Javed, Asad; Malhi, Avleen; Framling, Kary

# Edge Computing-based Fault-Tolerant Framework

*Published in:*
2020 International Wireless Communications and Mobile Computing, IWCMC 2020

*DOI:*
10.1109/IWCMC48107.2020.9148269

Published: 01/06/2020

*Document Version*
Peer-reviewed accepted author manuscript, also known as Final accepted manuscript or Post-print

*Please cite the original version:*
Javed, A., Malhi, A., & Framling, K. (2020). Edge Computing-based Fault-Tolerant Framework: A Case Study on Vehicular Networks. In *2020 International Wireless Communications and Mobile Computing, IWCMC 2020* (pp. 1541-1548). Article 9148269 IEEE. https://doi.org/10.1109/IWCMC48107.2020.9148269

# Edge Computing-based Fault-Tolerant Framework: A Case Study on Vehicular Networks

Asad Javed*, Avleen Malhi* and Kary Främling*†
*Department of Computer Science, Aalto University, Espoo, Finland
†Department of Computer Science, Umeå University, Umeå, Sweden
{firstname.lastname}*@aalto.fi / †@umu.se

***Abstract*—With the evolution of vehicular networks, the Intelligent Transportation System (ITS) has emerged as a promising technology for autonomous road transport. For a successful deployment of ITS, security and reliability are the most challenging factors to be tackled to ensure Vehicle-to-Infrastructure (V2I) and Infrastructure-to-Infrastructure (I2I) communications. Due to unreliable communications in vehicular networks, implementing fault-tolerant techniques for the Road Side Unit (RSU) infrastructure is an imperial need. Within this context, the contributions of this paper are twofold: (i) we propose a distributed fault-tolerant framework for V2I and I2I communications based on edge computing to resolve hardware- and network connectivity-based failures. The fault tolerance issue is addressed by employing open messaging standards as a subscription-based data replication solution at the edge. We also adopt Kubernetes for the fault-tolerant management, combined with high-availability mechanism, allowing automatic reconfiguration of the data processing pipeline; and (ii) we implement a demonstrator system for vehicular networks-based smart mobility to assess fault tolerance capabilities. The experimental results show that our proposed framework dynamically tolerates RSU-related failures during the vehicular communication phase.**

***Index Terms*—Internet of Things, vehicular network, roadside unit, fault tolerance, edge computing**

## I. Introduction

The Internet of Things (IoT) technology has led to an increased emphasis on vehicular networks in which vehicles and smart devices communicate with each other by exchanging user/vehicle data [1]. During the past decades, the expansion of road mobility has over-flooded major cities with vehicles, causing an increase in accidents and traffic congestion with the resultant negative impact on the economy, environment, and public health [2][3]. The Intelligent Transportation System (ITS) has emerged as a part of IoT to enable the collection, processing, and dissemination of valuable information in vehicular networks by exploiting Vehicle-to-Infrastructure (V2I) and Infrastructure-to-Infrastructure (I2I) communications, thereby providing safer and more coordinated driving [4]. Such networks often need an Ethernet-based setup between the Road Side Unit (RSU) stations despite having an ad hoc connectivity for V2I communication. These RSUs operate as wireless Access Points (APs), managing data transmission and broadcasting to other wireless networks. However, the real-time and fault-tolerant communication between the vehicles and RSU infrastructure is an important aspect that still needs to be tackled in ITS, since the failure of any RSU could compromise the reliability of the entire system. This raises the following four challenges: (i) The RSU can be physically damaged by some malicious activity or other harsh environments; (ii) the network connectivity during V2I and I2I communications may be temporarily cut off, affecting immediate data communication; (iii) the vehicular network should be sufficiently scalable to adapt to the increasing number of vehicles; and (iv) data processing needs to be performed closer to the data sources to minimize network latency. In this regard, the fault-tolerant design for RSU infrastructure is required to ensure the reliability of the network. In many vehicular systems, the data are transmitted to the remote server for processing as the computational capabilities of the vehicles onboard unit are limited. It is often necessary to process real-time data by allowing the cloud-enabled networks to run either on local vehicle/RSU terminal or can be off-loaded to the remote cloud. Edge computing has emerged as a promising solution for pushing the cloud service to the edge of the network [5][6], as illustrated in Fig. 1. Thus, providing data computation offloading closer to the vehicle terminals.

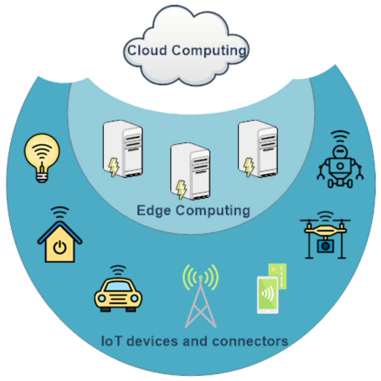


Fig. 1: Edge computing in IoT

Within this context, **the major contributions of this work are: (i)** to propose a distributed fault-tolerant framework for V2I and I2I communications, combined with the modular characteristics of edge-centric computing, providing a highly dynamic and fault-tolerant design. It adopts state-of-the-art cloud technologies including Docker, Kubernetes, and the Open Group standards: Open Messaging Interface (O-MI) and Open Data Format (O-DF) [7][8], which are both managed as

O-MI node[1]; and **(ii)** to implement a demonstrator system of vehicular networks-based smart mobility in the Aalto University lab containing five RSUs, which are then analyzed by considering the four possible real-case scenarios.

The rest of the paper is organized as follows. Section II presents the related work in ITS domain. Section III proposes a new fault-tolerant framework based on edge computing along with the use case scenarios and implementation in Section IV. Section V displays the experimental results for analyzing fault tolerance, followed by the conclusion in Section VI.

## II. Related Work

Several studies have been conducted in the literature concerning the fault tolerance, reliability, and security of vehicular networks. Faults can be mainly classified into three categories: transient, intermittent, and permanent [9][10]. The transient faults eventually disappear without any apparent intervention, whereas the intermittent faults are difficult to diagnose. On the other hand, the permanent faults will always remain in the network unless the external administrator repairs/removes them. These permanent faults are considered crucial in the literature, which needs to be resolved for correct system execution. In this regard, TABLE I provides state-of-the-art comparative analysis of our framework with some of the earlier fault-tolerant techniques based on various IoT characteristics. These selected papers in TABLE I are well aligned with our proposed solution. As can be seen, a fault-tolerant distributed path recommendation protocol is proposed by Younes et al. [9] for tolerating faults which occur between nodes and communication links. Similarly, Lygeros et al. [11] develop a fault-tolerant architecture to resolve network failures in the automated highway system. Once a fault has occurred, its classification is performed by considering the capabilities of the vehicle or roadside unit. Hiraiwa et al. [12] propose a communication control scheme for solving the handover mechanism in the roadside network which may lead to end-to-end throughput degradation. To enhance the fault-tolerant behavior of RSUs, Almeida et al. [13] propose a replica determination mechanism which ultimately impacts the real-time properties of vehicular networks. Another architecture is proposed by Almeida et al. [14] to improve the network dependence on roadside infrastructure for real-time data guarantee. For secure communications, Malhi and Batra [15][16] propose a security framework which addresses most of the security requirements, also providing reliability to the vehicular system. To offer a low latency solution on the edge, Zhang et al. [17] present an efficient predictive scheme based on mobile edge computing for vehicular networks in which the tasks are off-loaded to the edge servers. Further, Liu et al. [18] propose an edge computing-based architecture by integrating different types of access technologies to enable low latency and high-reliability communication. Although these solutions fulfill application-specific requirements and offers better responsiveness, there is no mechanism for handling hardware fault tolerance.

[1]O-MI reference implementation developed by Aalto University: https://github.com/AaltoAsia/O-MI, last accessed April, 2020

TABLE I: Comparative Analysis: ✓ means supported, ✗ means not supported, and $\rho$ means partially supported.

| **IoT Characteristics** | **[18]** | **[11]** | **[9]** | **[13] [14]** | **Proposed** |
|---|---|---|---|---|---|
| Scalability | ✓ | ✗ | ✗ | ✓ | $\rho$ |
| Layered design | ✓ | ✓ | ✗ | ✓ | ✓ |
| High availability | ✗ | ✗ | ✗ | ✓ | ✓ |
| Fault tolerance (data) | ✗ | ✗ | ✓ | ✗ | ✓ |
| Fault tolerance (network) | $\rho$ | ✓ | ✓ | ✓ | ✓ |
| Fault tolerance (server) | ✗ | ✓ | ✓ | ✓ | ✓ |
| Container virtualization | ✗ | ✗ | ✗ | ✗ | ✓ |
| Data replication | ✗ | ✗ | ✗ | ✓ | ✓ |

To the best of our knowledge, none of the previous mechanisms in vehicular networks have resolved the fault tolerance issues on data, network, and node level, while enhancing the scalability and availability of the system. Therefore, in this paper, we propose a distributed fault-tolerant framework, combined with the edge computing capabilities, to handle network- and RSU-based failures, which are dynamically tolerated at each level of the vehicular communication pipeline.

## III. Fault-Tolerant Framework

We propose a generic fault-tolerant framework for smart applications, accompanied by edge computing, to tackle hardware- and network connectivity-based failures. This framework is designed to dynamically overcome node, network, and data fault tolerance by allowing automatic reconfiguration of the data processing pipeline and enabling data processing in an edge-centric environment (i.e., closer to the data sources). Additionally, data computation on the edge ensures low latency and reduces a large amount of network bandwidth. The framework also considers the limited resources available at the edge by employing software containers. It should also be noted that the proposed design is hardware-based, which incorporates the underlying cloud technologies rather than algorithmic aspects. Our proposed framework is composed of four layers: A) Device, B) Communication, C) Management, and D) Application layer, as illustrated in Fig. 2. This layered design is inspired by our earlier fault-tolerant IoT architecture [19], ensuring: (i) subscription-based local data replication on the edge, (ii) fault-tolerant management for data processing pipeline, and (iii) application-level software portability.

### *A. Device Layer*

The Device Layer of our proposed framework provides various computing devices, sensors, and other information systems for collecting a large amount of real-time data. These devices are capable of performing computation on the logical extremes of the network by executing specified cloud technologies. In the case of sensors with no data processing and storage capability, the layer can integrate these sensors with other external cost-efficient processing module to perform heavy computation and other processing tasks.

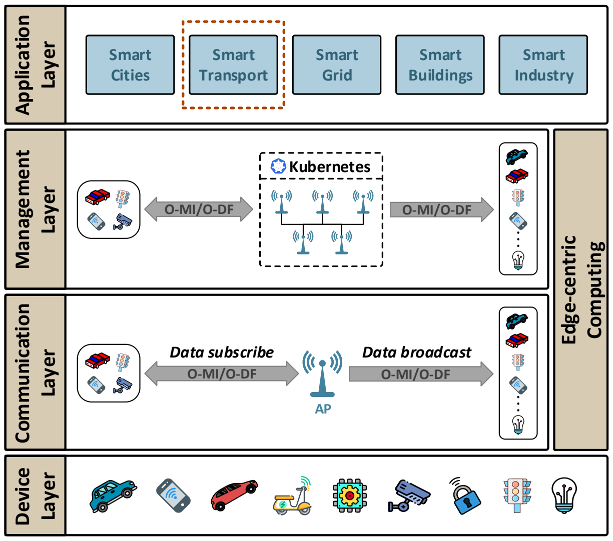


Fig. 2: A bottom-up overview of fault-tolerant distributed framework with edge-centric computing

### B. Communication Layer

The Communication Layer is responsible for providing a unified publish/subscribe data communication pipeline which publishes real-time data, replicates them, and stores them locally on the edge. Other nodes can also subscribe to these data and publish them based on the application requirements. In this layer, the devices are subscribed to an Access Point (AP) for obtaining the latest sensor values which then broadcasts to other nearby devices. Thus, resolving network and data fault tolerance by replicating the data in the entire cluster. We adopt O-MI and O-DF messaging standards [7][8] as a unified subscription-based data replication solution for implementing the needed functionality. These standards provide peer-to-peer communication and real-time interaction between heterogeneous systems. They reside independently at the communication and format levels of the OSI Application layer in which O-MI provides a generic open API for transporting data payloads in nearly any format. The complementary O-DF standard is currently the most common text-based payload format due to its flexibility [20]. O-DF is defined as a simple ontology, specified using XML schema, which is sufficiently generic for representing *any object* in the IoT. In contrast to the Web which uses HTTP to transmit HTML-coded information mainly intended for human users, O-MI is designed to transmit O-DF payload that is dedicated for information systems. TABLE II lists the basic operations for sending O-MI requests in which we consider “Read (Subscription)” operation to provide replication-based mechanism.

### C. Management Layer

The Management Layer enables the deployment of data processing pipeline and configures it to operate as a fault-tolerant system. This layer simplifies the problem of node failures by rescheduling failed processing stages on other available nodes, providing node fault tolerance on the hardware-level. We use the Kubernetes framework to orchestrate the placement of processing stages, as it enables fault-tolerant management and ensures high-availability solution. One of the key design ideas of Kubernetes is to have no single point of failure. All configuration data is stored in a replicated fashion, allowing Kubernetes to survive any single node failure (e.g., in the cluster of three nodes). Hence, this layer offers a fault tolerance functionality that is fully transparent to the application programmer.

### D. Application Layer

The Application Layer of our proposed framework is capable of providing various IoT and ITS applications that can be integrated with rest of the layers to ensure fault-tolerant behavior. In this article, we focus on the vehicular network-based smart mobility application.

## IV. Case Study: Smart Mobility in Vehicular Networks

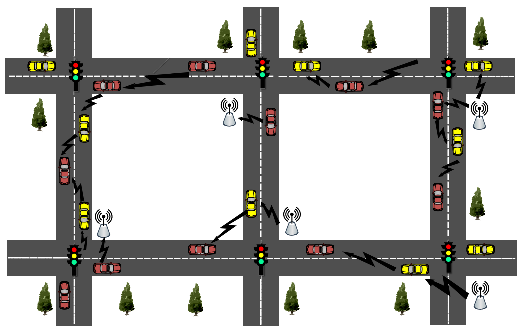
The fault tolerance capabilities of our proposed framework are applied to a smart mobility use case, which deals with safer and more coordinated V2I and I2I communications. In this case study, RSUs play an important role due to their capabilities of: (i) delivering valuable information to vehicles, (ii) forwarding received messages to the final recipients, and (iii) providing Internet access to vehicles. Indeed, the RSU infrastructure is configured to extend vehicle coverage and improve network performance [22]. These RSUs are controlled by the trusted authorities’ middleware, which assigns a unique ID to each RSU along with the communication range. As a result, each RSU handles the vehicles under its dedicated range, and communicates with other RSUs through the secure Ethernet channel. Furthermore, the communication between vehicles are performed via IEEE 802.11p technology in the bandwidth spectrum of 5.850 to 5.925 GHz. In this regard, Section IV-A describes the fault tolerance scenarios, followed by the implementation details in Section IV-B.

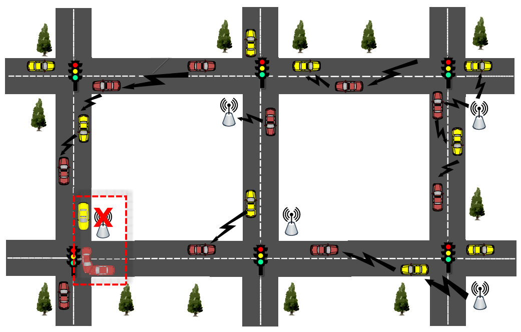
### A. Scenarios Description

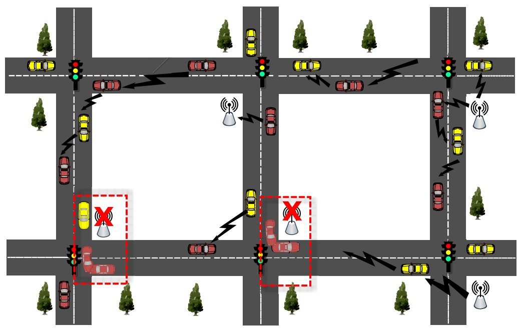
We consider the four possible smart mobility scenarios in the lab environment due to the limitations of hardware and software resources. Fig. 3(a) demonstrates a normal scenario in which the vehicles communicate with their nearest RSU to retrieve traffic information. In the case of RSU failure, the vehicles will not receive recent updates, thus potentially leading to disastrous situations. RSUs can malfunction due to transient power spikes, unexpected power interruptions, software failures, or intermittent network dis-connectivity. We describe one of the failure scenarios in Fig. 3(b) in which a single RSU fails, thereby halting communication with other available nodes. Consequently, the data load and computation handled by this RSU need to be distributed to other RSUs. Another scenario is depicted in Fig. 3(c) in which two RSUs are down, causing the vehicles to collide with each other. The same is the case in Fig. 3(d), which shows the communication link failing between two RSU nodes. This scenario represents

TABLE II: O-MI Basic Operations [21]

| Operation | Description |
|---|---|
| 1- Write | Used to write information updates from sensors, events, or other devices to O-MI nodes. |
| 2- One-Time Read | Used to retrieve immediate or old information from the O-MI nodes. |
| 3- Read (Subscription) | A specific read operation for retrieving information at regular intervals. Two types of subscription can be performed: • *Subscription with callback address:* The requested data are sent to the callback address with a specified interval (interval-based and event-based are supported). • *Subscription without callback address:* The data are stored on the subscribed node until the subscription is valid. The data can then be retrieved (polled) by issuing a new read request. |
| 4- Cancel | Used to cancel subscription before it expires. |
| 5- Delete | Used to delete parts of O-DF hierarchy. |


(a) Scenario-1: A normal scenario


(b) Scenario-2: A scenario with single RSU failure


(c) Scenario-3a: A scenario with two RSU failures

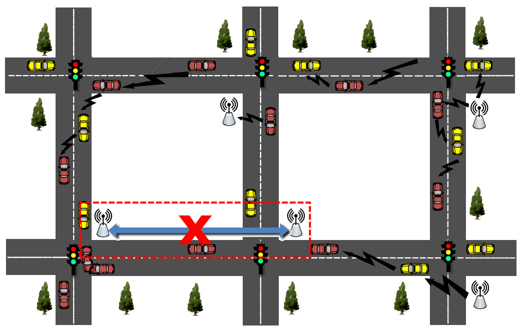

(d) Scenario-3b: A scenario in which communication link is down between two RSUs

Fig. 3: Real-case scenarios for the smart mobility use case

the data connectivity issues that may lead to system degradation in terms of data sharing and broadcasting.

## B. Implementation Details

A demonstrator system has been implemented in the Aalto University lab by using five Raspberry Pi (RPi) boards, together acting as a clustered system of five RSUs. We select RPi for testing purpose as it is a fully functional ARM-based Linux edge device, allowing the same Linux-based application software to run without any modifications. Fig. 4(i) shows a running implementation of the selected use case in which we have five RPi nodes, configured as RSUs. These nodes are located at fixed positions and each RSU is assumed to have dedicated coordinates, accordingly at $(100, 50)$ $(300, 50)$ $(450, 200)$ $(200, 250)$ $(50, 200)$, in the square grid. In the implementation, the Communication Layer is realized by running O-MI/O-DF standards inside Docker containers, providing software portability and replication-based communication pipeline. The Management Layer is logically observed by adopting the Kubernetes high availability framework, which configures the three Kubernetes master and two worker nodes. We use the *kubeadm* tool to spread Kubernetes on five RPi nodes. It performs the actions necessary to configure a minimum viable cluster in a user-friendly manner. TABLE III lists the specifications of hardware/software tools used in our implementation.

## V. Performance Evaluation and Discussions

For performance assessment, we simulate the smart mobility scenarios of Fig. 3, then analyze the fault-tolerant behavior, throughput, and latency measurements. The experimental setup, depicted in Fig. 4, consists of four parts: (i) Our frame-

TABLE III: Specifications of hardware/software tools

| No. of nodes | RPi | OS | Docker | Kubernetes |
|---|---|---|---|---|
| **5** | v3 | Raspbian 9.4 | v18.09 | v1.13.3 |

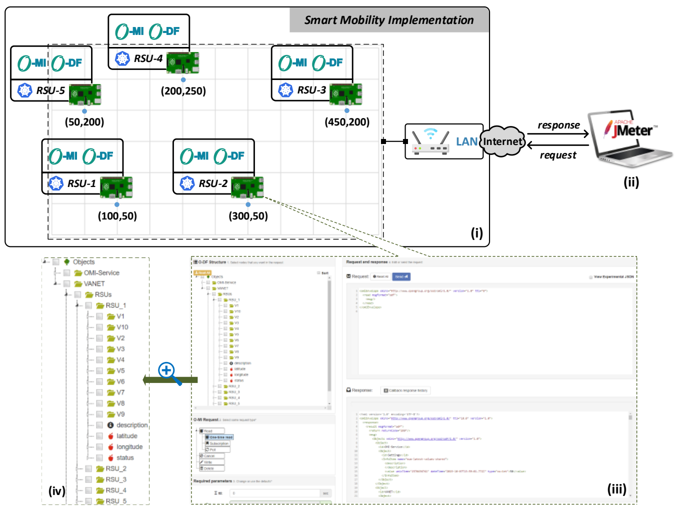


Fig. 4: Running implementation of our framework with five RPi nodes connected through LAN switch

work as described in Section IV-B; (ii) the open-source software Apache JMeter[2] (v4.0) runs on HP EliteBook Windows laptop with a memory of 8GB and Intel Core i5 2.40GHz CPU, which enables us to simulate V2I and I2I communications. For evaluation purposes, we only consider 10 vehicles in the use case implementation, which send concurrent requests through JMeter; (iii) the user interface of one of the O-MI nodes which shows the O-DF objects hierarchy that we formulate with respect to our case study; (iv) the magnified version of O-DF hierarchy in which each vehicle has a unique ID assigned by the trusted authority middleware.

## A. *Fault Tolerance Assessment*

We consider iptables as an assessment tool for testing the fault-tolerant behavior. This tool allows the configuration, maintenance, and inspection of a set of IP packet rules for the network. Such rules enable the system to accept or reject specific network traffic. The following command is used to disable network connectivity:

*iptables -A INPUT -j DROP*

TABLE IV lists the observations for four different cases in which the network connectivity between RPi nodes is temporarily cut off using the aforementioned command. For each case, the results of CPU and memory usage are collected in Fig. 5 for the fault-tolerant analysis as these become directly affected by a single or multiple-node failure. These graphs provide the time in MM:SS format on the horizontal axis and the aggregated CPU cores, which are dedicated to the entire cluster, on the vertical axis. We have in total 20 CPU cores in which each RPi has 4 cores. Similarly, the memory usage graphs provide a similar time duration on the horizontal axis and the aggregated memory in-use (out of total 5 Gbytes) on the vertical axis. These results are plotted on the Kubernetes Dashboard (web-based UI) using the Heapster metrics tool, which enables cluster monitoring and performance analysis for Kubernetes. Fig. 5(a) displays a normal behavior of all five RSUs when they are active. In the beginning (at around 14:41), the cluster initializes data communication pipeline through O-MI/O-DF read/write requests, thus increasing CPU usage from 2.75 cores to around 10 cores (at time 14:43). Similarly, the cluster occupies around 3.91 Gbytes out of 5 Gbytes as shown in Fig. 5(b). When one RSU fails, the CPU and memory usage decreases to around 8.25 cores and 2.93 Gbytes (at time 14:52) in Fig. 5(c) and Fig. 5(d), respectively. Consequently, the system continues to operate and the data communication remains uninterrupted through other available RSUs. Fig. 5(e) and Fig. 5(f) show the behavior of two RSUs when they are inactive. Both the CPU and memory usage decrease significantly to around 2.0 cores and 2.1 Gbytes, respectively. The system is still able to handle new requests as our implementation tolerates two RSUs failure. Similarly, Fig. 5(g) and 5(h) correspond to the case when both the RSUs come back online. The CPU and memory usage increases to 7.0 cores and 4.1 Gbytes, respectively, at around 15:10 to achieve normal behavior. As a consequence, the cluster continues to perform data processing tasks by self-adapting and independently recovering from the failures.

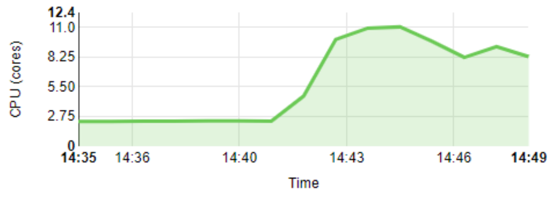
## B. *Throughput Analysis*

In addition to the CPU and memory usage, we calculate the throughput for analyzing fault-tolerant behavior. The graphs in Fig. 6 and Fig. 7 represent the network traffic for O-MI and Kubernetes providing throughput in bytes/s over the period of time. We execute *tcpdump* on each RPi to capture and filter the network traffic. Similarly, this command is also executed on our laptop (where we execute Apache JMeter), which is plotted in Fig. 6(a). As can be seen, all five RSUs handle O-MI requests initially with the average throughput of around 3000 bytes/s. When one RSU fails at around $80s$, the O-MI traffic is diverted to other available nodes and the vehicles continue to publish and consume traffic information. When the failed RSU comes back online at around $260s$, the RSU-1 begins to handle new requests. Likewise, Fig. 6(b) shows the throughput of Kubernetes API traffic for the three RSU nodes, which proves that the system will not halt and the requests can be answered by other RSUs between $80s$ to $300s$. On the other hand, Fig. 6(c) and Fig. 6(d) show the O-MI communication that are captured inside RSU-1 and RSU-2, respectively. As seen in Fig. 6(c), RSU-1 handles requests till $80s$ and afterwards the node becomes inactive, thus no further communication is performed between $80s$ and $260s$. Nevertheless, when RSU-1 comes online at around $260s$, the system continues to operate and the data from other RSUs are instantly replicated to RSU-1. This shows the data fault tolerance which renders the communication pipeline active.

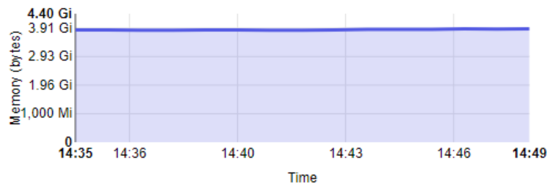
[2][Online]. http://jmeter.apache.org/, last accessed March, 2020

TABLE IV: Four cases for analyzing fault tolerance: ✓means active, × means inactive

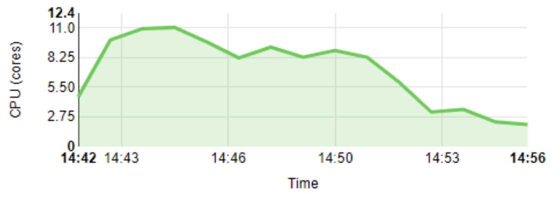
| **Case** | **RSU-1** | **RSU-2** | **RSU-3** | **RSU-4** | **RSU-5** | **Observations** |
|---|---|---|---|---|---|---|
| **1** | ✓ | ✓ | ✓ | ✓ | ✓ | The default condition in which all RSUs are up and running. |
| **2** | × | ✓ | ✓ | ✓ | ✓ | The cluster tolerates one node failure. Other master nodes handle the API requests. |
| **3** | × | ✓ | ✓ | × | ✓ | No effect on the high availability mechanism. However, the cluster reduces to three nodes. |
| **4** | ✓ | ✓ | ✓ | ✓ | ✓ | The cluster continues to operate once the nodes become online again. |



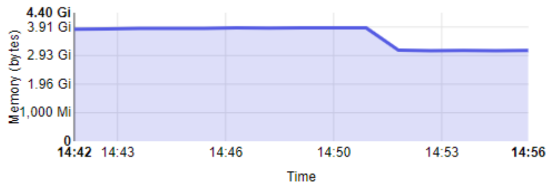
(a) Case 1: CPU usage (normal case)



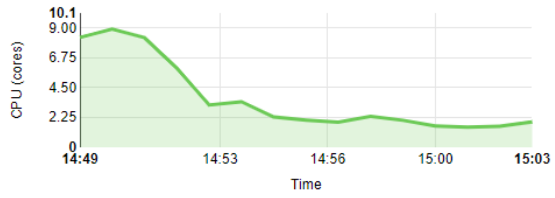
(b) Case 1: Memory usage (normal case)



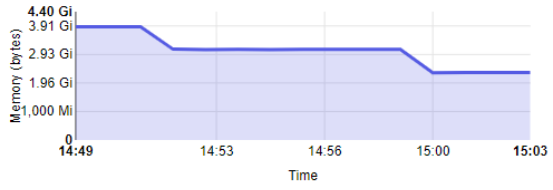
(c) Case 2: CPU usage (RSU-1 is inactive)



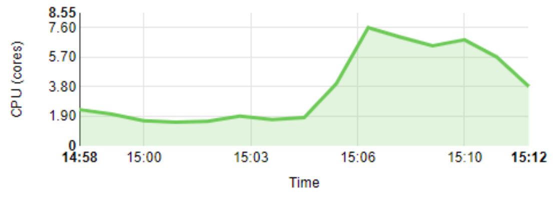
(d) Case 2: Memory usage (RSU-1 is inactive)



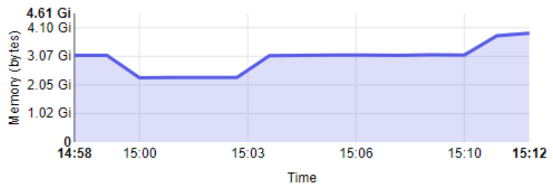
(e) Case 3: CPU usage (RSU-1 and RSU-4 are inactive)



(f) Case 3: Memory usage (RSU-1 and RSU-4 are inactive)



(g) Case 4: CPU usage



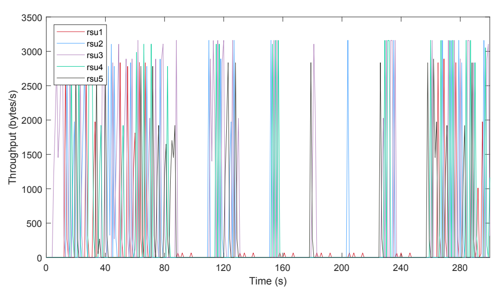
(h) Case 4: Memory usage

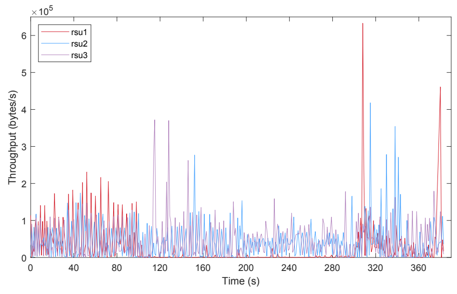
Fig. 5: CPU and Memory usage for fault tolerance cases. CPU (cores): the aggregated sum of active cores for the entire Kubernetes edge cluster, Memory (bytes): the aggregated memory in-use of the edge cluster, and Time: the time duration in MM:SS (minutes:seconds)

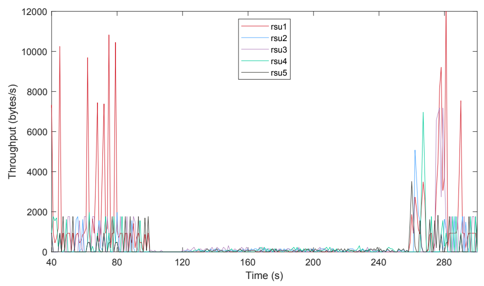
The same is the case in Fig. 6(d) in which other RSUs handle requests and the data is replicated to RSU-1 at around $260s$. This graph is only plotted to show that the communication pipeline is active at all times and the requests are handled without any interruption. Furthermore, Fig. 7(a) and Fig. 7(b) plots the O-MI traffic for Scenario-3. Although both RSU-1 and RSU-4 are down between $40s$ to $180s$, the system still handles the requests and the traffic information can be consumed from other RSU nodes. Once both the RSUs have restarted at around $180s$, the data is automatically replicated to the entire vehicular system.

## C. *Latency Measurements*

The experimental results of fault tolerance impact on latency are illustrated in Fig. 8. These graphs represent a boxplot providing minimum, $1^{st}$ quartile, median, $3^{rd}$ quartile, and maximum of the latency calculated over each 100-second period. In this paper, latency is the time taken to execute a Python script from JMeter, which includes concurrent requests (in O-DF) for reading and writing random traffic information along with a delay of 2 seconds. Fig. 8(a) shows a normal scenario in which the latency is less than 3s in each 100-second period. As compared to Fig. 8(a), Fig. 8(b) shows


(a) RSU-1 is down. O-MI traffic diverts to other RSUs


(b) Kubernetes API traffic


(c) The O-MI communication traffic captured in RSU-1

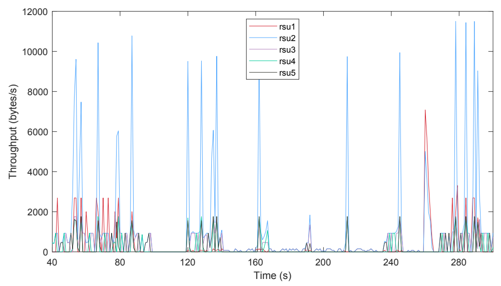
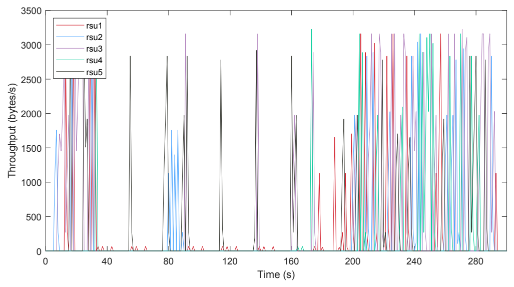

(d) The O-MI communication traffic captured in RSU-2

Fig. 6: The throughput for Scenario-1 and Scenario-2 in which RSU-1 becomes inactive after the normal operation


(a) RSU-1 and RSU-4 are down. O-MI traffic diverts to other RSUs

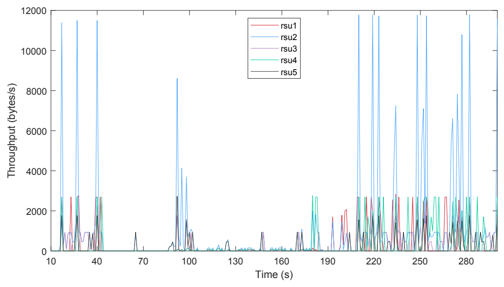

(b) The O-MI communication traffic captured in RSU-2

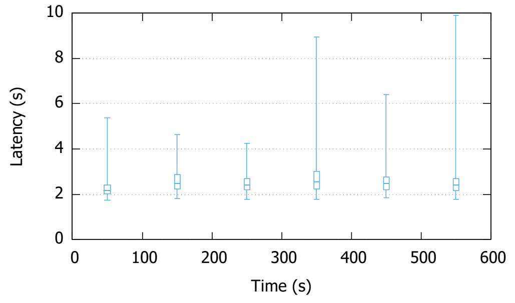
Fig. 7: The throughput for Scenario-3a in which RSU-1 and RSU-4 are inactive

the behavior of one RSU when it is inactive [180:360$s$], the latency increases accordingly to 23$s$ in the interval [200:300$s$]. This high latency is due to the concurrent read/write requests sent by 10 vehicles, which are now being handled by four RSUs (instead of five nodes). Thus, the load increases as well. Overall, it can be seen from latency measurements that the vehicular system will not halt in the case of one or more RSU failures, and the latencies return to normal once the RSU becomes active again.

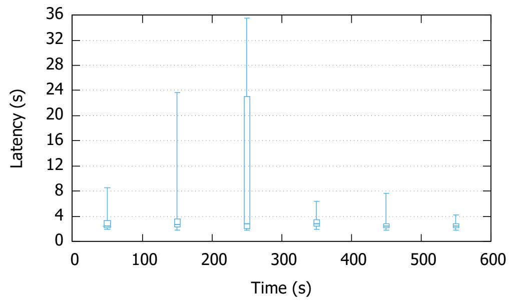
To conclude, it has been observed that the proposed framework is capable of tolerating RSU-based failures. This can be seen in the experimental results of fault tolerance, throughput, and latency measurements for the smart mobility use case. Let us note that the selection of this specific case study and the software/hardware tools reduced the possibilities of comparing our solution performance with the earlier discussed techniques in the literature. Therefore, due to the limitations of hardware resources, we implemented our framework by considering the four possible real-case scenarios in the lab environment and tested the robustness of the proposed system.

## VI. Conclusion and Future Work

This paper proposes a distributed fault-tolerant framework based on edge computing for V2I and I2I communications in the vehicular networks. It adopts state-of-the-art cloud technologies including Docker, Kubernetes, and the Open Messaging Interface (O-MI) standards, which are also deployed for edge computing. This framework consists of four layers: (i) Device, (ii) Communication, (iii) Management, and (iv) Application layer. An edge-based layered design for the RSU infrastructure provides fault-tolerant management and



(a) Latency in Scenario-1



(b) Latency in Scenario-2. RSU-1 is down between 180:360s

Fig. 8: Fault tolerance impact on latency

allows the data to be processed on the edge (i.e., closer to the data sources), enhancing the fault tolerance and reliability of the network. The fault tolerance capabilities of our proposed framework are then evaluated by considering a smart mobility use case, which is implemented as a demonstrator system of five RPi nodes (operated as five RSUs) in the Aalto University lab. The experimental results in terms of fault tolerance cases, latency, and throughput have validated the effectiveness of our framework in the vehicular networks.

In future work, the fault tolerance for Vehicle-to-Vehicle (V2V) communication will be addressed, particularly the real-time traffic information that exchanges between vehicles. Further, the framework will be evaluated for network and resources scalability by simulating more than 10 vehicles.

## Acknowledgements

This work has received funding from the European Union's Horizon 2020 research and innovation programme (grant 688203) and Academy of Finland (Open Messaging Interface; grant 296096). The authors would like to thank Aalto ASIA team at the Computer Science department at Aalto University.

## References

[1] N. Shrestha, S. Kubler, and K. Främling, "Standardized Framework for Integrating Domain-Specific Applications into the IoT," in *2014 International Conference on Future Internet of Things and Cloud*. IEEE, 2014, pp. 124–131.

[2] M. Alam, J. Ferreira, and J. Fonseca, "Introduction to Intelligent Transportation Systems," in *Intelligent Transportation Systems*. Springer, 2016, pp. 1–17.

[3] G. Dimitrakopoulos and P. Demestichas, "Intelligent Transportation Systems," *IEEE Vehicular Technology Magazine*, vol. 5, no. 1, pp. 77–84, 2010.

[4] F. Perry, K. Raboy, E. Leslie, Z. Huang, D. Van Duren *et al.*, "Dedicated Short-Range Communications Roadside Unit Specifications," United States. Dept. of Transportation, Tech. Rep., 2017.

[5] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE internet of things journal*, vol. 3, no. 5, pp. 637–646, 2016.

[6] K. Wang, H. Yin, W. Quan, and G. Min, "Enabling Collaborative Edge Computing for Software Defined Vehicular Networks," *IEEE Network*, vol. 32, no. 5, pp. 112–117, 2018.

[7] "Open Messaging Interface (O-MI), an Open Group Internet of Things (IoT) Standard," http://www.opengroup.org/iot/omi/, [Online]; last accessed March 2020.

[8] "Open Data Format (O-DF), an Open Group Internet of Things (IoT) Standard," http://www.opengroup.org/iot/odf/, [Online]; last accessed March 2020.

[9] M. B. Younes, A. Boukerche, R. De Grande, and H. Xie, "An efficient fault tolerant distributed path recommendation protocol for next generation of vehicular networks," in *2015 IEEE International Conference on Communications (ICC)*. IEEE, 2015, pp. 5783–5788.

[10] C. Huang, K. Wardega, W. Li, and Q. Zhu, "Exploring weakly-hard paradigm for networked systems," in *Proceedings of the Workshop on Design Automation for CPS and IoT*, 2019, pp. 51–59.

[11] J. Lygeros, D. N. Godbole, and M. Broucke, "A fault tolerant control architecture for automated highway systems," *IEEE Transactions on Control Systems Technology*, vol. 8, no. 2, pp. 205–219, 2000.

[12] M. Hiraiwa, H. Asakura, T. Narita, T. Yashiro, H. Shigeno, and K. Okada, "Dynamic communication zone control method on autonomous decentralized based roadside network infrastructure," *IEICE transactions on fundamentals of electronics, communications and computer sciences*, vol. 88, no. 7, pp. 1786–1799, 2005.

[13] J. Almeida, J. Ferreira, A. S. Oliveira, P. Pedreiras, and J. Fonseca, "Enforcing Replica Determinism in the Road Side Units of Fault-Tolerant Vehicular Networks," in *International Conference on Future Intelligent Vehicular Technologies*. Springer, 2016, pp. 3–12.

[14] J. Almeida, J. Ferreira, and A. S. Oliveira, "Fault Tolerant Architecture for Infrastructure based Vehicular Networks," in *Intelligent Transportation Systems*. Springer, 2016, pp. 169–194.

[15] A. Malhi and S. Batra, "Privacy-preserving authentication framework using bloom filter for secure vehicular communications," *International Journal of Information Security*, vol. 15, no. 4, pp. 433–453, 2016.

[16] A. K. Malhi and S. Batra, "Fuzzy-based trust prediction for effective coordination in vehicular ad hoc networks," *International Journal of Communication Systems*, vol. 30, no. 6, p. e3111, 2017.

[17] K. Zhang, Y. Mao, S. Leng, Y. He, and Y. Zhang, "Mobile-Edge Computing for Vehicular Networks: A Promising Network Paradigm with Predictive Off-loading," *IEEE Vehicular Technology Magazine*, vol. 12, no. 2, pp. 36–44, 2017.

[18] J. Liu, J. Wan, B. Zeng, Q. Wang, H. Song, and M. Qiu, "A scalable and quick-response software defined vehicular network assisted by mobile edge computing," *IEEE Communications Magazine*, vol. 55, no. 7, pp. 94–100, 2017.

[19] A. Javed, J. Robert, K. Heljanko, and K. Främling, "IoTEF: A Federated Edge-Cloud Architecture for Fault-Tolerant IoT Applications," *Journal of Grid Computing*, vol. 18, no. 1, pp. 57–80, 2020.

[20] A. Javed, N. Yousefnezhad, J. Robert, K. Heljanko, and K. Främling, "Access Time Improvement Framework for Standardized IoT Gateways," in *IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. IEEE, 2019, pp. 220–226.

[21] J. Robert, S. Kubler, Y. Le Traon, and K. Främling, "O-MI/O-DF standards as interoperability enablers for industrial internet: A performance analysis," in *IECON 2016-42nd Annual Conference of the IEEE Industrial Electronics Society*. IEEE, 2016, pp. 4908–4915.

[22] J. Tao, L. Zhu, X. Wang, J. He, and Y. Liu, "RSU Deployment Scheme with Power Control for Highway Message Propagation in VANETs," in *IEEE Global Communications Conference*. IEEE, 2014, pp. 169–174.