Journal of Network and Computer Applications 231 (2024) 103978

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

Research paper

Fatriot: Fault-tolerant MEC architecture for mission-critical systems using a SmartNIC

Taejune Park [a], Myoungsung You [b], Jinwoo Kim [c,*], Seungsoo Lee [d,*]

[a] Chonnam National University, Republic of Korea
[b] KAIST, Republic of Korea
[c] Kwangwoon University, Republic of Korea
[d] Incheon National University, Republic of Korea

ARTICLE INFO

Keywords:
Mission-critical system
Multi-access Edge Computing (MEC)
Programmable data plane

ABSTRACT

Multi-access edge computing (MEC), deploying cloud infrastructures proximate to end-devices and reducing latency, takes pivotal roles for mission-critical services such as smart grids, self-driving cars, and healthcare. Ensuring fault-tolerance is paramount for mission-critical services, as failures in these services can lead to fatal accidents and blackouts. However, the distributed nature of MEC architectures makes them more susceptible to failures than traditional cloud systems. Existing research in this field has focused on enhancing *robustness* to prevent failures in MEC systems rather than restoring them from failure conditions. To bridge this gap, we introduce *Fatriot*, a SmartNIC-based architecture designed to ensure fault-tolerance in MEC systems. *Fatriot* actively monitors for anomalies on MEC hosts and seamlessly redirects incoming service traffic to backup hosts upon detecting failures. Operating as a stand-alone solution on a SmartNIC, *Fatriot* guarantees the continuous operation of its fault-tolerance mechanism, even during severe errors (e.g., kernel failure) on the MEC host, maintaining uninterrupted service in mission-critical services. Our prototype of *Fatriot*, implemented on the NetFPGA-SUME, demonstrates effective mitigation of various failure scenarios, achieving this with minimal overhead to services (less than 1%).

1. Introduction

The recent advancements in cloud and network technology have enabled a broader range of services, including the Internet of Things (IoT) and cyber–physical systems, which are increasingly integral to daily life. For instance, contemporary power systems, often referred to as ‘smart grids’, utilize a centralized cloud for efficient electricity management (Fang et al., 2011). Additionally, self-driving cars operate autonomously, relying on data exchange with remote cloud servers for navigation (Gerla et al., 2014). In healthcare, devices like pacemakers monitor patient health and utilize cloud-based anomaly detection to identify issues (Pace et al., 2018). Given their significant relevance to daily life, one of the primary concerns for these services is ensuring high availability. For example, failures in smart grid power management can result in extensive blackouts (Asrari et al., 2020), and even minor delays in the braking systems of self-driving cars can lead to catastrophic accidents (Gerla et al., 2014).

These systems, known as mission-critical systems, necessitate high levels of reliability in addition to availability, and thus must be robustly protected and managed to minimize downtime and data loss. Multi-access edge computing (MEC), also referred to as mobile edge computing, is an emerging architecture designed to achieve these objectives. MEC, a form of distributed cloud technology, aims to reduce traffic latency and enhance reliability by positioning a compact cloud infrastructure, known as an *MEC host*, proximate to end devices. This approach significantly reduces the physical distance between a core network node and end devices, diverging from traditional cloud architectures. For these reasons, in recent mission-critical systems, where network quality is a crucial component of service level agreements, MEC has become an indispensable architecture, playing a pivotal role in modern network infrastructures like 5G/6G (Patel et al., 2014; Hu et al., 2015; Chiang and Zhang, 2016; Mao et al., 2017).

Despite its advantages, the primary focus of MEC remains on ensuring network quality for mission-critical services, often overlooking the aspect of reliability under failure conditions. Like traditional cloud systems, MEC is susceptible to outages caused by application errors, virtual machine faults, or hardware failures. A crucial distinction, however, lies in their outage management strategies. Cloud systems can mitigate these issues through service migration or backup systems, leveraging their centralized resources. In contrast, MEC, due to

* Corresponding authors.
E-mail addresses: jinwookim@kw.ac.kr (J. Kim), seungsoo@inu.ac.kr (S. Lee).

https://doi.org/10.1016/j.jnca.2024.103978
Received 27 December 2023; Received in revised form 9 June 2024; Accepted 30 June 2024
Available online 29 July 2024

its distributed nature,[1] encounters challenges in implementing such solutions. Additionally, MEC must contend with a wider spectrum of failure scenarios beyond service outages. For instance, processing delays, seemingly minor but frequent in MEC environments due to traffic surges or resource limitations, can critically impact mission-critical systems. Therefore, issues that might be minor in standard cloud settings emerge as significant reliability challenges in MEC. Addressing these challenges necessitates a more comprehensive and nuanced approach, tailored to the unique properties and requirements of the services in MEC contexts.

To address the issue of reliability in MEC, several previous studies have been conducted (Wang et al., 2023a; Tuli et al., 2022b,a; Grover and Garimella, 2018; Samanta et al., 2021; Sun et al., 2020). However, our review reveals that existing research primarily focuses on enhancing system *robustness* itself to prevent failures, while research on *fault-tolerance*—the capability to maintain reliable services during a failure—is notably lacking. This gap in fault-tolerance research for MEC systems poses challenges in ensuring true reliability for mission-critical services that depend on MEC.

In this paper, we introduce *Fatriot*, a novel architecture meticulously designed to enhance fault-tolerance in MEC systems. Serving as a network interface card (NIC) for MEC hosts, *Fatriot* incorporates comprehensive fault-tolerance management features. It continuously monitors for anomalies such as service or host unavailability and service-specific processing delays. Upon detecting packet processing failures, our system activates a fail-safe mode that seamlessly redirects affected traffic to a backup host, ensuring uninterrupted service continuity. Importantly, this fail-safe mode operates independently of the host system's state and, once configured, functions autonomously, thereby enabling our system to maintain operational integrity. Even in the face of a critical failure rendering the host system unresponsive, *Fatriot* can ensure continuous service, provided that its power is on. We have developed a *Fatriot* prototype using NetFPGA-SUME (NetFPGA, 2024a; Zilberman et al., 2014), and extensive evaluations confirm that *Fatriot* effectively addresses failure scenarios with minimal overhead, allowing services to continue without interruption during failures.

Contributions. In summary, this paper makes the following contributions:

- We present potential failure scenarios within MEC hosts that could compromise the reliability of mission-critical services. This categorization underscores the diverse conditions that can constitute a failure in mission-critical services, extending beyond mere service downtime.
- We propose the design of a new data plane architecture, *Fatriot*, specifically tailored for MEC hosts. This architecture can detect the failures in MEC hosts and provides a fail-safe mode. This mode facilitates the seamless transition of affected traffic flows to backup hosts in the event of a failure, indicating that our system is a stand-alone functionality for effective response to host-wide failures.
- We implement and evaluate a prototype of *Fatriot* using NetFPGA-SUME. Our results show that *Fatriot* introduces minimal overhead compared to a standard NIC, yet it efficiently detects and addresses failure conditions.

2. Background and motivation

In this section, we provide a concise overview of mission-critical systems and Multi-access Edge Computing (MEC). Subsequently, we explore the unique reliability concerns associated with MEC and the challenges encountered in implementing fault-tolerance mechanisms in this context.

[1] Note that in general MEC architecture, MEC hosts are distributed across a network and operate individually with a limited number of devices.

2.1. Mission-critical systems and multi-access edge computing

With the advancement of communication technologies, such as 5G and 6G, numerous systems including the Internet of Things (IoT) and cyber–physical systems, now rely on networked operations to connect services with remote servers (e.g., cloud platforms) or nearby devices. The uninterrupted 24/7 functioning of certain systems is critical, considering the potential risks to life and safety, significant economic impacts, and legal consequence associated with any downtime or malfunctioning. These mission-critical systems are required to adhere to stringent standards encompassing high availability, performance, reliability, and security. These standards are part of ultra-reliable and low-latency communications (URLLC), which necessitate a latency of less than 1 ms and a permissible loss rate not exceeding 1 in 1,000,000.

In this context, maintaining high network quality is of paramount importance. For this, the implementation of Multi-access Edge Computing (MEC), a distributed cloud infrastructure, has become a prominent solution today. As illustrated in Fig. 1, MEC hosts are strategically located near end devices (i.e., end users), enabling them to provide services on behalf of remote cloud servers located far away from the end devices. This proximity significantly reduces the physical distance between end-devices and the services, leading to a marked decrease in network latency. Consequently, this results in improved latency and enhanced reliability of services. Therefore, many operators in the mission-critical systems sector have adopted MEC, leveraging its advantages for network quality enhancement (ETSI, 2024; ISGMEC ETSI, 2019; Hu et al., 2015).

MEC inherits several architectural properties from cloud computing, primarily utilizing virtualization techniques such as network and server virtualization, in compliance with the standards established by the European Telecommunication Standards Institute (ETSI) (ISGMEC ETSI, 2019; Tao et al., 2019; Sabella et al., 2016). According to those properties, services are executed on virtual machines (VMs), and the virtualization platform, commonly referred to as the hypervisor,[2] effectively manages incoming traffic at the host, routing it to the relevant services. However, due to its close association with cloud infrastructure, MEC encounters challenges similar to those in cloud environments, such as potential VM/host failures. These challenges are particularly critical in the context of MEC's role in supporting mission-critical systems, where reliability and uninterrupted service are imperative.

2.2. Reliability issues and failure conditions in MEC

The reliability of mission-critical systems is paramount, as failures within these systems can lead to significant risks. For example, self-driving cars rely heavily on the exchange of critical data with nearby vehicles and control servers for accurately assessing car speeds, monitoring traffic and road conditions, and receiving essential control signals. Considering the critical roles played by control servers in self-driving cars, a temporary disruption resulting in the loss of control signals could lead to catastrophic failures and even fatal accidents (Papadimitratos et al., 2009; Lu et al., 2014; Uhlemann, 2015; Khan et al., 2019). Therefore, even though uninterrupted service maintenance is essential for MEC hosts, their architectural design renders them susceptible to various failure risks, which can potentially compromise service reliability.

To explore the reliability in MEC systemically, we first categorize various failure conditions within an MEC host into two main types: *(i) component faults* and *(ii) processing delays*. Subsequently, we examine how these failures impact the reliability of mission-critical systems, as detailed below (Perez-Botero et al., 2013; Coppolino et al., 2017) and described in Fig. 2.

[2] Note that although OS-level virtualization has recently become more popular, the ETSI standard advocates for hypervisor-based server virtualization.

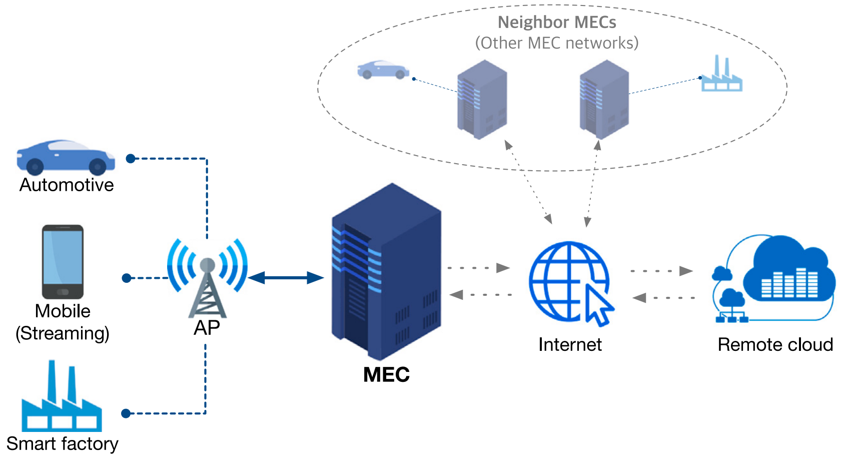


Fig. 1. An illustration of the MEC-deployed network. The remote cloud transfers mission-critical services, which are highly dependent on network performance, to an MEC host situated in close proximity to end-devices.

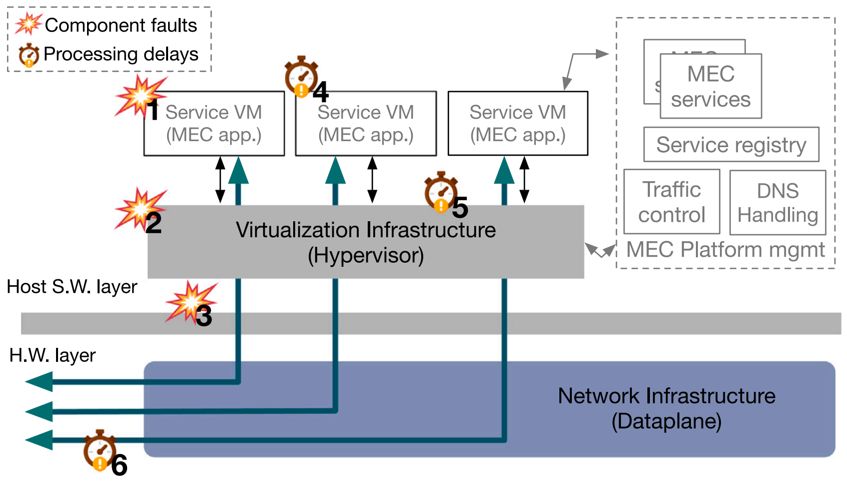


Fig. 2. Reliability issues on the standard MEC architecture (ISGMEC ETSI, 2019; Tao et al., 2019; Sabella et al., 2016) for mission-critical systems, where the component faults are denoted by #1–3 and processing delays are #4–6.

(i) Component faults: Operational errors in virtual machines (VMs), which run networked service applications, can arise from multiple factors (#1). These include application bugs, conflicts with other applications, operating system issues, or improper configurations. Such errors can result in mission-critical systems malfunctioning or unexpectedly halting. Hypervisors are prone to fatal errors due to various unexpected factors, such as resource limitations and misconfigurations (#2). These errors can disrupt all the hosted VMs, adversely affecting the mission-critical systems running on them. In a similar vein, MEC hosts themselves, which operate the hypervisors, are also susceptible to disruptions. Failures in the host device due to hardware malfunctions, power issues, network outages, or similar factors can lead to significant disruptions in mission-critical systems as well (#3).

(ii) Processing delays: MEC hosts, unlike cloud servers, may face significant request volumes from certain regions or locations, leading to server overload and processing delays. Such congestion can slow down the response times of mission-critical systems, leading to user-perceived service lags. (He et al., 2016; Lee et al., 2019; Park et al., 2022). In this context, inadequate resources on an MEC host can slow down the processing of applications or VMs (#4). Additionally, inefficient application design or suboptimal data processing methods can cause processing delays, potentially breaching the low-latency requirements of mission-critical systems. The complexity introduced by virtualization technology, and the interactions between VMs and the underlying virtualization infrastructure, can result in processing delays as well (#5). These issues are magnified under high service loads or when resources are shared with other VMs, which can degrade service quality and compromise system consistency and reliability (Zhou et al., 2012; Allan et al., 2016; Anwar et al., 2017). In addition, network faults are particularly critical in MEC, as it relies on network-based services (#6). Communication between applications and end-devices can be hindered by problems such as incorrect network configurations or insufficient bandwidth. Additionally, this congestion can cause packet loss, resulting in data inconsistencies.

The presence of those factors presents significant challenges to the reliability of mission-critical systems operating within the MEC environment. However, the existing architectural design of MEC has not adequately addressed these conditions, nor has it elaborately discussed strategies for the detection and recovery from such failures. This oversight leads to a noticeable research gap in the realm of MEC reliability, particularly concerning mission-critical systems. Addressing this gap is crucial, as it is imperative to ensure continuous operation of these systems, even under fault conditions.

2.3. Challenges in achieving fault-tolerance on MEC

Fault-tolerance is a crucial design principle for mission-critical systems to ensure uninterrupted operation and resilience in the face of failures. Considering the role of MEC in conjunction with mission-critical systems, implementing efficient fault-tolerance mechanisms on MEC is essential for enhancing the reliability and availability of mission-critical systems (Tao et al., 2019). In what follows, we discuss lacking of fault-tolerance mechanisms in MEC (Bala and Chana, 2012; Cheraghlou et al., 2016; Koren and Krishna, 2020; Sorin, 2022) and analyze their limitations encountered when integrating these mechanisms into current MEC environments.

(i) Redundancy: MEC implementation should incorporate server redundancy to facilitate service transfer to other servers and the utilization of load-balancing techniques for even traffic distribution. This approach enhances the availability of MEC hosts in the event of failures and improves overall system reliability by preventing server overloading. However, scaling out for redundancy presents challenges in migrating services across distributed MEC hosts, unlike common cloud servers located within the same data center (Sabharwal et al., 2013; Khan et al., 2019; Tao et al., 2019; Wang et al., 2023b).

(ii) Monitoring and detection: The foundation of fault-tolerance lies in system monitoring, where monitored data is utilized to diagnose and address failures appropriately. Real-time monitoring is ideally necessary for quick diagnosis and response. However, conducting real-time monitoring for MEC hosts is challenging due to their limited resources compared to cloud servers. As a result, monitoring can significantly impact the performance of an MEC host, leading to a reduction in available system resources for mission-critical systems. Paradoxically, system monitoring can potentially contribute to the occurrence of fault conditions in an MEC host rather than preventing them (Popiolek and Mendizabal, 2012; Suneja et al., 2015; Rameshan, 2016; Popiolek et al., 2021).

(iii) Gradual degradation and automatic recovery: As described above, even a brief interruption of services in the MEC host caused by failures such as network disconnection, kernel crash, or unexpected system error, though typically short in duration, can significantly impact the entire mission-critical systems. Therefore, fault-tolerant mission-critical systems should continue functioning even in the presence of failures by allowing certain performance or functionality degradation instead of immediately halting entire services. Moreover, these systems should be designed to autonomously recover from the failures and restore their normal operations. In such cases, simply redundancy can offer a solution to ensure service continuity, but it does raise cost concerns, as mentioned earlier.

In summary, attaining fault-tolerance for MEC with mission-critical systems poses several challenges: (i) redundancy may introduce additional costs, (ii) monitoring anomalies can impact the performance of MEC hosts, and (iii) MEC hosts are not inherently designed to seamlessly handle the failures.

2.4. Research goal

To address the challenges previously outlined, we aim to develop a fault-tolerance system that is specifically tailored for MEC environments supporting mission-critical systems. We recognize that the primary challenges stem from the resource-intensive nature of these systems and their tight integration with the MEC host in current architectures. In response, we propose a novel hardware-assisted architecture that utilizes a SmartNIC. This SmartNIC operates within an isolated execution environment and switches to a standalone mode during failures in the MEC host. This design ensures that the fault-tolerance system remains independent of the MEC host's primary operations, thereby conserving host resources and significantly enhancing the system's capability to manage faults and sustain uninterrupted functionality in mission-critical scenarios.

3. System design

In this section, we first explore the design considerations for our system. Following this, we introduce *Fatriot*, a novel fault-tolerance system for MEC environments, developed using a SmartNIC.

3.1. Design considerations

Reliable fault detection: Our primary objective is to achieve effective and reliable fault detection on an MEC host even under fatal errors like a kernel panic. This approach is crucial to prevent any adverse effects on mission-critical systems. Furthermore, it is imperative that the fault-tolerance system itself remains robust against host interruptions, thereby ensuring continuous operation and bolstering the reliability of mission-critical applications. Achieving these objectives requires the system should operate independently of the MEC host's state.

Real-time performance monitoring: As previously discussed, performance degradation in mission-critical systems can significantly compromise the reliability of their continuous operations. Therefore, it is essential not only to identify system failures but also to consider performance anomalies when analyzing the state of the MEC system. Furthermore, to proactively address these issues, the monitoring system tasked with detecting anomalies must operate in real-time. This real-time operation is critical to enable prompt responses and actions, ensuring the continuous and reliable functioning of mission-critical services.

Fail-safe operation: The fundamental principle of a fault-tolerant system is to ensure continuous operation of mission-critical services, even amidst failures. For instance, when a malfunction occurs in an MEC host, an alternative server should be capable of seamlessly taking over to handle requests, thereby guaranteeing uninterrupted functionality of mission-critical services. Consequently, the system should incorporate a fail-safe mechanism that ensures a reliable and continuous service experience, especially for applications classified as mission-critical.

3.2. Fatriot overview

Fig. 3 provides an overview of the architecture of *Fatriot*. It is conceptualized as a Network Interface Card (NIC) tailored for an MEC host system, encompassing all essential functionalities for managing fault-tolerance. This approach empowers *Fatriot* to function autonomously, detached from the state of the host device—except for the policy configuration, elaborated upon later—as long as power is sustained. As a result, *Fatriot* can continually monitor the host for any anomalies, promptly responding to emerging faults in real-time, and thereby ensuring the uninterrupted operation of mission-critical services.

System components: *Fatriot* consists of four fundamental components: (i) the *heartbeat manager*, which generates heartbeat signals targeted at each service and the host platform (i.e., OS or hypervisor) to ascertain potential faults, (ii) the *traffic classifier*, responsible for identifying the traffic associated with mission-critical services to facilitate fail-safe operations, (iii) the *queue*, which carries out the task of forwarding or mirroring packets to the upper layer or another module, and (iv) the *fail-safe handler*, responsible for triggering fail-safe operations. This module reroutes mission-critical traffic to a neighboring device upon the detection of host or application failures, thereby ensuring uninterrupted operation despite the identified issues.

Overall workflow: The operational sequence of *Fatriot* unfolds as follows (refer to Fig. 3). ①Incoming traffic of an MEC host is subjected to initial processing through the traffic classifier within the *Fatriot* NIC to classify whether it is a packet for a mission-critical service. ②The packets are then enqueued with their classification result. ③Non mission-critical service packets are forwarded directly from the queue to the host layer without additional processing. ④In contrast, packets deemed mission-critical are not only forwarded but also mirrored to the fail-safe handler to facilitate the preparation of a fail-safe operation. ⑤Concurrently, the heartbeat manager dispatches heartbeat packets to applications or VMs by inserting them to the queue to assess their availability. ⑥Should any fault conditions (as mentioned in Section 2.2) be detected on the host, the heartbeat manager signals the fail-safe handler to redirect the packets to neighboring devices, wherein mission-critical

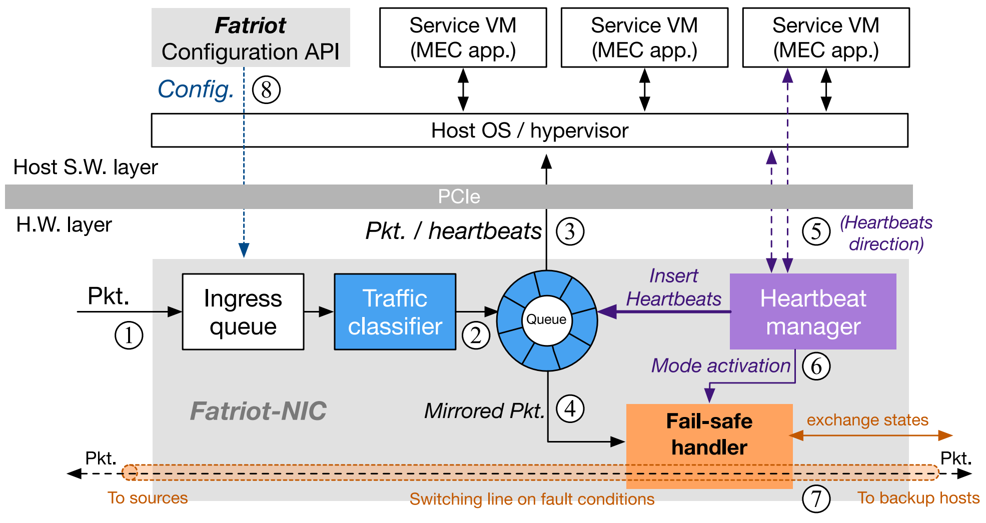


Fig. 3. The overall architecture of *Fatriot* system.

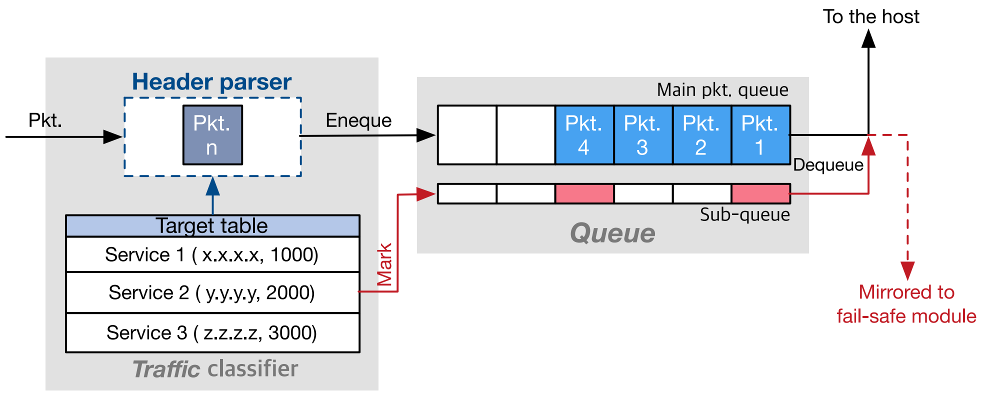


Fig. 4. The design and workflow of the traffic classifier and queue.

services can be attended to. ⑦In such scenarios, *Fatriot* seamlessly operates as a network switch, transparently facilitating communication with another MEC host. ⑧Finally, the *Fatriot* control plane in the host software layer configures the *Fatriot*-NIC, enabling the identification of mission-critical traffic and the early detection of faults.

3.3. Traffic classifier and queues

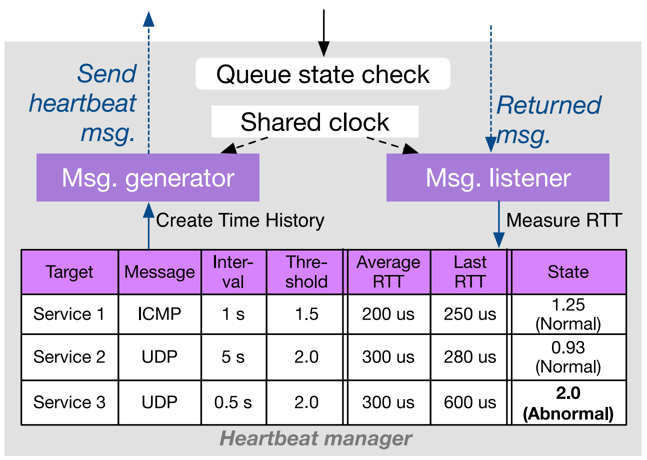
While a primary requirement of *Fatriot* is to guarantee fail-safe operations, even in instances of host failures, a significant challenge arises from the limitations inherent in traditional NIC architectures in effectively accommodating these operations. In the traditional NIC architecture, incoming packets are typically routed to the host. However, in scenarios where the host experiences a fault, there emerges a need to redirect enqueued packets to an alternative host. Unfortunately, the standard NICs lack the functionality required to support such redirection. To address this limitation, we propose the design of an advanced NIC queue, complemented by a specialized traffic classifier.

This combination is specifically engineered to enable the packet forwarding mechanism we envision, as illustrated in Fig. 4. For instance, when a packet is incoming into *Fatriot*, before getting into a queue to be forwarded to a host, the packet firstly goes through the traffic classifier to determine whether it is intended for a mission-critical service. The traffic classifier contains the target table, which is a list of service signatures (e.g., IP and port addresses of a destination service) and its addresses designated for mission-critical systems, and the header parser to identify whether the packet is a mission-critical one or not based on the table. The identification of services per packet is carried out using the IP address of a service VM and its application's port number. Then, the packet is enqueued.

The queue consists of two 1:1 corresponding queues: one for forwarding packets normally (i.e., main packet queue) and the other for identifying whether the packet is meant for a mission-critical service (i.e., sub-queue). When the packet that is identified as a mission-critical one is enqueued, the corresponding sub-queue is also enqueued with a flag data indicating that the packets entering the queue now are for a mission-critical service and a service ID. When the packet in the main packet queue is dequeued towards the host layer, the sub-queue also dequeues at the same time. If the current packet is identified as belonging to a mission-critical service, it is mirrored to the fail-safe handler. Consequently, regardless of whether the host is behaving in a faulty manner, the packets for mission-critical systems are always ready to be forwarded to their neighbor via the fail-safe handler. Conversely, if the host is functioning normally, the packet will be served through the host as usual.

3.4. Heartbeat manager

Fatriot monitors the operational states of host platforms, including the operating system and hypervisor, as well as service applications. This monitoring is achieved by measuring the round-trip time (RTT) of heartbeat messages sent to these components. If the RTT exceeds a predetermined threshold, expressed as a ratio to the average RTT observed under normal operating conditions, *Fatriot* identifies this deviation as indicative of an operational abnormality in the target component.



Target	Message	Inter-val	Thre-shold	Average RTT	Last RTT	State
Service 1	ICMP	1 s	1.5	200 us	250 us	1.25 (Normal)
Service 2	UDP	5 s	2.0	300 us	280 us	0.93 (Normal)
Service 3	UDP	0.5 s	2.0	300 us	600 us	**2.0 (Abnormal)**

Fig. 5. The design and workflow of the heartbeat manager.

Fig. 5 illustrates the design of the heartbeat manager. Its structure is fundamentally based on the previous research, specifically the Formullar project (Park et al., 2021), which is an FPGA-based network testing tool. Formullar is known for its ability to measure the generation and reception of messages using a shared timer and for implementing various traffic models in a programmable way. While its original design was focused on the precise measurement of network performance between hosts, we redesign its functionality to measure the performance and availability of intra-host components, such as services and VMs. This modification enables the generation of heartbeat messages and monitoring of the host system's running state based on RTT measurements with hardware-level precision.

The heartbeat manager begins by configuring entries for heartbeat messages, its interval and threshold ratio for a target (i.e., host or service instances), $\langle target, message, interval, threshold\rangle$ and manages the entries in a table. The heartbeat messages can take ICMP or UDP with predefined messages. These can be configured by a system administrator through control messages to check against application behavior, such as intentionally sending a request message that triggers the return of an error message. The *interval* can be configured as small if the administrator is concerned about any failures; thus, *Fatriot* sends heartbeat packets frequently, increasing the heartbeat transmission rate. However, it is important to note that injecting many heartbeat packets does not impose performance overhead. After the configuration, the heartbeat manager measures the average RTT for the target (e.g., the host or services on it) using the designated heartbeat message under normal conditions. After that, it transmits the heartbeat message to the target at regular specified intervals, measures its RTT, and computes the $RTT_{last}/RTT_{average}$ ratio. If this ratio surpasses the threshold, it signifies an abnormal condition, triggering the activation of the fail-safe mode for the target. Consequently, the timeout duration for the heartbeat is inherently established as the product of the average RTT and the threshold.

Fig. 5 contains an example of the heartbeat manager with its configuration table. The heartbeat message for Service 1 in the host system is set to ICMP, operates every 1 s, and the threshold is set to 1.5. Its average RTT is measured as 200 μs, the last RTT is measured 250 μs, and the $(last\ RTT)/(average\ RTT)$ ratio is 1.25, lower than the threshold, i.e., its state is normal. In the case of Service 2, it works with a user-defined message, and its interval is set to be relatively long at 5 s as if Service 2 is assumed that it does not need to be checked often. Its $(last\ RTT)/(average\ RTT)$ ratio is 0.93, i.e., it is also normal. Otherwise, in Service 3, its interval 0.5 s as if it should be check frequently, but the last RTT is measured as 600 μs as the heartbeat has timed out by exceeding the threshold. Therefore, it is considered as an abnormal state so that the fail-safe mode for Service 3 is activated.

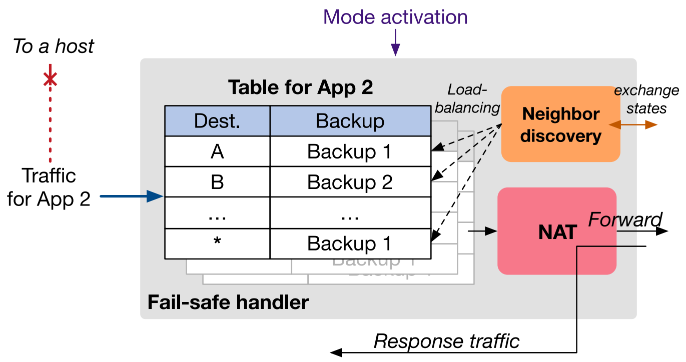


Fig. 6. The design and workflow of the fail-safe handler.

Whereas, if the last RTT returns to within the threshold again, the fail-safe mode is deactivated, and processing is performed again on the MEC host. If an entire host is in an abnormal state (e.g., kernel failures), the fail-safe mode is automatically activated for all registered services because *Fatriot* remains functional independently of the availability of host software stacks, requiring only power to function. Leveraging this advantage, *Fatriot* incorporates a hybrid structure; its fail-safe mode transitions from a NIC to a network switch upon detecting host system or application failures via the heartbeat manager. Consequently, even if the service processing becomes unavailable on the host, critical traffic is rerouted to pre-configured neighboring hosts to maintain service continuity.

3.5. Fail-safe handler

The fail-safe handler allows *Fatriot* to operate like a network switch. When a host (or services running on it) is unavailable, the fail-safe handler forwards incoming traffic to pre-designated backup hosts or servers so that the services can continue there as illustrated in Fig. 6.

The fail-safe handler manages a service table that specifies backup hosts against certain services as well as a default backup server (i.e., the symbol $*$). Upon detection of a packet destined for an unavailable service (App 2 of Fig. 6), the traffic classifier within *Fatriot* forwards the packet to the fail-safe handler instead of forwarding it to the host. The fail-safe handler then retrieves the service table, using the packet's destination IP and port as a key, to identify an appropriate backup host. It proceeds to modify the packet's destination IP and port (DNAT), ensuring the packet's seamless reception at the backup host. Concurrently, the source IP and port are modified to predefined values so that response packets can be routed back to the current *Fatriot* instance. After address translation, the fail-safe handler recalculates the necessary checksums to maintain packet integrity and forwards the packet to the designated backup host. When a response packet is received from the backup host, the NAT-modified address is restored to its original state (SNAT), and the packet is then relayed back to the original source. This process enables the fail-safe handler to seamlessly relay traffic between the backup host and end-users for services that are temporarily unavailable, thus maintaining the continuity and reliability of services.

The neighbor discovery module autonomously manages the list of available backup hosts by communicating other hosts, which also includes those utilizing *Fatriot*. After an administrator configures the backup hosts, the discovery module enables the exchange of available service capacities and states among these backup hosts based on the service signatures. Subsequently, it ensures that the available backup hosts for a specific service are evenly distributed and utilized across multiple hosts, preventing the concentration of load on any single backup server.

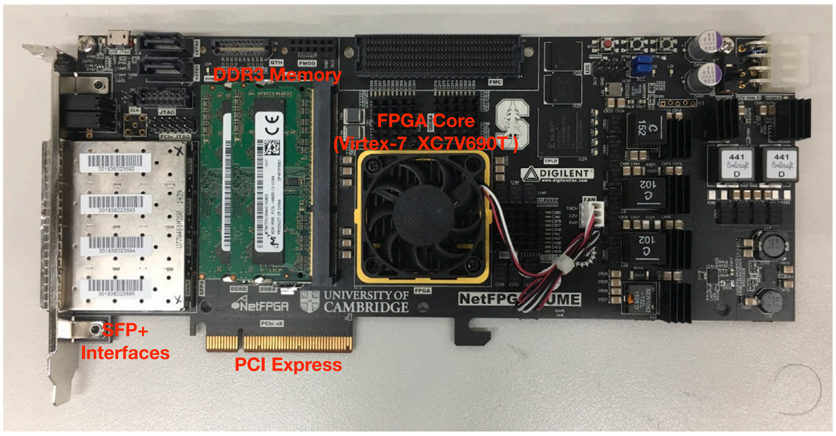


Fig. 7. NetFPGA-SUME.

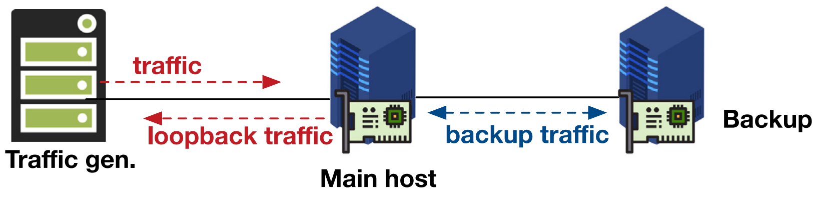


Fig. 8. Evaluation environment.

3.6. Control interface

Fatriot serves as a SmartNIC that interfaces with a host machine through a PCIe connection, guaranteeing service reliability for edge systems. To enable communication with the host machine for the configuration, *Fatriot* utilizes its device driver within the host's software layer and offers a range of APIs for management purposes (the configiruation API depicted in Fig. 3). Through this configuration API, the host or the network administrator can configure necessary tables essential for *Fatriot*'s operation.

4. Implementation

We implement a full prototype of *Fatriot* utilizing NetFPGA-SUME, an FPGA-based PCI Express board featuring the Xilinx Virtex-7 XC7V690T FPGA and equipped with four SFP+ 10 Gbps interfaces (NetFPGA, 2024a; Zilberman et al., 2014) (see Fig. 7). It is installed on a host machine powered by a Xeon Gold 5520 processor and 64 GB of RAM, connected via a PCI Express interface. The host environment includes KVM (RedHat, 2024) and Open vSwitch (Open vSwitch, 2024; Pfaff et al., 2015) to implement the standard Multi-access Edge Computing (MEC) architecture as defined by the European Telecommunications Standards Institute (ETSI) (ISGMEC ETSI, 2019).

The overall development is based on the Reference NIC project offered by the official NetFPGA-SUME project repository (NetFPGA, 2024b). The host software interface is implemented using the reference driver provided by the official repository as well. Also, the host user interface (API) is implemented via netlink to the NetFPGA-SUME reference device driver.

5. Evaluation

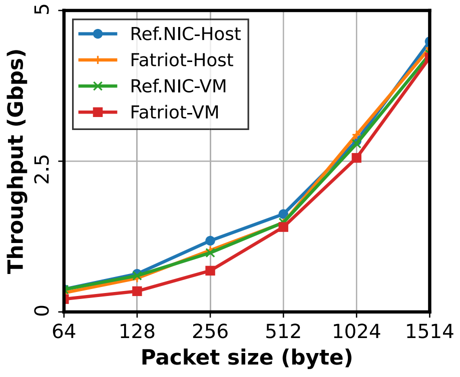
Experimental environment. The environment consists of a main MEC host, a backup host, and a traffic generator as illustrated in Fig. 8. Each host machine has an Intel Xeon Gold 5520, 64 GB of RAM, and NetFPGA-SUME for the *Fatriot* prototype. These hosts run KVM-based virtual machines (VMs), which act as mission-critical edge services, and utilize Open vSwitch (Open vSwitch, 2024; Pfaff et al., 2015) to provide network connectivity for VMs. All VMs perform a loop-back function to respond (i.e., echo-ing) with the received payload under network contentions.

To validate the feasibility of *Fatriot*, we assess its network performance and compare it against the reference NIC of NetFPGA-SUME. This reference NIC, a simple FIFO-based 10GbE network interface card, is commonly utilized as a baseline in numerous NetFPGA-SUME projects. Additionally, we evaluate the effectiveness of *Fatriot*'s fail-safe mode by benchmarking its response time in detecting and addressing failure conditions on MEC hosts. Furthermore, we analyze the network performance of backup hosts when the fail-safe operation is activated. The test traffic is generated using the FPGA-based packet generator, Formullar (Park et al., 2021). It transmits a specified amount of packets at a certain traffic rate, allowing us to measure round-trip times with a precision of 6.25 ns.

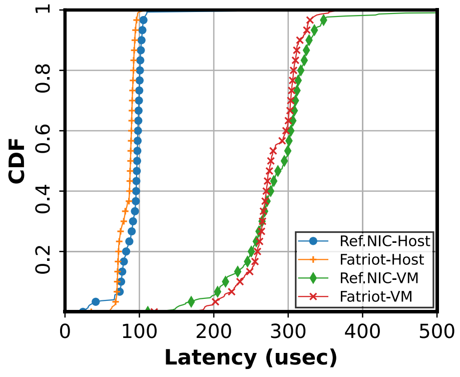
5.1. Performance overhead

To assess the performance impact of *Fatriot*'s design, we initially compare it to the reference NIC of NetFPGA-SUME. We subject the main host to burst traffic generated by Formullar and measure both throughput and latency. In this evaluation, the main host forwards the received traffic to its VM, which returns the received traffic without processing. Fig. 9 depicts the results of this evaluation; `Ref.NIC-Host/VM` denotes the communication performance metrics observed between the traffic generator and the main host (or its VM) when using the reference NIC. Conversely, `Fatriot-Host/VM` represents the communication performance metrics when *Fatriot* is employed instead of the reference NIC.

The throughput and latency of *Fatriot* closely resemble those of the reference NIC. As shown in Fig. 9(b), an increase of approximately 200 microseconds in latency is observed when traversing the VM, but this overhead is equally evident on both the reference NIC and *Fatriot*, attributable to the VM's network stack (e.g., Open vSwitch). This result aligns with expectations, given that the operations of *Fatriot* occur entirely at the hardware level with no direct impact on host

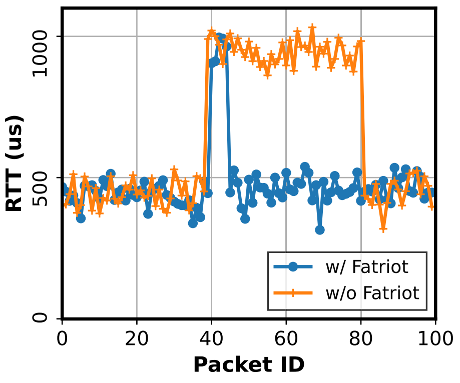


(a) Throughput

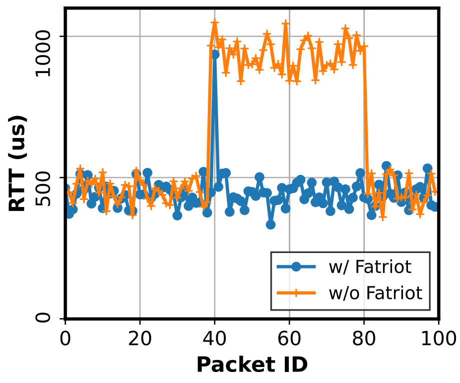


(b) Latency

Fig. 9. The results of the performance comparison between Reference NIC and *Fatriot*.



(a) Fail-safe interval 0.5 seconds



(b) Fail-safe interval 0.1 seconds

Fig. 10. The results of mitigating processing delay failure.

performance. Consequently, the design of *Fatriot* does not introduce extra overhead to the host systems, making it a promising solution for enhancing availability in MEC environments.

Please note that although our platform, NetFPGA-SUME, is designed to support 10 Gbps of bandwidth, our achieved result is only half of that capacity. This degradation is attributed to the bottleneck in the official device driver responsible for transmitting traffic between the NIC and the host kernel (or vice versa). The official driver provided by NetFPGA organization lacks optimal implementation, contributing to this bottleneck. We believe that this overhead can be mitigated with the implementation of an improved device driver. Nonetheless, our primary objective revolves around enhancing the availability of MEC, rather than focusing on performance improvement. Hence, since we verify that there is no performance degradation in *Fatriot* when compared to the reference NIC, we maintain efforts to improve its performance as well as the reference device driver in future works.

5.2. Effectiveness of fail-safe mode

To evaluate the effectiveness of *Fatriot* in handling failure conditions, we perform a test by deliberately triggering a failure in the main host while initiating a traffic flow. In this experiment, traffic is generated at a rate of 10 packets per second, directed towards a VM on the main host. The VM then returns the received packets back to the traffic generator.

Processing delay: This scenario considers potential failures arising from factors such as suboptimal host performance and traffic congestion, resulting in delayed response times. To simulate this scenario, when the service VM responds to a request, we configured the VM to intentionally introduce a time delay in its replies starting from the 40th packet. Then, assuming system recovery, this simulated delay is removed beginning with the 80th packet. Subsequently, we assess changes in round-trip time (RTT) using the traffic generator under two conditions: (1) with fail-safe mode enabled (i.e., with *Fatriot*) and (2) with fail-safe mode disabled (i.e., without *Fatriot*). Fail-safe detection in *Fatriot* is configured to trigger when latency increases by over 50% within a 0.5-second cycle. In such instances, traffic is rerouted to the backup host for continued service.

The evaluation results are illustrated in Fig. 10(a), with the orange line representing the scenario involving *Fatriot* and the blue line depicting the situation without *Fatriot*. In the absence of *Fatriot*, the RTT experiences two increments commencing from the 40th packet, persisting until the failure restoration at the 80th packet. Conversely, in the presence of *Fatriot*, an abrupt RTT surge is promptly detected, prompting the activation of fail-safe mode and the rerouting of traffic to the backup host. As a result, RTT rapidly diminishes by the 50th packet. It is worth noting that the implementation of the fail-safe mode may introduce a slight RTT increase owing to the extended distance to the backup host. Nevertheless, this delay typically falls within the range of a few microseconds, which is considered negligible in a real-world environment.

The experiment depicted in Fig. 10(a) necessitates a minimum of 0.5 s for the activation of the fail-safe mode. This delay is a result of *Fatriot* measuring RTTs within intervals specified by the administrator. Initially, this delay may appear unsuitable for mission-critical services. However, to address this concern, the interval time can be configured to a shorter duration (e.g., 0.1 s), as exemplified in Fig. 10(b). By doing

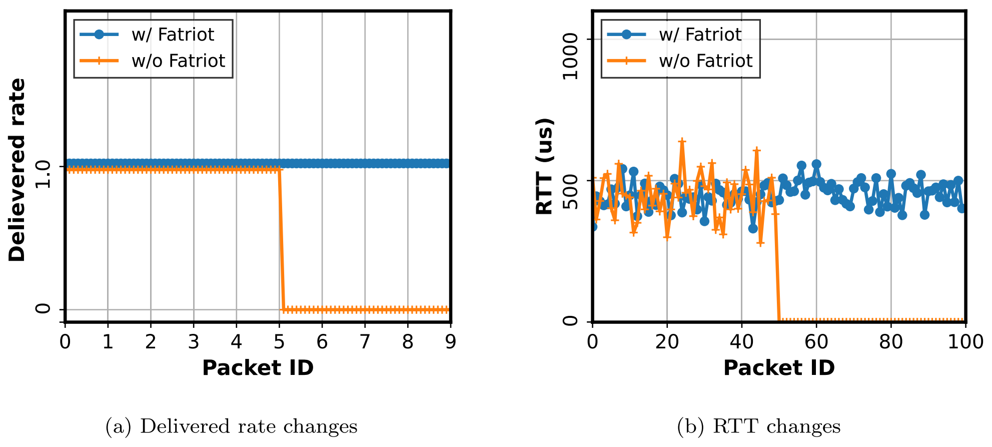


Fig. 11. The results of mitigating host down failure.

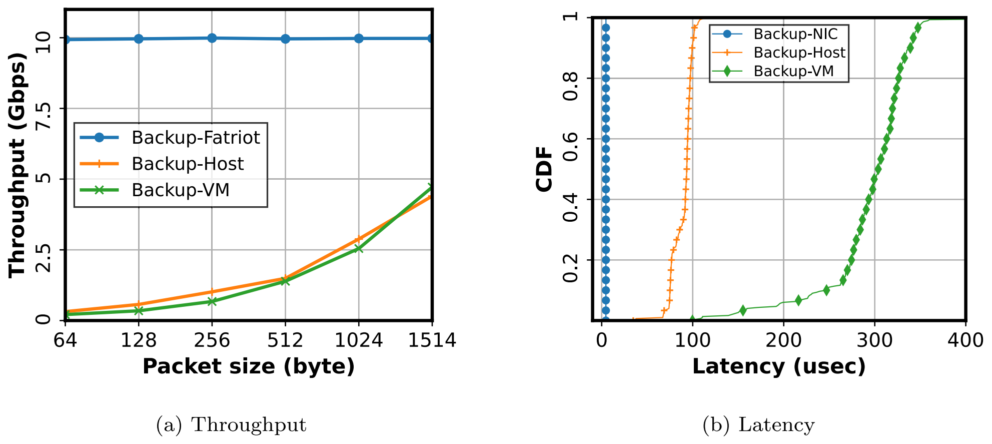


Fig. 12. The results of performance overheads via the backup host's *Fatriot* NIC only, host system, and VM.

so, *Fatriot* can transition into fail-safe mode more rapidly, mitigating any potential delays in service processing.

Host down failure: This scenario reflects a situation where the host system (e.g., OS or hypervisor) undergoes a crash, rendering it incapable of delivering any services. To simulate this scenario, we intentionally induce a system crash on the main host at the 50th packet while continuing to send traffic to it. Subsequently, we assess both the delivered rate of traffic and changes in RTT from the traffic generator under two conditions: with fail-safe mode enabled and with fail-safe mode disabled.

The results are presented in Fig. 11. In the scenarios where the fail-safe mode is disengaged (the orange line), packet delivery is suspended, resulting in both the delivered rate and RTT recording zero. Conversely, with the fail-safe mode engaged (the blue line), the delivered rate and RTT exhibit stability as the service seamlessly transitions to the backup host, ensuring uninterrupted service provision. It is acknowledged that a marginal RTT increase is discernible; however, it is deemed inconsequential within the broader context and remains imperceptible in the graphical representation. Taken together, we can see that *Fatriot* is able to adequately detect anomalies in a system and handle them in a timely manner and with low overhead.

5.3. Network performance

Finally, to see if relaying and rerouting traffic to the backup host via the fail-safe mode is practical, we measure the network throughput and latency from the traffic generator to the backup host under the fail-safe mode.

In Fig. 12, the blue line represents the result of measuring network performance to the backup host's *Fatriot* via the main host's *Fatriot* (i.e., traffic generator → main host's *Fatriot* → backup host's *Fatriot*). It is evident that *Fatriot* achieves line-rate performance, showcasing minimal observable overhead. Note that the throughput degradation attributed to the device driver does not affect this evaluation result, as the traffic is relayed at the hardware layer without going through the host layer. Consequently, we observe *Fatriot* achieving nearly identical performance as earlier measurements conducted on the main host, whether the traffic is directed to a host or VM of the orange and green lines (i.e., traffic generator → main host's *Fatriot* → backup host's *Fatriot* → backup host's OS or VM). In fact, there is a slight increase in latency attributable to increased travel and processing within *Fatriot*. However, this increment is on the order of a few hundred nanoseconds, which remains negligible at the overall scale. In summary, the overhead of the fail-safe module is almost non-existent, and with enough backup hosts, MEC should be able to support a reliable service in failure conditions.

6. Limitations and discussion

Dependency on admin configuration: *Fatriot* now relies on manual configuration by administrators in many aspects, including detection policies and backup host configurations. Hence, failure detection may not work properly if not configured correctly, and requires constant updates from administrators as services are added or changed, and as network conditions change on a permanent basis. We believe these issues can be easily addressed by combining Software-Defined Networking (SDN), introducing a global management scheme for the network, and deploying a number of known intelligent/automated detection and network environment configuration techniques. However, the intelligent behavior of anomaly detection and fail-safe mode activation is beyond the scope of this paper, which is primarily aimed at hardware design, and will be addressed in future work.

Reliability and deployment issue: *Fatriot* is an extension of the MEC NIC, and there is a possibility of functional failure in this itself. However, the primary functionality of *Fatriot* involves 'extending' the operations of a typical NIC with various features, including fail-safe. The functionalities up to the traffic classifier and queues are almost identical to those of a common NIC. In other words, the main features of *Fatriot* (e.g., fail-safe) and general packet processing are structured separately, so even if there is a malfunction in *Fatriot*, the NIC operations remain functional. Therefore, the impact on the system is minimal. However, to fully utilize the functionality of *Fatriot*, it is necessary to exchange state information between neighbor MECs considering backup hosts, which implies that *Fatriot* needs to be installed on other neighbors, increasing overall operational costs. While this is a direct disadvantage of *Fatriot*, it can be mitigated by configuring the Fail-safe handler and manually addressing responses from the backup host.

Stateful connection with backup servers: The current design of *Fatriot* does not consider stateful connections with backup servers when a flow/service turns into the fail-safe mode. This omission is due to the complexities involved in maintaining a session with an external device, as it necessitates duplicating all traffic and synchronizing all states each time to sustain communication, resulting in significant overhead. In addition, a session migration might be a viable solution only for a L4 layer but if service packets contain contextual information (i.e., L7), requiring advanced technology that could be a huge research topic for this. The implementation of such aspects is beyond the scope of this paper's topic, while the major scope of this paper is to implement a fault-tolerant mechanism in a MEC. However, we believe that by combining *Fatriot* with several well-known stateful migration strategies (Junior et al., 2020; Horii et al., 2018; Zandi et al., 2023) and incorporating precise protocol information for each service, it is possible to achieve a sufficiently stateful fault-tolerant system on MEC.

7. Related work

As reliability becomes critical in MEC systems, numerous studies have emerged to address this challenge. However, most focus on preventing MEC system failures through early-stage detection or optimizing resource utilization, rather than on robust mitigation strategies. This section explores these studies, highlighting how our approach, which leverages a SmartNIC for real-time failure detection and redirection, differs significantly. Note that existing studies and *Fatriot* are based on the standard MEC architecture proposed by ETSI (European Telecommunication Standards Institute) (ISGMEC ETSI, 2019), where a MEC host runs multiple virtualized applications.

Predicting failures in MEC systems: Most previous studies (Wang et al., 2023a; Tuli et al., 2022b,a) in this domain aim to avoid fault recovery by predicting failures in advance and taking appropriate actions in response. B-Detection (Wang et al., 2023a) harnesses the power of deep learning, utilizing a prediction model that identifies runtime reliability anomalies in services on MEC systems, leveraging the historical data distribution of these services. Another solution, PreGan (Tuli et al., 2022b), employs Generative Adversarial Networks (GANs) for early failure prediction, facilitating preemptive migrations for fault tolerance. Additionally, DRAGON (Tuli et al., 2022a) adopts a memory-efficient deep learning model, generative optimization networks (GON), enhancing the agility of failure detection mechanisms. While invaluable for early detection, these studies do not provide immediate fault response mechanisms, which is where our system's real-time detection and redirection capabilities using SmartNICs offer a novel contribution.

Task deployment strategy for MEC systems: Another line of research (Grover and Garimella, 2018; Samanta et al., 2021; Sun et al., 2020) explores enhancing MEC system reliability through task replication and agent-based fault monitoring. Sun et al. (2020) propose a QoS-aware task deployment model for dynamic task rearrangement, and Grover and Garimella (2018) introduce software agents for failure monitoring and task redirection. However, these methods are limited in addressing failures in real-time and depend on the availability of the host system. Our approach, in contrast, leverages the inherent capabilities of a SmartNIC to detect and respond to failures immediately, providing a robust and more effective fault tolerance solution for MEC systems regardless of the availability of the host system.

Hardware extended MEC architecture: Advances in programmable data plane technologies have prompted researchers (Ricart-Sanchez et al., 2019; Park et al., 2022; Paolucci et al., 2021; Mai et al., 2020) to enhance MEC system performance and reliability by utilizing SmartNICs and programmable switches. MECaNIC (Park et al., 2022), for example, offloads existing software-based network stacks of a MEC host to a SmartNIC for improved packet processing capabilities, thereby ensuring network-level reliability even under resource contention scenarios. Similarly, Mai et al. (2020) utilize in-network computing for task acceleration for services running on MEC systems. While these studies effectively utilize hardware for performance and reliability, they do not focus on real-time failure detection and response. Our system fills this gap by not only utilizing a SmartNIC for performance enhancement but also for real-time failure detection and mitigation, marking a significant advancement in MEC system reliability.

8. Conclusion

MEC stands as an essential architecture for ensuring the reliability of mission-critical services over networks so that ensuring service availability is paramount in MEC, prompting the proposal of numerous solutions to address this critical need. However, previous research on MEC reliability has been focused on making MECs more robust against failure but placing lesser emphasis on strategies to maintain service continuity in the event of an actual MEC failure. Therefore, in this paper, we have organized a more rigorous definition of the failure conditions that can occur in MEC services and then proposed *Fatriot*, a SmartNIC for MEC that as well as a way to ensure service availability in the event of those conditions. Despite functioning as a Network Interface Card (NIC), it can monitor the service status of the MEC host. Upon detecting a failure, it relays traffic to backup hosts, ensuring continuous service operation. Notably, it has standalone capabilities, allowing it to establish a connection with the backup host even under a host system down. We anticipate that *Fatriot* will assume a critical role in the Internet of Everything (IoE) era, as the proliferation of connected devices increases. It will contribute significantly by enhancing mission-critical services availability to preventing fatal malfunctions.

CRediT authorship contribution statement

Taejune Park: Writing – original draft, Validation, Software. **Myoungsung You:** Writing – review & editing, Investigation. **Jinwoo Kim:** Writing – review & editing, Writing – original draft, Conceptualization. **Seungsoo Lee:** Writing – review & editing, Writing – original draft.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The data that has been used is confidential.

Acknowledgments

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2022R1C1C1006967, RS-2022-00166401).

References

Allan, T., Brumley, B.B., Falkner, K., Van de Pol, J., Yarom, Y., 2016. Amplifying side channels through performance degradation. In: Proceedings of the 32nd Annual Conference on Computer Security Applications. pp. 422–435.

Anwar, S., Inayat, Z., Zolkipli, M.F., Zain, J.M., Gani, A., Anuar, N.B., Khan, M.K., Chang, V., 2017. Cross-VM cache-based side channel attacks and proposed prevention mechanisms: A survey. J. Netw. Comput. Appl. 93, 259–279.

Asrari, A., Ansari, M., Khazaei, J., Cecchi, V., 2020. Real-time blackout prevention in response to decentralized cyberattacks on a smart grid. In: 2020 IEEE Texas Power and Energy Conference. TPEC, IEEE, pp. 1–5.

Bala, A., Chana, I., 2012. Fault tolerance-challenges, techniques and implementation in cloud computing. Int. J. Comput. Sci. Iss. (IJCSI) 9 (1), 288.

Cheraghlou, M.N., Khadem-Zadeh, A., Haghparast, M., 2016. A survey of fault tolerance architecture in cloud computing. J. Netw. Comput. Appl. 61, 81–92.

Chiang, M., Zhang, T., 2016. Fog and IoT: An overview of research opportunities. IEEE Internet Things J. 3 (6), 854–864.

Coppolino, L., D'Antonio, S., Mazzeo, G., Romano, L., 2017. Cloud security: Emerging threats and current solutions. Comput. Electr. Eng. 59, 126–140.

ETSI, 2024. Multi-access Edge Computing (MEC) https://www.etsi.org/technologies/multi-access-edge-computing.

Fang, X., Misra, S., Xue, G., Yang, D., 2011. Smart grid—The new and improved power grid: A survey. IEEE Commun. Surv. Tutor. 14 (4), 944–980.

Gerla, M., Lee, E.-K., Pau, G., Lee, U., 2014. Internet of vehicles: From intelligent grid to autonomous cars and vehicular clouds. In: 2014 IEEE World Forum on Internet of Things. WF-IoT, IEEE, pp. 241–246.

Grover, J., Garimella, R.M., 2018. Reliable and fault-tolerant IoT-edge architecture. In: 2018 IEEE Sensors. IEEE, pp. 1–4.

He, F., Yan, X., Liu, Y., Ma, L., 2016. A traffic congestion assessment method for urban road networks based on speed performance index. Procedia Eng. 137, 425–433.

Horii, M., Kojima, Y., Fukuda, K., 2018. Stateful process migration for edge computing applications. In: 2018 IEEE Wireless Communications and Networking Conference. WCNC, IEEE, pp. 1–6.

Hu, Y.C., Patel, M., Sabella, D., Sprecher, N., Young, V., 2015. Mobile Edge Computing—-A Key Technology Towards 5G. ETSI White Pap. 11 (11), 1–16.

ISGMEC ETSI, 2019. Multi-Access Edge Computing (MEC); Framework and Reference Architecture. Tech. Rep. 2016.

Junior, P.S., Miorandi, D., Pierre, G., 2020. Stateful container migration in geo-distributed environments. In: 2020 IEEE International Conference on Cloud Computing Technology and Science. CloudCom, IEEE, pp. 49–56.

Khan, W.Z., Ahmed, E., Hakak, S., Yaqoob, I., Ahmed, A., 2019. Edge computing: A survey. Future Gener. Comput. Syst. 97, 219–235.

Koren, I., Krishna, C.M., 2020. Fault-Tolerant Systems. Morgan Kaufmann.

Lee, K., Kim, M., Park, T., Chwa, H.S., Lee, J., Shin, S., Shin, I., 2019. MC-SDN: Supporting mixed-criticality real-time communication using software-defined networking. IEEE Internet Things J. 6 (4), 6325–6344.

Lu, N., Cheng, N., Zhang, N., Shen, X., Mark, J.W., 2014. Connected vehicles: Solutions and challenges. IEEE Internet Things J. 1 (4), 289–299.

Mai, T., Yao, H., Guo, S., Liu, Y., 2020. In-network computing powered mobile edge: Toward high performance industrial iot. IEEE Netw. 35 (1), 289–295.

Mao, Y., You, C., Zhang, J., Huang, K., Letaief, K.B., 2017. A Survey on Mobile Edge Computing: The Communication Perspective. IEEE Commun. Surv. Tutor. 19 (4), 2322–2358.

NetFPGA, 2024. NetFPGA-SUME https://netfpga.org/NetFPGA-SUME.html.

NetFPGA, 2024. NetFPGA-SUME github https://github.com/NetFPGA/NetFPGA-SUME-public,

Open vSwitch, 2024. Open vSwitch, http://openvswitch.org/.

Pace, P., Aloi, G., Gravina, R., Caliciuri, G., Fortino, G., Liotta, A., 2018. An edge-based architecture to support efficient applications for healthcare industry 4.0. IEEE Trans. Ind. Inform. 15 (1), 481–489.

Paolucci, F., Cugini, F., Castoldi, P., Osiński, T., 2021. Enhancing 5G SDN/NFV edge with P4 data plane programmability. IEEE Netw. 35 (3), 154–160.

Papadimitratos, P., De La Fortelle, A., Evenssen, K., Brignolo, R., Cosenza, S., 2009. Vehicular communication systems: Enabling technologies, applications, and future outlook on intelligent transportation. IEEE Commun. Mag. 47 (11), 84–95.

Park, T., Shin, S., Shin, I., Lee, K., 2021. Formullar: An FPGA-based network testing tool for flexible and precise measurement of ultra-low latency networking systems. Comput. Netw. 185, 107689.

Park, T., You, M., Cui, J., Jin, Y., Lee, K., Shin, S., 2022. MECaNIC: SmartNIC to assist URLLC processing in multi-access edge computing platforms. In: 2022 IEEE 30th International Conference on Network Protocols. ICNP, IEEE, pp. 1–12.

Patel, M., Naughton, B., Chan, C., Sprecher, N., Abeta, S., Neal, A., et al., 2014. Mobile-edge Computing Introductory Technical White Paper. pp. 1089–7801, White paper, mobile-edge computing (MEC) industry initiative.

Perez-Botero, D., Szefer, J., Lee, R.B., 2013. Characterizing hypervisor vulnerabilities in cloud computing servers. In: Proceedings of the 2013 International Workshop on Security in Cloud Computing. pp. 3–10.

Pfaff, B., Pettit, J., Koponen, T., Jackson, E., Zhou, A., Rajahalme, J., Gross, J., Wang, A., Stringer, J., Shelar, P., Amidon, K., Casado, M., 2015. The design and implementation of open vswitch. In: 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15). USENIX Association, Oakland, CA, ISBN: 978-1-931971-218, pp. 117–130, URL https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/pfaff.

Popiolek, P.F., Mendizabal, O.M., 2012. Monitoring and analysis of performance impact in virtualized environments.

Popiolek, P.F., dos Santos Machado, K., Mendizabal, O.M., 2021. Low overhead performance monitoring for shared infrastructures. Expert Syst. Appl. 171, 114558.

Rameshan, N., 2016. On the role of performance interference in consolidated environments. In: IEEE/USENIX International Conference on Autonomic Computing. ICAC, KTH Royal Institute of Technology.

RedHat, 2024. Linux KVM: Kernel Virtual Machine https://www.linux-kvm.org/page/Main_Page.

Ricart-Sanchez, R., Malagon, P., Alcaraz-Calero, J.M., Wang, Q., 2019. P4-netfpga-based network slicing solution for 5G MEC architectures. In: 2019 ACM/IEEE Symposium on Architectures for Networking and Communications Systems. ANCS, IEEE, pp. 1–2.

Sabella, D., Vaillant, A., Kuure, P., Rauschenbach, U., Giust, F., 2016. Mobile-edge computing architecture: The role of MEC in the internet of things. IEEE Consum. Electron. Mag. 5 (4), 84–91.

Sabharwal, N., Wali, P., Sabharwal, N., Wali, P., 2013. Cloud Capacity Management. Springer.

Samanta, A., Esposito, F., Nguyen, T.G., 2021. Fault-tolerant mechanism for edge-based IoT networks with demand uncertainty. IEEE Internet Things J. 8 (23), 16963–16971.

Sorin, D., 2022. Fault Tolerant Computer Architecture. Springer Nature.

Sun, H., Yu, H., Fan, G., Chen, L., 2020. Qos-aware task placement with fault-tolerance in the edge-cloud. IEEE Access 8, 77987–78003.

Suneja, S., Isci, C., De Lara, E., Bala, V., 2015. Exploring vm introspection: Techniques and trade-offs. Acm Sigplan Notices 50 (7), 133–146.

Tao, Z., Xia, Q., Hao, Z., Li, C., Ma, L., Yi, S., Li, Q., 2019. A survey of virtual machine management in edge computing. Proc. IEEE 107 (8), 1482–1499.

Tuli, S., Casale, G., Jennings, N.R., 2022a. Dragon: Decentralized fault tolerance in edge federations. IEEE Trans. Netw. Serv. Manag. 20 (1), 276–291.

Tuli, S., Casale, G., Jennings, N.R., 2022b. Pregan: Preemptive migration prediction network for proactive fault-tolerant edge computing. In: IEEE INFOCOM 2022-IEEE Conference on Computer Communications. IEEE, pp. 670–679.

Uhlemann, E., 2015. Introducing connected vehicles [connected vehicles]. IEEE Veh. Technol. Mag. 10 (1), 23–31.

Wang, L., Chen, S., Chen, F., He, Q., Liu, J., 2023a. B-detection: Runtime reliability anomaly detection for MEC services with boosting LSTM autoencoder. IEEE Trans. Mob. Comput..

Wang, P., Xu, J., Zhou, M., Albeshri, A., 2023b. Budget-constrained optimal deployment of redundant services in edge computing environment. IEEE Internet Things J..

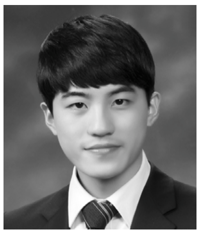
Zandi, F., Zadeh, S.A., Abbasloo, S., Pazhooheshy, P., Ganjali, Y., Hu, Z., 2023. Live stateful migration of a virtual sub-network. In: NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium. IEEE, pp. 1–9.

Zhou, Y., Wang, Z., Zhou, W., Jiang, X., 2012. Hey, you, get off of my market: detecting malicious apps in official and alternative android markets. In: NDSS, Vol. 25, No. 4. pp. 50–52.

Zilberman, N., Audzevich, Y., Covington, G.A., Moore, A.W., 2014. NetFPGA SUME: Toward 100 Gbps as research commodity. IEEE Micro 34 (5), 32–41.

Taejune Park is an assistant professor at the Department of Artificial Intelligence Convergence, Chonnam National University, South Korea. He received B.S. in Computer Engineering at Korea Maritime and Ocean University, South Korea, and M.S. and Ph.D. in information security at KAIST, South Korea. His research interests focus on network and IoT security and reliable/low-latency communications.

Myoungsung You is a Ph.D. candidate in the School of Electrical Engineering at KAIST. He received his M.S. degree from the Graduate School of Information Security at KAIST and his B.S. degree in Computer Science from Chungbuk National University. His research interests include programmable network data planes, cloud security, and distributed systems.



Jinwoo Kim is an assistant professor in the School of Software at Kwangwoon University, Seoul, South Korea. He received his Ph.D. degree in School of Electrical Engineering and his M.S degree in Graduate School of Information Security from KAIST, and his B.S degree from Chungnam National University in Computer Science and Engineering. His research topic focuses on investigating security issues with software defined networks and cloud systems.

Seungsoo Lee is an assistant professor in the Department of Computer Science and Engineering at Incheon National University, Incheon, South Korea. He received his B.S. degree in Computer Science from Soongsil University in Korea. He received his Ph.D. degree and M.S. degree both in Information Security from KAIST. His research interests focus on cloud computing and network systems security. He is especially focusing his attention on software-defined networking (SDN), network function virtualization (NFV), containers, and its security issues.