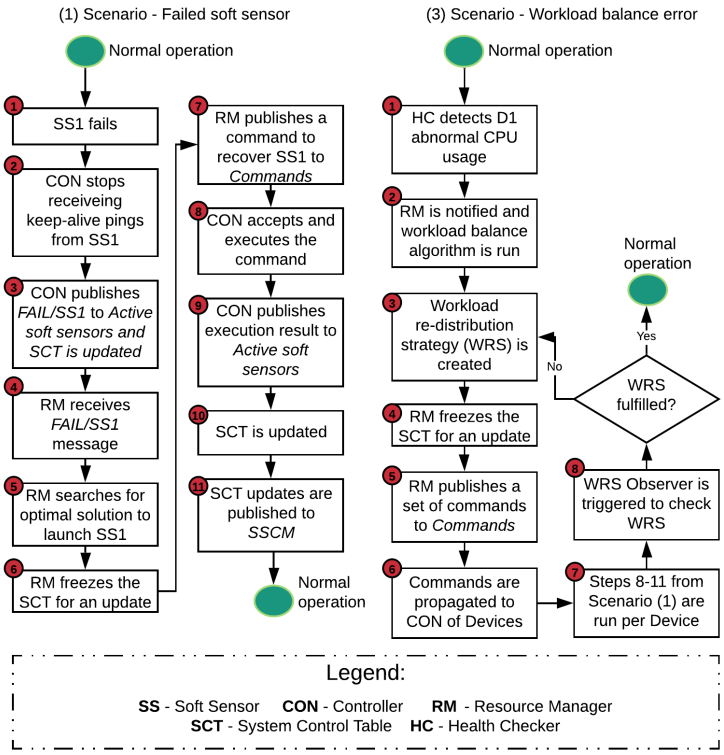

Fig. 3. Example: Fail-recover scenarios handling inside RMF.

sensors and platform business processes can converge quickly, while the platform administrator is notified of the problem with a device to manually check out the issue and issue manual workload redistribution commands if needed later. Resource Manager of RMF leader is notified through the *Commands* topic that running of the Workload Balance Algorithm (WBA) is required for Device^N (step 2). WBA analyzes the running soft sensors on Device^N and all other devices capable of running 1 or more soft sensors of Device^N. Based on this analysis the WBA outputs (step 3) a Workload Re-distribution Strategy (WRS) and creates two new controlling scripts to take care of the execution of the WRS: (a) WRS Controller and (b) WRS Observer. The WRS controller is in charge of executing the WRS through communicating with the RM to issue commands to appropriate devices. The WRS controller will trigger the WRS Observer when the WRS has been implemented. Since SCT is about to be updated shortly, it is frozen, as before, to prevent other actions (step 4). Based on the WRS the Resource Manager of RMF leader issues a set of system commands to stop/start soft sensors on different devices, through the *Commands* topic (step 5). Controllers of all devices affected by the WRS will receive these system commands, execute them, and publish new information to *Active Soft Sensors*, thus updating the SCT. These updates are pushed to all devices through the *SSCM* topic (steps 6, 7). Upon the completion of the WRS, the WRS Controller notifies the WRS Observer. The WRS Observer analyzes the WRS outputs versus the state in the SCT and concludes whether the WRS has been implemented successfully (step 8). Based on that analysis, a decision is made whether a new WRS needs to be created to compensate for the eventual problems (returning to step 3). This loop of events will execute a maximum 3 times before a MAR event is fired.

VI. Experiments, Results and Discussion

Proof-of-concept experiments are carried out in a simulated environment (written in Python) with 50 devices with Raspberry Pi 4 capabilities. Each simulated physical device has a random number of soft sensors (2–50), and a random distribution of connected physical sensors (2–10). An event (T2–T9) is triggered every 60-90 seconds and executes with a P probability (per trigger), while the simulation runs non-stop – meaning that every 90 seconds a trigger might execute. RMF leadership role switch (T1) is triggered every 300 seconds. The simulation was programmed to reset after every fail-recover scenario. A high-level architecture of the simulation is displayed in Figure 4. As part of this simulation, we want to verify that RMF successfully reacts to scenarios described in Section V.

The simulation was run for 72 hours non-stop. The results of the simulation are recorded in Table I.

Although the simulation was short, results are affirmative of the effectiveness of the proposed RMF solution.

Numbers in the last row of Table I are rounded to the higher integer. The average number of triggers executed is 190, and

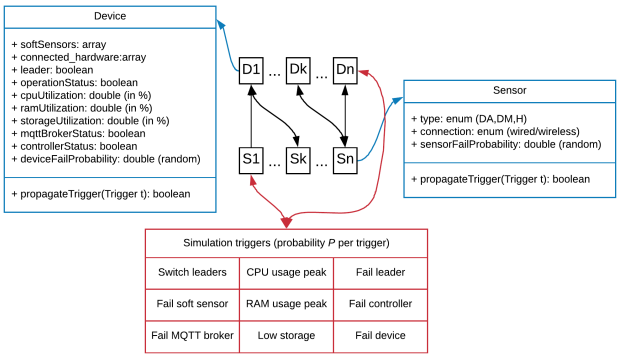

Fig. 4. RMF simulation high-level overview.

TABLE I
RMF 72-HOUR SIMULATION RESULTS.

Trigger type	Triggered no.	Resolved no.	MAR no.	Average delay (ms)
T1	299	295	4	1204
T2	214	204	10	2892
T3	260	255	5	989
T4/T5/T6	267	267	0	6620
T7	210	210	0	2476
T8	215	211	4	1001
T9	249	246	3	1705
Average	**190**	**187**	**3**	**3915**

a total of executed triggers is 1714 – in 51% of cases the trigger was not executed in the simulation framework. As an upside, on average, for every 190 triggers propagated, 187 were successfully resolved by the RMF, leading up to 98% of autonomously resolved issues inside the observed simulated IoT platform.

However, several downsides need to be accounted for. The total number of manual actions required (MAR) over the 72 hours was 26 – this number is still high for an RMF to be considered automated and autonomous. In the simulation MAR was in most cases caused by 2 issues: **Case (1)** RMF leadership role change signal synced with the trigger causing a failure to update the SCT properly; **Case (2)** Multiple device failure occurs and a 1st layer soft sensor cannot be started due to missing connectivity with other viable devices. While case (2) is out of RMF's reach and impact, case (1) has to be accounted for. Next, triggers T4/T5/T6 are associated with Health Checker (HC) and Workload Balance Algorithm (WBA). Although they were resolved in all of the cases the average delay is much higher than for other triggers. That is caused by the loop in the scenario handling strategy – only 29% of triggers T4/T5/T6 were resolved in the first loop, 60% in the second and 11% in the final (third) loop. This resulted in a high average delay, yielding a high average delay in total (3915ms). Scenarios handling triggers T4/T5/T6 need to be handled with more sophistication and WBA needs to be re-calibrated to create finer Workflow Re-distribution Strategies (WRS). Finally, in general, the average delays for all triggers are high for an industry-grade IoT platform and need to be cut approximately ten-fold to be considered for such platforms, especially for critical use-cases [19], [20].

VII. Conclusion and Future Work

In this paper, we have described a Resource Management Framework for edge computing systems handling fail-rescue of software components, infrastructure and software health checking, and platform workload re-distribution. The RMF with workflows and fail-recover scenarios described in Sections III and V is a generic framework best-suited to IoT platforms with event-driven, publish-subscribe communication mechanisms.

As part of the future work we will focus on increasing the sophistication of the RMF through three aspects: (1) inspect and reduce MAR, (2) refine the Workload Balance Algorithm and Workload Re-distribution Strategy Controller and Observer components, and (3) work to reduce RMF's average delay. A milestone in RMF implementation and a part of our future work is the integration of RMF with a commercial IoT platform.

Acknowledgment

The authors acknowledge the financial support of the Ministry of Education, Science and Technological Development of the Republic of Serbia (Grant No. 620 451-03-68/2020-14/200125) and VizLore Labs Foundation (Novi Sad, Serbia).

References

[1] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, "Internet of things for smart cities," *IEEE Internet of Things journal*, vol. 1, no. 1, pp. 22–32, 2014.

[2] D. Kyriazis, T. Varvarigou, D. White, A. Rossi, and J. Cooper, "Sustainable smart city iot applications: Heat and electricity management & eco-conscious cruise control for public transportation," in *2013 IEEE 14th International Symposium on" A World of Wireless, Mobile and Multimedia Networks"(WoWMoM)*. IEEE, 2013, pp. 1–5.

[3] D. Minoli, K. Sohraby, and B. Occhiogrosso, "Iot considerations, requirements, and architectures for smart buildings—energy optimization and next-generation building management systems," *IEEE Internet of Things Journal*, vol. 4, no. 1, pp. 269–283, 2017.

[4] T. Renner, M. Meldau, and A. Kliem, "Towards container-based resource management for the internet of things," in *2016 International Conference on Software Networking (ICSN)*. IEEE, 2016, pp. 1–5.

[5] B. Manate, T.-F. Fortis, and V. Negru, "Optimizing cloud resources allocation for an internet of things architecture," *Scalable Computing: Practice and Experience*, vol. 15, no. 4, pp. 345–355, 2014.

[6] H.-W. Kim, J. H. Park, and Y.-S. Jeong, "Efficient resource management scheme for storage processing in cloud infrastructure with internet of things," *Wireless Personal Communications*, vol. 91, no. 4, pp. 1635–1651, 2016.

[7] A. Sehgal, V. Perelman, S. Kuryla, and J. Schonwalder, "Management of resource constrained devices in the internet of things," *IEEE Communications Magazine*, vol. 50, no. 12, pp. 144–149, 2012.

[8] S. Li, N. Zhang, S. Lin, L. Kong, A. Katangur, M. K. Khan, M. Ni, and G. Zhu, "Joint admission control and resource allocation in edge computing for internet of things," *IEEE Network*, vol. 32, no. 1, pp. 72–79, 2018.

[9] Y. Mao, J. Zhang, S. Song, and K. B. Letaief, "Stochastic joint radio and computational resource management for multi-user mobile-edge computing systems," *IEEE Transactions on Wireless Communications*, vol. 16, no. 9, pp. 5994–6009, 2017.

[10] J.-J. Chen, J.-M. Liang, and Z.-Y. Chen, "Energy-efficient uplink radio resource management in lte-advanced relay networks for internet of things," in *2014 international wireless communications and mobile computing conference (IWCMC)*. IEEE, 2014, pp. 745–750.
[11] C.-H. Hong and B. Varghese, "Resource management in fog/edge computing: A survey," *arXiv preprint arXiv:1810.00305*, 2018.
[12] A. Alreshidi and A. Ahmad, "Architecting software for the internet of thing based systems," *Future Internet*, vol. 11, no. 7, p. 153, 2019.
[13] O. Leiba, Y. Yitzchak, R. Bitton, A. Nadler, and A. Shabtai, "Incentivized delivery network of iot software updates based on trustless proof-of-distribution," in *2018 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE, 2018, pp. 29–39.
[14] L. Fortuna, S. Graziani, A. Rizzo, and M. G. Xibilia, *Soft sensors for monitoring and control of industrial processes*. Springer Science & Business Media, 2007.
[15] M. Tosic, O. Ikovic, and D. Boskovic, "Soft sensors in wireless networking as enablers for sdn based management of content delivery," in *2016 39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. IEEE, 2016, pp. 559–564.
[16] V. Karagiannis, P. Chatzimisios, F. Vazquez-Gallego, and J. Alonso-Zarate, "A survey on application layer protocols for the internet of things," *Transaction on IoT and Cloud computing*, vol. 3, no. 1, pp. 11–17, 2015.
[17] M. Singh, M. Rajan, V. Shivraj, and P. Balamuralidhar, "Secure mqtt for internet of things (iot)," in *2015 Fifth International Conference on Communication Systems and Network Technologies*. IEEE, 2015, pp. 746–751.
[18] N. Naik, "Choice of effective messaging protocols for iot systems: Mqtt, coap, amqp and http," in *2017 IEEE international systems engineering symposium (ISSE)*. IEEE, 2017, pp. 1–7.
[19] A. Al-Fuqaha, A. Khreishah, M. Guizani, A. Rayes, and M. Mohammadi, "Toward better horizontal integration among iot services," *IEEE Communications Magazine*, vol. 53, no. 9, pp. 72–79, 2015.
[20] S. Mubeen, P. Nikolaidis, A. Didic, H. Pei-Breivold, K. Sandström, and M. Behnam, "Delay mitigation in offloaded cloud controllers in industrial iot," *IEEE Access*, vol. 5, pp. 4418–4430, 2017.