Aalto University
School of Science
Master's Programme in Security and Cloud Computing

Adika Bintang Sulaeman

A Highly-Available Multiple Region Multi-access Edge Computing Platform with Traffic Failover

Master's Thesis
Espoo, July 29, 2020

Supervisors: Professor Antti Ylä-Jääski, Aalto University
Professor Panagiotis Papadimitratos, KTH
Advisors: Dr. Kimmo Hätönen, Nokia Bell Labs
Marco Spanghero, KTH

Aalto University
School of Science
Master's Programme in Security and Cloud Computing

ABSTRACT OF
MASTER'S THESIS

Author: Adika Bintang Sulaeman

Title:
A Highly-Available Multiple Region Multi-access Edge Computing Platform with Traffic Failover

Date: July 29, 2020 **Pages:** 57

Major: Security and Cloud Computing **Code:** SCI3084

Supervisors: Professor Antti Ylä-Jääski, Aalto University
Professor Panagiotis Papadimitratos, KTH

Advisors: Dr. Kimmo Hätönen, Nokia Bell Labs
Marco Spanghero, KTH

One of the main challenges in the Multi-access Edge Computing (MEC) is steering traffic from clients to the nearest MEC instances. If the nearest MEC fails, a failover mechanism should provide mitigation by steering the traffic to the next nearest MEC. There are two conventional approaches to solve this problem, i.e., GeoDNS and Internet Protocol (IP) anycast. GeoDNS is not failover friendly because of the Domain Name System (DNS) cache lifetime. Moreover, the use of a recursive resolver may inaccurately translate the IP address to its geolocation. Thus, this thesis studies and proposes a highly available MEC platform leveraging IP anycast. We built a proof-of-concept using Kubernetes, MetalLB, and a custom health-checker running on the GNS3 network emulator. We measured latency, failure percentage, and Mean Time To Repair (MTTR) to observe the system's behavior. The performance evaluation of the proposed solution shows an average recovery time better than one second. The number of failed requests and latency overhead grows linearly as the failover time and latency between two MECs increases. This thesis demonstrates the effectiveness of IP anycast for MEC applications to steer the traffic to the nearest MEC instance and to enhance resiliency with minor overhead.

Keywords: multi-access edge computing, traffic failover, anycast, high availability

Language: English

Acknowledgements

I would like to express my gratitude to all people whose help and assistance help me finishing the thesis. First of all, I would like to thank the supervisors/examiners and advisors from Aalto University and KTH: Professor Antti Ylä-Jääski, Professor Panagiotis Papadimitratos, and Marco Spanghero. My deepest gratitude goes to the Nokia Bell Labs, especially Dr. Kimmo Hätönen, for the guide, assistance, and help throughout the technical and writing sections of the thesis.

My thanks also go to my thesis opponent, Abhishek Kumar Misra, whose feedback on my thesis has been very constructive. I would also give my gratitude to the European Union with the Erasmus+ scholarship to help me study in these universities.

Last but not least, I would like to thank my family and my fiancée for their support throughout my life and study.

Espoo, July 29, 2020

Adika Bintang Sulaeman

Abbreviations and Acronyms

3GPP	3rd Generation Partnership Project
AF	Application Function
API	Application Programming Interface
ARP	Address Resolution Protocol
AS	Autonomous System
ASN	Autonomous System Number
AWS	Amazon Web Services
BFD	Bidirectional Forwarding Detection
BGP	Border Gateway Protocol
CDN	Content Delivery Network
CNCF	Cloud Native Computing Foundation
DN	Data Network
DNS	Domain Name System
DoS	Denial of Service
eBGP	external Border Gateway Protocol
ECMP	Equal-Cost Multi-Path
ECS	EDNS Client Subnet
EIGRP	Enhanced Interior Gateway Routing Protocol
EKS	Elastic Kubernetes Service
ETSI	European Telecommunications Standards Institute
FQDN	Fully Qualified Domain Name
GCP	Google Cloud Platform
GKE	Google Kubernetes Engine
GSLB	Global Server Load Balancer
iBGP	internal Border Gateway Protocol
IGP	Interior Gateway Protocol
IoT	Internet of Things
IP	Internet Protocol
IS-IS	Intermediate System to Intermediate System
ISP	Internet Service Provider

IXP	Internet Exchange Point
LADN	Local Area Data Network
LDP	Label Distribution Protocol
LER	Labeled Edge Routers
LSP	Label-Switched Path
LSR	Label Switched Routers
LTE	Long-Term Evolution
MEC	Multi-access Edge Computing
MEO	Multi-access Edge Orchestrator
MNO	Mobile Network Operator
MPLS	Multiprotocol Label Switching
MRAI	Minimum Route Advertisement Interval
MTBF	Mean Time Between Failure
MTTF	Mean Time To Failure
MTTR	Mean Time To Repair
MTU	Maximum Transmission Unit
NB-IoT	Narrowband IoT
NEF	Network Exposure Function
OOM	Out Of Memory
OSPF	Open Shortest Path First
OSS	Operations Support Systems
P Router	Provider Router
PE Router	Provider Edge Router
PoP	Point of Presence
RAN	Radio Access Network
REST	Representational state transfer
RFC	Request For Comments
RIP	Routing Information Protocol
RPS	Requests Per Second
SBA	Service-Based Architecture
SLA	Service Level Agreement
TCP	Transmission Control Protocol
TTL	Time to live
UDP	User Datagram Protocol
UE	User Equipment
UPF	User Plane Function
VIP	Virtual Internet Protocol
VPN	Virtual Private Network

Contents

Chapter 1

Introduction

Internet of Things (IoT) generates a continuous data stream, resulting in a large volume of data. Sending all data to a remote process that is too far from the data source may not be economical due to the data transfer cost. Furthermore, higher network round-trip time from clients to remote processes may increase the overall application latency. Edge computing, loosely defined as a system that brings computation closer to the source of data, aims to shorten the latency by minimizing round-trip time between two geographically separate processes. In this thesis, edge computing refers to Multi-Access Edge Computing (MEC), an ecosystem that provides a platform to deploy applications in the Radio Access Network (RAN) of the 5G system. With edge computing, it is possible to filter and compress data near the data source before sending it to a remote storage or give adequate responses to the IoT device to operate with a shorter latency.

A single MEC system aims to serve clients within the specified area within a few kilometers to keep the network latency minimized. Thus, companies may want to deploy multiple applications in different MECs to serve more users in different locations. MEC ecosystem must ensure the end-users access the most optimally performing MEC services and provide failover or backpressure handling if the MEC services are not responsive to enhance the system's uptime in case of failure, improving the overall quality of MEC.

Megasense project [7][6] is a large scale IoT project that deploys a massive number of geographically scattered sensors. It can take the benefit of the MEC data offloading so that data can be filtered and compressed before sending it to a remote persistent data storage. This thesis focuses on building an availability-improved MEC platform to minimize the downtime and data loss in several failure scenarios.

1.1 Research Questions

There are some challenges in building an availability-improved MEC. Finding the nearest MEC application instance is an inherent challenge of this type of applications. This thesis formulates the following research questions to address the previously mentioned challenges:

- **RQ1. How does the MEC system help the clients connect to the optimal MEC application instances?**

  The optimal MEC application instances are the nearest healthy MEC application instances. A healthy MEC application instance is the one that can respond to a client request within a user-defined timeout. The question also applies to a situation where there is a failure in the nearest healthy MEC application instance.

- **RQ2. How to enhance the availability of MEC application instances?**

  The answer to the research question **RQ1** must always hold even if the nearest MEC fails to keep the availability high. Availability refers to the ability of the whole system to keep serving clients despite an ongoing failure. The availability of a system implies the reliability. This research question focuses on how to provide the availability on the infrastructure level, not on the application level.

1.2 Objectives and Thesis Contributions

The main objective of the thesis is to study and propose a MEC system that can answer the research question **RQ1** and **RQ2**, which is to find the correct MEC application instances and to enhance the availability of a MEC system. The key contributions of this thesis are as follows:

- Studying, proposing, and building a proof of concept of a MEC platform that helps clients discover the nearest MEC application instances.

- Studying, proposing, building a proof of concept, and discussing enhancement to the availability of a MEC system.

1.3 Methodology

This thesis follows the design science methodology to achieve its objectives. There are seven guidelines of design science methodology [34] which this thesis follows:

- Design as an artifact: This thesis delivers proof of concepts as a software and simulation as artifacts
- Problem relevance: The objectives of this thesis is to develop a solution for Nokia Bell Labs, especially for the MegaSense project [7]
- Design evaluation: The evaluation involves testing the system and analyze the data of experiments to measure correctness, performance, security and reliability
- Research contributions: This thesis aims to contribute to the field of edge computing by studying a certain way to improve availability and discusses the trade-offs
- Research rigor: The thesis relies on thorough literature review, design, experiment, and evaluation
- Design as a search process: To find the right artifact to propose, this thesis relies on finding and combining the existing approaches and applying it to the MEC problems it focuses on
- Communication of research: The thesis is presented to academia, technology-oriented fellows in industry

1.4 Scope and Delimitation

The thesis' primary focus is on the MEC infrastructure level, especially on the IP network layer and above. Therefore, there are some assumptions made as readily available systems.

The first assumption is that this thesis assumes the 5G core network functions are already implemented and accessible via a particular programming interface. For example, European Telecommunications Standards Institute (ETSI) defines some of the 5G functionalities, such as Network Exposure Function (NEF) is accessible through a Representational state transfer (REST) Application Programming Interface (API). Second, the thesis assumes that multiple MECs belong to a single Mobile Network Operator

(MNO). Thus, the managed MECs belong to the same Autonomous System (AS) number.

A network emulation software is used to develop the proof of concept of the MEC network infrastructure. Due to the limitation of the network emulator used, the connection from the user equipment (clients) to the 5G network is omitted. The capability of the network nodes in the topology built is limited to the inherent capability of the software used.

1.5 Ethics and Sustainability

This thesis is built on top of various free and open-source software libraries and tools. All of the libraries and tools used for this thesis have either `MIT` or `Apache-2.0` license, while the emulator has `GNU GPL` license. This thesis does not violate the terms and conditions of the licenses.

The data gathered in this thesis contains neither personal nor production data. Thus, this thesis has no ethical issues regarding the data exposure. Also, this thesis does not fabricate any data for the experiment. The related work of others as the ground material of this thesis has been appropriately cited.

The sustainability of this thesis is supported by leveraging existing protocols to solve the research problems. The thesis proposes a solution that should work backward compatible and is possible to extend for future needs.

1.6 Structure of the Thesis

The thesis is organized as follows. Chapter 2 reviews the literature and gives background information of the project, MEC, and related works. Chapter 3 shows the design and implementation detail of the proposed solution. Chapter 4 gives the experiment detail as well as the result of the experiment and measurement. Chapter 5 provides discussion and analysis on the availability enhancement achieved and the security consideration of the system. Finally, Chapter 6 concludes this thesis and discusses the future work.

Chapter 2

Background

This chapter provides a study of the related technologies, techniques, and literature. It begins with the overview of Nokia Bell Labs IoT data streaming platform and its use for the MegaSense project, a project aiming to gather environment data with IoT technology accurately. Then, it discusses the MEC infrastructure and its framework. The chapter continues with the conventional techniques to help clients discover the nearest application instances and their tradeoffs. This chapter also discusses the standard availability enhancement techniques. Last but not least, it presents some of the related work to this thesis.

2.1 Megasense Project and Nokia Bell Labs IoT Data Streaming Platform

2.1.1 Megasense Project

MegaSense is a joint research project by several parties, including Nokia Bell Labs and the University of Helsinki, aiming to collect data from both low-end and high-end sensors and leverage machine learning to calibrate data from low-end sensors [7][6]. The sensor devices in a sensor network send data to a single sink or gateway. This gateway is connected to the internet using various interfaces, such as Ethernet, Narrowband IoT (NB-IoT), Long-Term Evolution (LTE), and the upcoming 5G network. The gateway or sink sends the sensor data stream to the Nokia Bell Labs IoT data streaming platform, which processes, stores, and manages the dissemination of the sensor data.

2.1.2 Nokia Bell Labs IoT Data Streaming Platform

The Nokia Bell Labs IoT data streaming platform is a distributed system-based platform built by Nokia Bell Labs to ingest, process, and disseminate data from IoT devices or other network equipment devices. The system is based on microservices with the asynchronous messaging pattern, utilizing a pluggable message broker such as Kafka or ActiveMQ. The critical components of this streaming platform are:

- Data Fetcher (DF): Sensors in the IoT network sends data to a gateway. DF can run in the gateway or farther machine, such as MEC. DF is responsible for data pre-processing such as filtering and compressing, before sending it to the remotely deployed Data Switch (DS).
- Data Switch (DS): DS is the pluggable message broker, such as Apache Kafka. DS runs in MEC or cloud.
- Data Hub (DH): DH is the endpoint for the users or operators. For example, if users want to see some data, they send a request to the DH. The DH then subscribes to DS to fetch the necessary data.
- Coordinator: The Coordinator is responsible for managing and coordinating all components of the IoT data streaming platform.

The IoT data streaming platform can virtually be deployed in various underlying infrastructures, such as on-demand cloud infrastructure such as Amazon Web Services (AWS) or Google Cloud Platform (GCP), or on-premise cloud and edge infrastructure. The deployment options of this streaming platform may vary depending on the available resources and the system's goal. The first and simplest option is to deploy everything on the cloud. This approach is the simplest to deploy and manage because every component runs in a single location, which makes deployment and monitoring easier. However, the downside is that it introduces higher latency and is cost-ineffective since gateway needs to send all sensor data to the cloud. The second option is to have DF on the sink/gateway of the sensor network, and the other components on MEC or cloud. In this way, the data sent from the client-side to the remote data processors is already filtered and compressed, which improves the efficiency of the data transmission.

2.2 Multi-Access Edge Computing (MEC)

Multi-access edge computing (MEC) is a platform that provides means of deploying applications on the 5G RAN. In a simple term, MEC brings cloud

capabilities closer to the clients. MEC aims to bring computation closer to the data source, so that data offloading and processing can be done not too far from the data source. User equipment devices, such as mobile phones, IoT devices, and autonomous cars, are the prime example of the data sources [56]. With MEC, the network round-trip time and the data sent to a further endpoint can be minimized, improving the quality of the service and reduces the data transfer cost.

The MegaSense project is one of the use cases of MEC. If one million devices send temperature, humidity, and air pollution data every second in a region, after 6 hours, there will be 21.6 million data sent from all devices. The cost of data transfer for a large amount of data may be expensive, especially if it must transit through a paid network. The IoT devices instead send data to the MEC, and the MEC application compresses the data before sending it to the persistent database in a remote-cloud. Another example is when the IoT devices need some feedback from the remote application. Interacting with a MEC instead of a farther cloud reduces the overall application latency by shortening the network's round trip time.

2.2.1 MEC and 5G

5G technology embraces MEC by providing means to deploy MEC in the user plane of the 5G network. 5G architecture is based on Service-Based Architecture (SBA), in which each component communicates with each other via a defined Application Programming Interface (API) [9]. Figure 2.1 shows the components of 5G system architecture. According to The European Telecommunications Standards Institute (ETSI), MEC is deployed in the `N6` reference point, which is the interface between User Plane Function (UPF) and Data Network (DN), which is an external IP network.

5G system provides some critical enablers to MEC [9]. First, 5G provides a concept of Local Area Data Network (LADN), a geographically isolated network that can provide high data rate, low latency, and service localization in 5G [43]. The location of the LADN is within the DN component, which has the `N6` interface connection to the UPF, as shown in Figure 2.1. 5G core network is responsible for selecting the UPF to route the traffic to the LADN, known as Local Routing and Traffic Steering. 5G core network and Application Function (AF) can exchange information via Network Exposure Function (NEF) for various purposes, such as exposing bandwidth manager, User Equipment (UE) identity, and radio network information APIs.

There are two main APIs that engineers might consider when architecting and developing MEC applications [52]. The first one is `Mx2` API [28], which enables the operators to interact with the MEC applications, such as

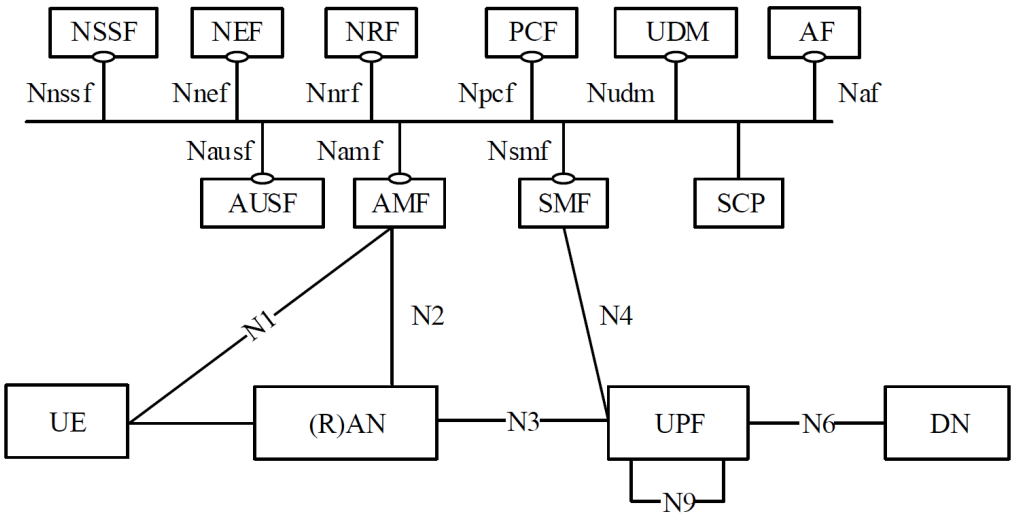

Figure 2.1: 5G SBA system architecture [9]

giving commands to start, stop, and delete MEC applications. The second one is Mp1 API [25], which allows MEC applications to interact with MEC systems so that they can make service discovery of and consume other MEC applications. There are also Radio Network Information API [27] and Location API [26] although these APIs are not mandatory for deploying MEC applications. Figure 2.2 shows the entities of MEC in the 5G ecosystem with its reference points.

In the Figure 2.2, the blue boxes are the MEC platform infrastructure components. The operators deploy, delete, and manage MEC application services through Operations Support Systems (OSS) software. The OSS requests operations to the Multi-access Edge Orchestrator (MEO). The MEO validates the requests from OSS and checks the request to the company policy. Then, MEO sends requests to the MEC Platform Manager. The MEC Platform is responsible for providing the runtime functionalities to the running MEC application services, such as service registry, DNS handling, and filtering rules control.

2.2.2 Intra-MEC Network

A MEC can be considered as a microdata center which is close to the data sources. Hence, the network architecture can be thought of as a smaller version of a typically much bigger data center. MEC hosts are virtualized on top of bare-metal machines, such as the Nokia AirFrame Open Edge

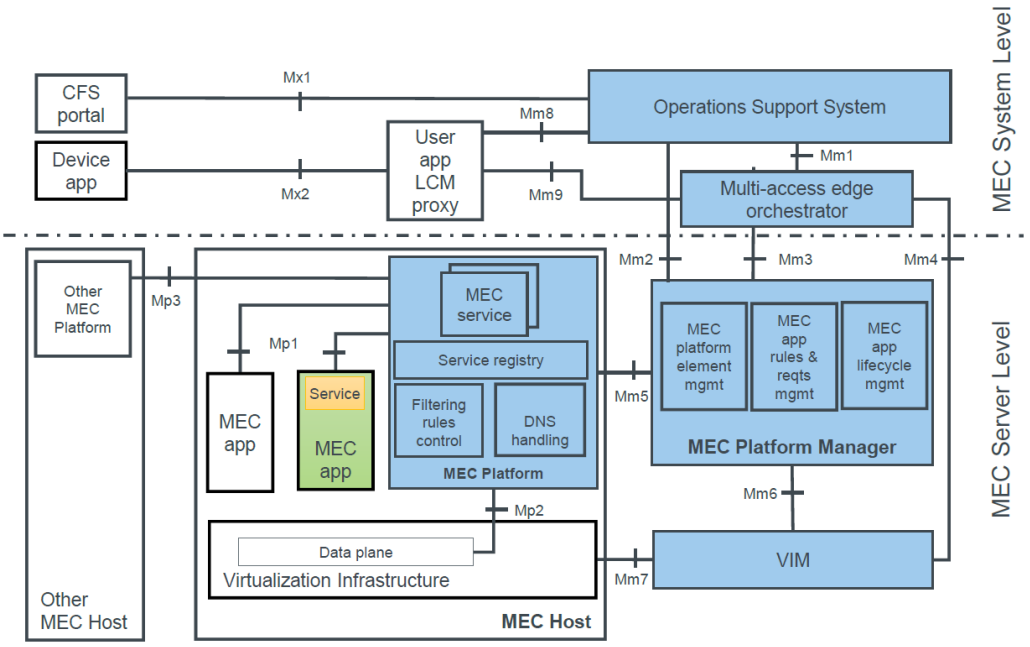

Figure 2.2: Entities in MEC [52]

Server [1], with type-1 hypervisors such as VMWare, Xen or Linux KVM. The compute resources on the MEC can be managed by a platform, such as OpenStack and OpenVIM [54].

The internal MEC network topology can follow the Clos network topology, which consists of the spine and leaf layer to build a full-mesh host network [23]. The leaf layer consists of Top of Rack (ToR) switches, and the spine layer consists of switches that connect all ToR switches in the leaf layer [23][10]. Figure 2.3 illustrates the network architecture with Clos topology of a MEC. This network architecture is a simplified version of the network architecture found in the data center. MEC uses a border router to connect itself to the outside network [14].

Inside the MEC, the best practice is to run external Border Gateway Protocol (eBGP) with private Autonomous System (AS) numbers (64512 to 65535) for the nodes inside [23]. Using eBGP instead of internal Border Gateway Protocol (iBGP) is simpler to manage because it does not rely on Internal Gateway Protocol (IGP) to distribute the advertisement and best path selection. To prevent the count-to-infinity problem, the leave nodes have different AS numbers, but the spine nodes have the same AS numbers, except for inter-POD nodes. In this way, since the path lengths from the border router to every MEC host are the same, the border router can load

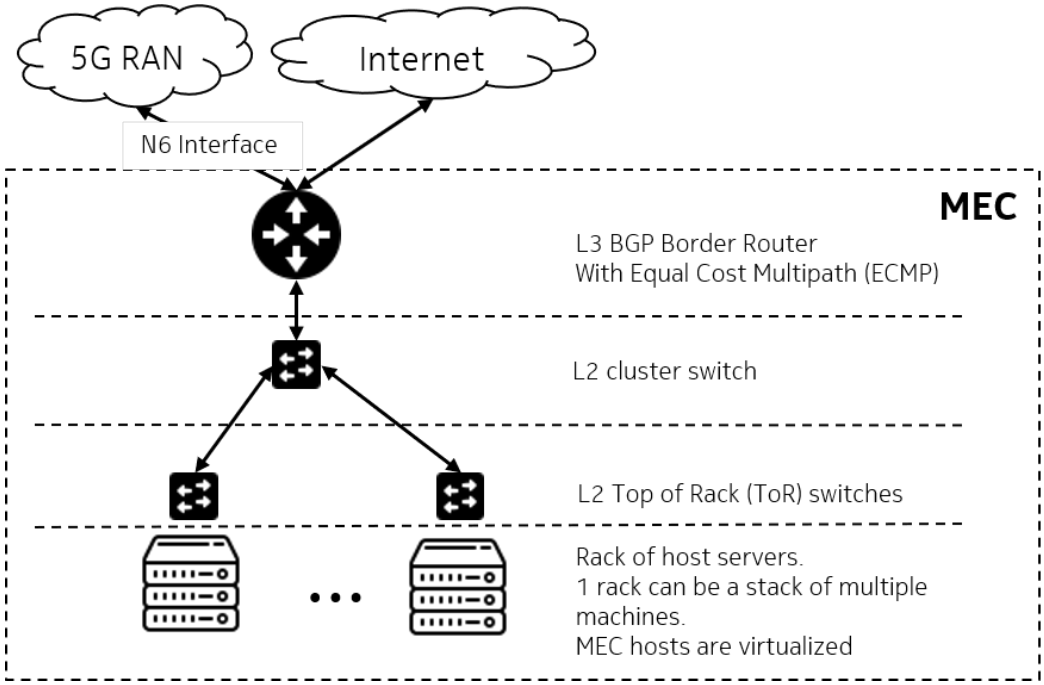

Figure 2.3: Intra-MEC Network Architecture

balance the traffic with Equal Cost Multipath (ECMP).

2.2.3 Inter-MEC Network Architecure

The `N6` interface connects the MEC to the users in the mobile network via RAN. The border router is connected to the edge router of the Internet Service Provider (ISP) or MNO to have the Internet connection [41]. The backbone network of the ISP can use any IGP routing protocol to operate, such as Open Shortest Path First (OSPF), Routing Information Protocol (RIP), Intermediate System to Intermediate System (IS-IS), or Enhanced Interior Gateway Routing Protocol (EIGRP) [14]. One of the options to build a dedicated network interconnecting these MECs on top of the ISP backbone network is with Multiprotocol Label Switching (MPLS) Virtual Private Network (VPN). Figure 2.4 illustrates the inter-MEC network architecture.

MPLS uses labels in layer-2 instead of IP addresses to route the traffic. In this way, it does not rely on the IP routing table in the routers in the ISP backbone network. The MPLS path is called the Label-Switched Path (LSP) [30]. In the MPLS network, the edge routers are called Labeled Edge Routers (LER) or Provider Edge (PE) routers. MEC border routers connect to the

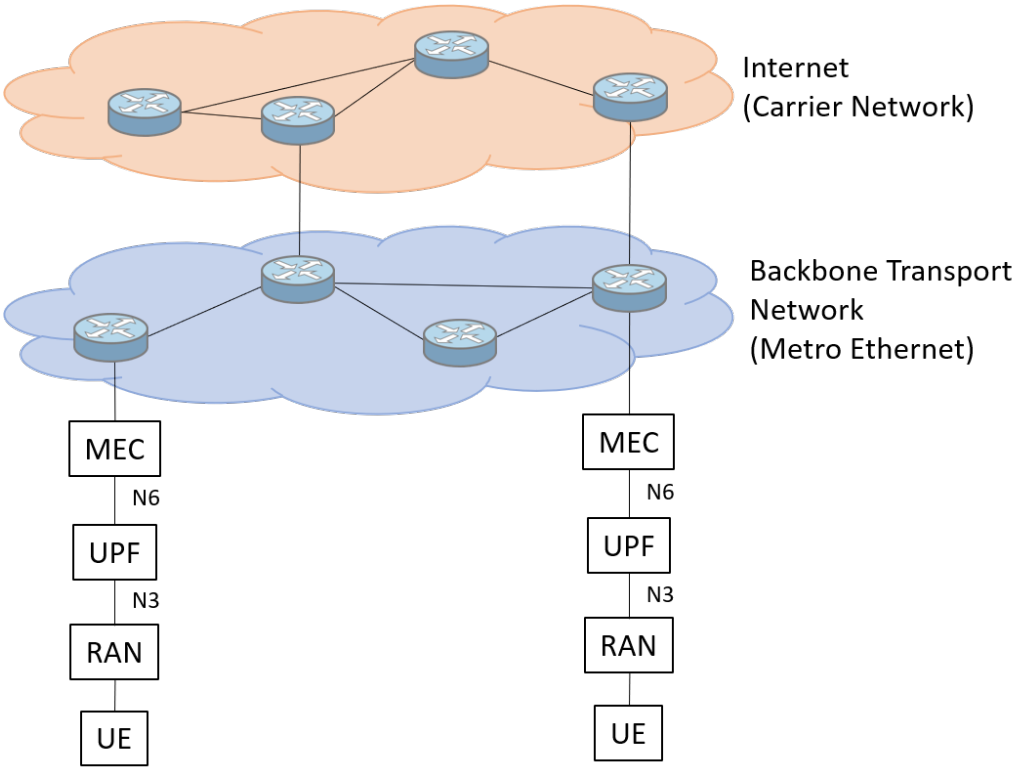

Figure 2.4: Inter-MEC Network Architecture

PE routers. The intermediate routers in the LSP are called Label Switched Routers (LSR) or Provider (P) routers. The most straightforward protocol to distribute the label of MPLS is Label Distribution Protocol (LDP), which relies on IGP routing protocol such as OSPF or RIP to distribute the label.

2.2.4 MEC and Cloud Computing

MEC is not a replacement for the cloud technology. MEC is intended as the complementary technology for cloud systems. With MEC, the applications may follow the fog for a transparent computing framework [50]. This framework divides the system into an end-user layer consisting of IoT devices, edge server layers consisting of edge services, core network layers comprising the core internet network infrastructure, and cloud layer comprising clusters of servers.

Megasense project follows the transparent computing framework. The data source is the IoT devices. The data offloading happens in both IoT

gateway, referred to as fog, and MEC, referred to as the edge. The data may be optionally sent to a remote persistent data storage in the cloud. Following this abstraction is useful when there is a need to offload data at multiple layers.

2.2.5 Kubernetes as a Platform for MEC

There are some flavors of Kubernetes designed for edge computing, such as KubeEdge and K3S. KubeEdge is a Cloud Native Computing Foundation (CNCF) sandbox project that aims to adapt Kubernetes as a platform for edge computing [4]. K3S, built by Rancher, is a lightweight and stripped-down version of Kubernetes [3]. K3S uses a lightweight `containerd`, an alternative container engine as a replacement of Docker and uses sqlite3 to replace `etcd` as the default storage.

As previously mentioned, MEC consists of several host machines and an edge router. The Kubernetes as a platform to orchestrate the MEC applications, runs on bare-metal hosts. Unlike exposing applications in a managed Kubernetes solution such as Amazon Elastic Kubernetes Service (EKS) or Google Kubernetes Engine (GKE), exposing deployed Kubernetes service on bare-metal environments requires some cumbersome works. For instance, exposing any service, including `Ingress Controller` with Kubernetes' `Service` type `LoadBalancer`, cannot be done out-of-the-box. Exposing services with `Service` type `NodePort` is difficult to manage in the long run and does not scale well. To alleviate this problem, a project named MetalLB leverages Address Resolution Protocol (ARP) and Border Gateway Protocol (BGP) to expose the services [5].

In the MetalLB ARP mode, one Kubernetes node announces the services' IP addresses to the border router with ARP protocol. In other words, one node spoofs the IP addresses of the services. When incoming traffic arrives at that node, the traffic is forwarded to the service by `KubeProxy`. In the MetalLB BGP mode, all nodes make a BGP peering session with the border router. MetalLB also has a BGP speaker, which tells the border router how to route the traffic to the intended services. MetalLB in BGP mode is more complicated than in ARP mode. However, MetalLB in BGP mode allows load balancing on the IP layer level if the border router supports Equal Cost Multi-Path (ECMP). One caveat with ECMP is if one node fails, the running connections to different nodes may also be disrupted depending on the hashing algorithm used [44]. For example, with consistent hashing algorithms such as Maglev, at most only k/n connections will be disrupted, where k is the number of active connections and n is the number of nodes [24].

2.3 Connecting Clients to the Nearest MEC Application Instances

The clients start to communicate with the MEC application instance by making a service discovery. Service discovery, in this context, is defined as the mechanism that allows the clients to query the IP address associated with the domain of a service. The service registry, which lets the MEC services register themselves, is coupled with the service discovery provider. A prominent example of a service discovery is DNS-based service discovery.

It is essential to build a secure service discovery. Since the service discovery is the first step clients do before making any contact with the intended services, a false returned IP address or disruption of the service discovery provider's availability can be fatal. Trabelsi et al. suggest the requirements of secure service discovery [57]:

- Authentication: the client is sure that the reply comes from the right service registry.
- Privacy: the client's privacy is protected so that any parties cannot learn the pattern and do inference attack.
- Access control: service registry may want to advertise the services to the authorized clients only.
- Availability: service registry is protected to any attack that may disrupt the availability, such as Denial of Service (DoS) attack.

In addition, in the MEC context, service discovery provides a way for clients to discover the nearest available MEC service. Achieving this goal while maintaining the requirements of secure service discovery is not a trivial task.

ETSI suggests several considerations when building service discovery solutions for MEC services [52]:

- DNS-based: clients of the edge services may not be aware of edge services, and must be able to get the IP address of the edge services using DNS queries. This aims to ease of deployment by not introducing new protocols to existing deployed clients [8].
- Domain name: the service must register the Fully Qualified Domain Name (FQDN) that clients know and can query to DNS server to get the IP of the FQDN

There are two common techniques to help clients traffic go to the nearest application instances, GeoDNS and anycast.

2.3.1 GeoDNS

If there are multi-region MEC services, MEC can adapt the GeoDNS technique to return the nearest MEC service to the client. This technique, also known as Geo-IP mapping, is commonly used in Content Delivery Network (CDN) [35]. GeoDNS receives IP address clients (in the DNS context, clients are also called resolvers) and gets the coordinate of the resolving by looking up from a database. It then measures the distance between the clients and services and returns the nearest service IP. GeoDNS is simple to implement, since it relies only on DNS implementation to achieve its goal. However, GeoDNS-based approach has several disadvantages.

The first distinct disadvantage is that GeoDNS is not the best way to provide failover because of the DNS caching in the recursive DNS resolver or the end devices themselves. If the application for the respective IP address fails, and the DNS updates its database for the next query, the update is not updated to the cache of the recursive DNS resolver. The window time between the failure and the next query is when clients cannot talk to the respective endpoint. From our experience, some IoT devices even query DNS only once upon start-up and do not respect the DNS cache timeout.

Another disadvantage is if the clients connect to a public DNS resolver, such as Google's or Cloudflare's public DNS, the GeoDNS sees the public DNS resolver IP address instead of the clients' IP address, which can mislead GeoDNS to not return the nearest endpoint to the client. This is not desirable because it makes the Geo-IP mapping useless. For example, if the client's location is in Helsinki, and the DNS resolver is in Munich, and there are two server endpoints in Stockholm and Aachen, the Geo-IP mapping will not give an advantage to the client. If the DNS recursive resolver is provided by the ISP, which is supposedly not too far from the clients, the impact might not be that significant [40].

The motivation behind public DNS is to let users change their preferred DNS because the DNS provided by ISP may violate privacy by selling browsing history to advertisers [15], may be less performant, or may block some domain name [49][2]. DNS EDNS Client Subnet (ECS) (`RFC 7871`) mitigates this problem by bringing the clients' subnets to the recursive DNS queries [16].

The DNS ECS solution in `RFC 7871` [16] does not come without a problem. The first issue with bringing clients' subnets to the DNS recursive queries is the privacy of the clients [16][21]. Using the ECS-enabled DNS

query makes it possible for the third party to learn the services and clients geographical distribution. Due to this privacy related-issue, `RFC 7871` recommends the ECS is disabled by the DNS and suggests that any approach that introduces additional metadata be avoided in order to keep users privacy [16]. As a result, many local ISP DNSes do not implement this feature.

2.3.2 Anycast

Anycast refers to a practice where multiple discrete instances have a single IP address [36]. Clients traffic go to the nearest application instances because of the shortest path that a routing protocol chooses. Anycast is a common practice for DNS operations [18].

Unlike the GeoDNS approach, anycast neither suffer the privacy-related issue nor the DNS cache and Time To Live (TTL) problems. Since it does not rely on the DNS query, the failover between one instance to another might be faster, depending on the routing convergence time within the network. Another advantage of anycast is that it helps absorb DoS attack localized to the nearest application instance only, and keep the other application instances untouched [48].

Anycast also has some downsides. In practice, it is more complex to set up than GeoDNS because it requires some network set up by the ISP in the backbone network. Furthermore, if the failure happens too often, which means the route changes too often, it leads to route flapping, which prevents the network from converging [18]. Anycast is inherently not aware of the system load. Hence, it will always route the traffic to the nearest destination regardless of its conditions.

2.4 Related Works

Google combined anycast and load balancing for some of its application services [58]. The application was running behind a load balancer, and the load balancer leveraged IP anycast. The application instances were replicated behind the load balancer. If an instance failed, the load balancer knew by monitoring the heartbeat and removed the instance from the load balancer entry. The global route changed only happen when the load balancer failed, not when the application instances failed.

Dropbox used Global Server Load Balancer (GSLB) on the DNS level, which decided which IP address it should return by combining Geo-IP mapping techniques with Real User Metrics [32]. Real User Metrics was a collection of Point of Presence (PoP) performance, obtained by Dropbox desktop

clients. The desktop clients periodically probed multiple PoPs and sent the results to the aggregator. In this way, the GSLB was aware of the performance of each PoP.

More advanced techniques to route traffic between edges can also be done with traffic engineering. Facebook developed `Edge Fabric`, an SDN-based solution to steer traffic to the optimal CDN by modifying BGP routes of peered routers [53]. `Edge Fabric` continuously monitored the performance and capacity of the CDNs. It dynamically changed the routing table of the BGP routers to steer the traffic based on the monitored CDN performance and capacity. Google built `Espresso`, an SDN-based approach to do global traffic engineering [61]. `Espresso` built the routing path in a centralized server and applied the routing with MPLS to the MPLS switches as the data plane. MPLS was commonly used for traffic engineering by applying label switching, allowing a routing that is independent of the routers' IP routing tables [55]. `Espresso` also had BGP speakers to peer with other BGP routers. `Espresso` enabled Google to do application-aware traffic routing.

Chapter 3

Designing and Building Highly Available MEC Systems

This chapter discusses the proof of concept of the proposed solution. It begins with the system requirements, adopted design principles, and the system design of the proposed solution.

3.1 Requirements

To deliver the desired system, the design must ensure that it can be integrated into the existing mechanisms. There are several requirements that the proposed solution has to meet:

- The operation of the proposed solution must comply with the 3GPP standards and/or ETSI group specifications for MEC.
- The MEC infrastructure is transparent to the clients, and the clients may not be aware of MEC at all. The proposed solution must not rely on changing the behavior of the clients or to change the protocol that is already standardized.
- The MEC application must be stateless. In other words, the clients' requests do not depend on the previous responses from the MEC applications.

3.2 Design Principles

The system design of this thesis follows these principles:

- Pragmatic approach: the solution tries to solve an engineering problems pragmatically. The implemented system design as the proof of concept will be measured to see its effectiveness in solving the problems.
- Seamless integration: the solution solves the problem seamlessly, without breaking changes or backward compatibility issues with the existing status of software stacks and protocols.
- Leveraging existing tools, libraries, software, and protocols: the solution uses existing open source tools and protocols to build the solution.

3.3 System Design

This section discusses the system design of the MEC system and how it improves the availability. It starts with an overview of the system. Then, it goes deeper by discussing the design of the MEC internal network, followed by the inter-MEC network.

3.3.1 Overview of the System Design

MEC application instances leverage anycast addressing to help routing clients' traffic to the nearest instance. The main reason for the use of anycast addressing instead of GeoDNS is because many IoT devices do not resolve the hostname when the cache is expired, which does not allow the traffic to go to a different endpoint throughout the lifetime of the IoT devices. Anycast also allows for faster failover than the DNS because DNS has cache timeout and no strict implementation on the client's side to do DNS lookup when the DNS timeout expires.

Figure 3.1 shows the generic design of MEC system. The internal MEC network follows the architecture explained in Section 2.2.2 enhanced with Bidirectional Forwarding Detection (BFD) to detect hosts or links failure quickly. With BFD, the border router can detect the failure in around 50 milliseconds by consuming only limited bandwidth in its operation [19]. BFD works by making a session between any two nodes and sending a periodic message to notify their life. If the session is down and several periodic message is not received, the link is considered down.

Every application instance exposed to the outside world is allocated a virtual Internet Protocol (VIP) address. The BGP speaker advertises the VIP address to the MEC border router. A custom agent monitors the application performance by fetching the performance data from the default system monitor. The agent also probes the application instances periodically as a

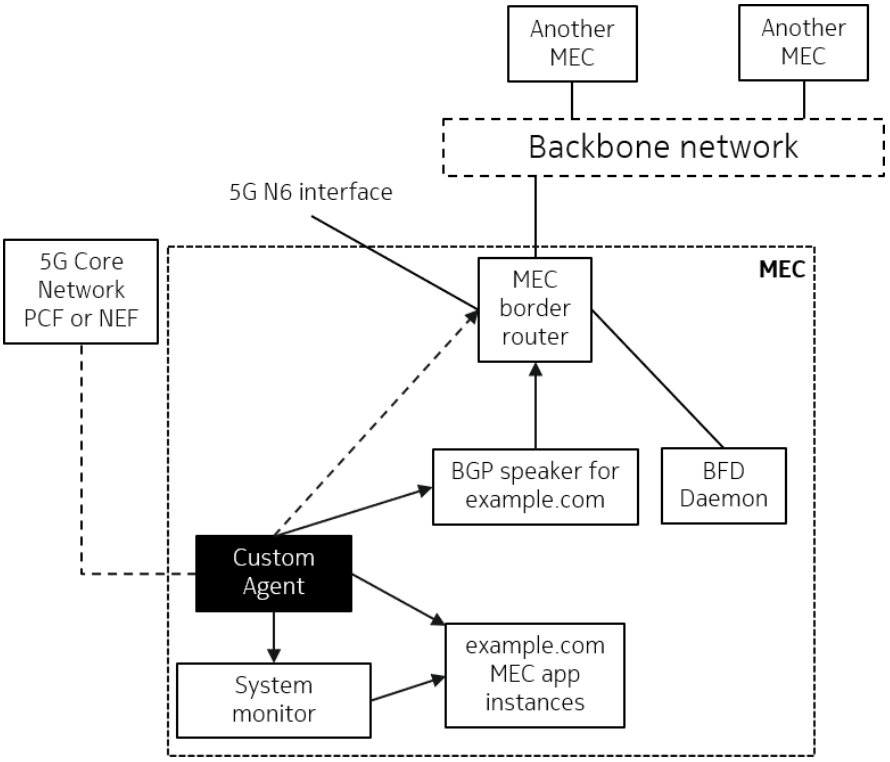

Figure 3.1: Internal MEC Design

health check. If an application instance does not respond to the probes, the agent can ask the BGP speaker to stop advertising the associated VIP address. BFD daemon runs on every node so that the MEC border router knows instantly if the MEC hosts are running.

This thesis also experimented with a proxy in front of the MEC application instances to investigate a further possibility of enhancement. The proxy may help implement rate-limiting, Transport Layer Security (TLS) termination, and IP addresses blocking in case there is a need to block malicious IP addresses. This thesis examines the effectiveness of a rate-limiter as a proxy, while the other tasks such as TLS termination and IP address blocking are left for future work.

Figure 3.2 shows the inter-MEC network design. The ISP backbone network provides an inter-MEC network with MPLS LSP. With MPLS, the routing for the traffic does not rely on the IP routing table on the router, but on the label assigned on the router in between layer-2 and layer-3 of the protocol. MPLS LSP makes the routing independent of the routing table and makes it an MPLS VPN. Combined with anycast, this inter-MEC network

is the failover path in case of a complete application failure in one MEC.

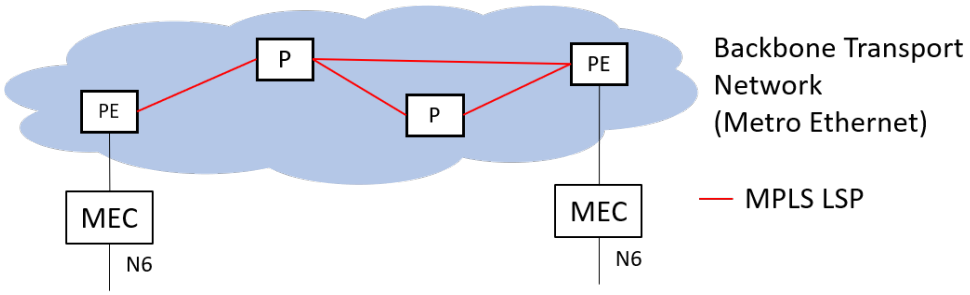

Figure 3.2: Inter-MEC Network Design

3.3.2 Algorithms for MEC Traffic Failover

The MEC application instances fail if they are unable to serve clients' requests. Failure happens for various reasons, but this thesis focuses on two main reasons: MEC hosts failure, or the application is overloaded and fails to restore its state.

If a failure happens, MEC stops advertising the VIP address of the failed MEC application instances. Since the application instance uses anycast, a new advertisement containing different routes to the same VIP address destination will come to the border routers of MEC. The network converges when all nodes in the network have received an advertisement and form a steady path. The failover time is bound to the network convergence time because the traffic relies on inter-MEC network that the routing protocol builds.

In a multiple-operator network, the advertisement interval may vary. The variance in the advertisement interval can lead to exponential convergence time [29]. However, all of the MEC resides in a single backbone network in this inter-MEC network design, and the advertisement time can be set homogeneous to prevent the exponential convergence time problem.

The agent, as shown in Figure 3.1, monitors the health of the MEC application instances. It probes the port of the application instances periodically and monitors the CPU usage of the MEC application instances. The agent decides that the MEC should stop advertising the VIP address of an application if the agent fails to probe the MEC application instance subsequently or if the application hits CPU limit usage set by the operator for a prolonged amount of time. It must be noted that the limit set must not be too small to

avoid route flapping, but not too large to keep the failover's responsiveness. Algorithm 1 shows the pseudocode of how the agent works.

Algorithm 1: Algorithms for the agent to do the health check

```
   Data: interval, appFailureCounter, threshold
 1 start the agent;
 2 allApps = read file configuration;
 3 while true do
 4 |   wait(interval);
 5 |   foreach app ∈ allApps do
 6 |   |   status = probe host;
 7 |   |   if status == unhealthy then
 8 |   |   |   appFailureCounter[app] += 1;
 9 |   |   |   if appFailureCounter[app] > threshold then
10 |   |   |   |   stop advertising VIP of the app;
11 |   |   |   end
12 |   |   end
13 |   end
14 end
```

If some hosts fail, the traffic still goes to the same MEC location, although there will be some possible connection resets by the router when calculating the equal-cost multipath. If all hosts in a MEC fail, the border router removes those failed hosts from the routing table. Then, the new route coming from the anycast advertisement will re-route the traffic to the second nearest MEC instance.

3.3.3 Enhancing Availability Using Rate Limiting

Rate limiting means limiting incoming rates to a particular defined limit. If the rate of the incoming traffic from a particular IP address exceeds the limit, the rate limiter will return an error to the request. This thesis used `Nginx` as the rate limiter.

`Nginx` uses leaky bucket algorithm to implement the rate-limiting [45]. Leaky bucket algorithm puts the incoming requests in a buffer (or a bucket as an analogy) and processes the requests with a maximum rate of R. If the rate of the incoming requests is larger than R and the buffer is full, the incoming requests "spills" out of the buffer and not processed [11]. In this way, the rate received by the MEC application instance is limited to R requests per second for each client.

3.4 Proof of Concept Implementation

The proof of concept consists of two main parts. The first one is the agent that monitors the application instances and decides on the failover, written in Golang 1.14. The second part is the MEC ecosystem emulation. GNS3 Network Emulator is used to build the network topology. All the routers used in the emulator run Linux OS, with FRRouting installed to provide the Linux kernel with routing capabilities. Kubernetes and MetalLB are used as the orchestration framework and as a BGP speaker for the exposed MEC application instances. An open-source BFD daemon, `aiobfd`, is used as daemons for all the MEC hosts. Table 3.1 summarizes the tools used to build the proof of concept.

Software Name	**Version**	**Function**
GNS3	2.2.6	Network emulator
Kubernetes	v1.18.2	Orchestration framework
FRRouting	v7.3	Software suite providing router capability to Linux
MetalLB	v0.9.3	Load-balancer for bare metal Kubernetes
aiobfd	v0.2	BFD daemon written in Python
client-go	v0.18.0	Golang client library to interact with Kubernetes cluster

Table 3.1: Software and libraries used

3.4.1 Internal-MEC Implementation

The implementation of the internal MEC system follows the design in Figure 3.1. There is at least one border router as the ingress and egress of the MEC. The border router makes eBGP peers session with MEC hosts as well as the router in the backbone network. The border router and all connected nodes inside the MEC are BFD enabled to detect a link or node fault early. The border router runs Linux version 5.3 with `fib_multipath_hash_policy=1` to enable per-flow load balancing [39]. Per-flow load balancing decides the path to select based on the source IP address, source port, destination IP address, and destination port. In this way, the multipath hash policy helps avoid transport layer packet re-ordering.

Kubernetes is installed on all MEC hosts, referred to as Kubernetes nodes in the Kubernetes context. Kubernetes `metrics-server` is installed to read the resource consumption, such as CPU and memory usage, of the applications deployed. The agent reads the resource consumption data through

the Kubernetes API server. The BFD daemon is installed as `DaemonSet` so that it runs in every Kubernetes node. The BFD `DaemonSet` must be set `hostNetwork=True` to expose its port without load-balancing the traffic by the `KubeProxy`.

The agent, named `srvbend`, is deployed as the type `Deployment`. To make the decision done by the `srvbend` simple and consistent, the number of replicas needed is only one. Figure 3.3 illustrates the design implementation of the internal system of the MEC.

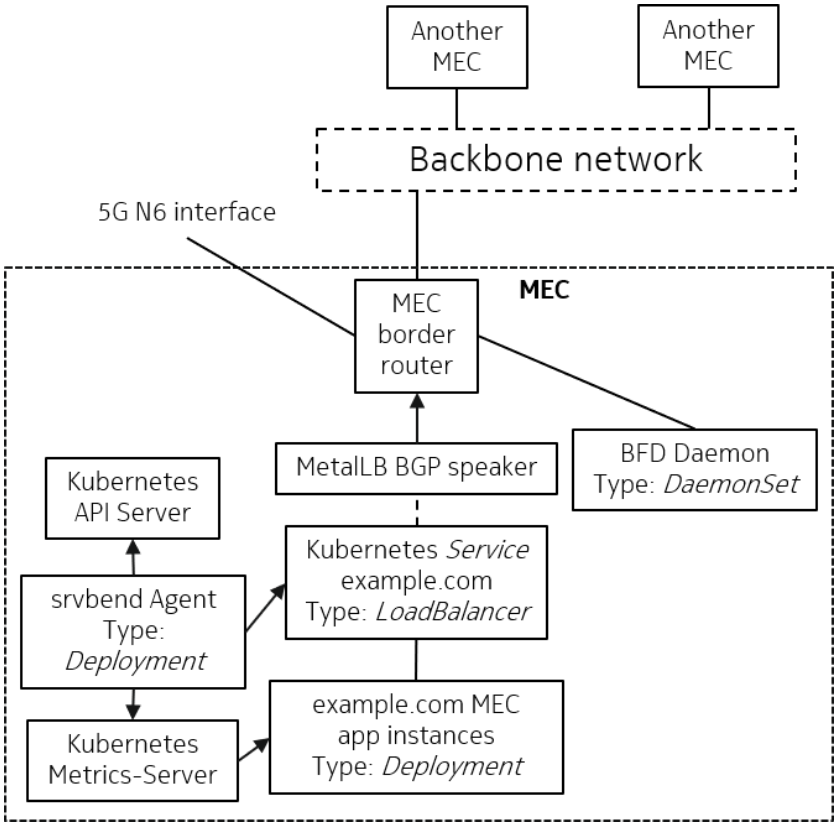

Figure 3.3: Internal MEC implementation diagram

MEC application instances are typically deployed with type `Deployment`, although the application owner is not restricted to choose another type such as `DaemonSet`. Independent MEC application should be ideally exposed with `Service` type `LoadBalancer` with `externalTrafficPolicy: Local` set. In this way, the border router load balances the incoming traffic with equal cost to the nodes where the application instance `Pod`s are running. MetalLB looks at the `Service` with type `LoadBalancer`, assigns a VIP address for

that `Service`, and the MetalLB's BGP speaker advertises the VIP address to the border router.

If a proxy is used in front of the MEC application instances, the proxy can be deployed as a Kubernetes ingress controller. The ingress controller has a Kubernetes `Service` with type `LoadBalancer`, in which the MetalLB BGP speaker will announce the external VIP address. The ingress controller will forward the traffic to the internal `Service` of the MEC application, which has the type of `ClusterIP`. The Kubernetes `Ingress` sets the rules to forward the traffic. The rate-limiting can be configured by changing the configuration of the ingress annotation [46].

3.4.2 Inter-MEC Network Implementation

The implementation of the inter-MEC network follows the design in Figure 3.2, which is implemented as shown in Figure 3.4. MECs are connected via an MPLS network, forming a full-mesh MEC network. LDP is configured on the routers in the backbone network to generate labels and exchange them between MPLS routers. Each MEC is connected to PE router, and the PE routers might be connected via P routers.

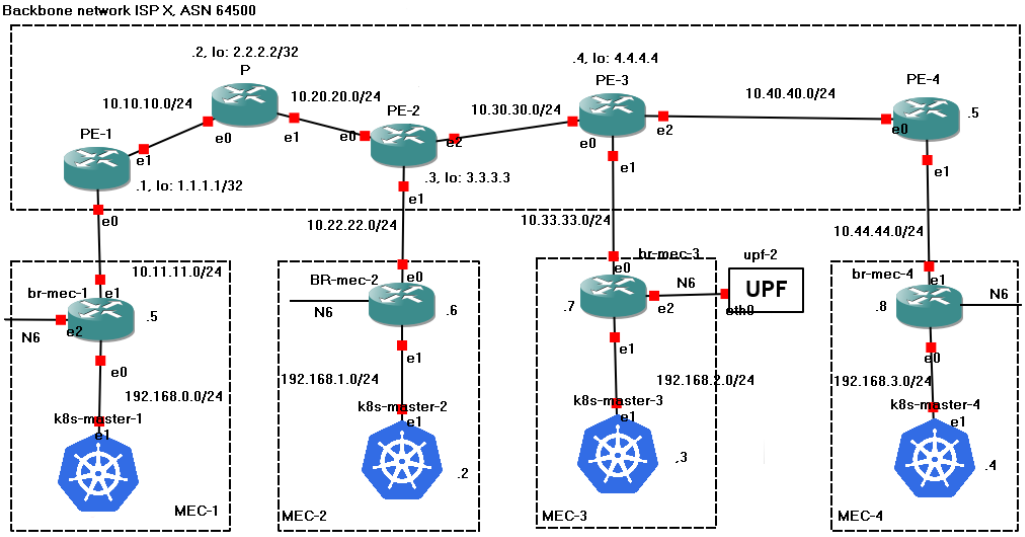

Figure 3.4: Inter-MEC network via backbone network emulated in GNS3

The interior network routing protocol of the backbone network is OSPF with area 0. PE and P routers are routed with OSPF via their loopback

interface. All PE routers form iBGP sessions with Minimum Route Advertisement Interval (MRAI) set to 5 seconds, as suggested by the RFC 4271 [59].

Chapter 4

Experiment

This chapter provides the details of the experiment. It starts with the detail of the experiment testbed setup. Then, it defines the experiment metrics and how to measure them. Finally, it shows the result of the metrics measurement.

4.1 Experiment Testbed Setup

We built a testbed on GNS3 network emulator to experiment with the proposed solution. The GNS3 network emulator server that hosted the virtual machines ran on a remote host owned by Nokia. The remote host ran Linux Ubuntu 18.04 with 10 CPU cores and 32 GB of RAM. Since all the virtual machines ran on a single machine, the network latency between the nodes in the experiment was negligible. Each router in the experiment had 1500 MB of RAM and one virtual CPU, while the MEC hosts had 3300 MB of RAM and three vCPUs.

The experiment setup consisted of four MECs and one cloud environment. The MECs were connected to a single backbone network of a certain ISP with a certain AS number, while the cloud belonged to another AS. These two different backbone networks were connected via an Internet Exchange Point (IXP). Table 4.1 shows the AS numbers set up in the experiment.

A sample application addressed with IP anycast was deployed in all MECs and cloud. This sample application was an HTTP server written in Python 3 with Flask framework version 1.1.1. Each application instance `Pod` had 30 milli-CPU to run. To emulate the latency in the backbone network, each router in the backbone network shaped the traffic using `tc` command to add 6 ms additional latency for each of its network interfaces. Figure 4.1 shows the complete setup of the system.

Name	ASN
MEC-1 (border router)	65533
MEC-2 (border router)	65534
MEC-3 (border router)	65535
MEC-4 (border router)	64513
ISP X	64500
ISP Y	64501
Cloud (border router)	65536

Table 4.1: AS numbers of MECs, ISPs, and clouds

4.2 Method of Experiment

The experiment compares the data of the experiment metrics with and without the proposed solution. The goal is to observe the improvement in the availability in respect to some design decision that this thesis takes [42]. The method for the experiment loosely follows the Chaos Engineering approach [51]. In this approach, the experiment defines the steady state behavior of the MEC application services, introduces events that can disrupt the system, hypothesize the system state under those events, and takes measurement to observe the deviation from the hypothesis. The steady state is defined as:

Steady State. *The latency and successful requests that clients perceive when interacting with MEC application services in an optimal condition*

Then, we introduce two events that may disrupt the availability of the MEC application services:

- **e0**: MEC application instances stop running.
- **e1**: An extreme increase of traffic from clients.

The event **e0** happens when there is something wrong with the application that can be killed by the underlying system, such as when it has memory leak and eventually killed by Out Of Memory (OOM) killer, or a bug in the code of the MEC application that can cause it to stop working. This case is also possible when the Kubernetes fails to schedule the `Pod`s to the nodes, while the nodes are still maintaining BFD connection with the border router. The event **e1** happens when the number of clients within the served area is increasing or under a DoS attack. Given the steady-state and the events introduced, we come up with a hypothesis:

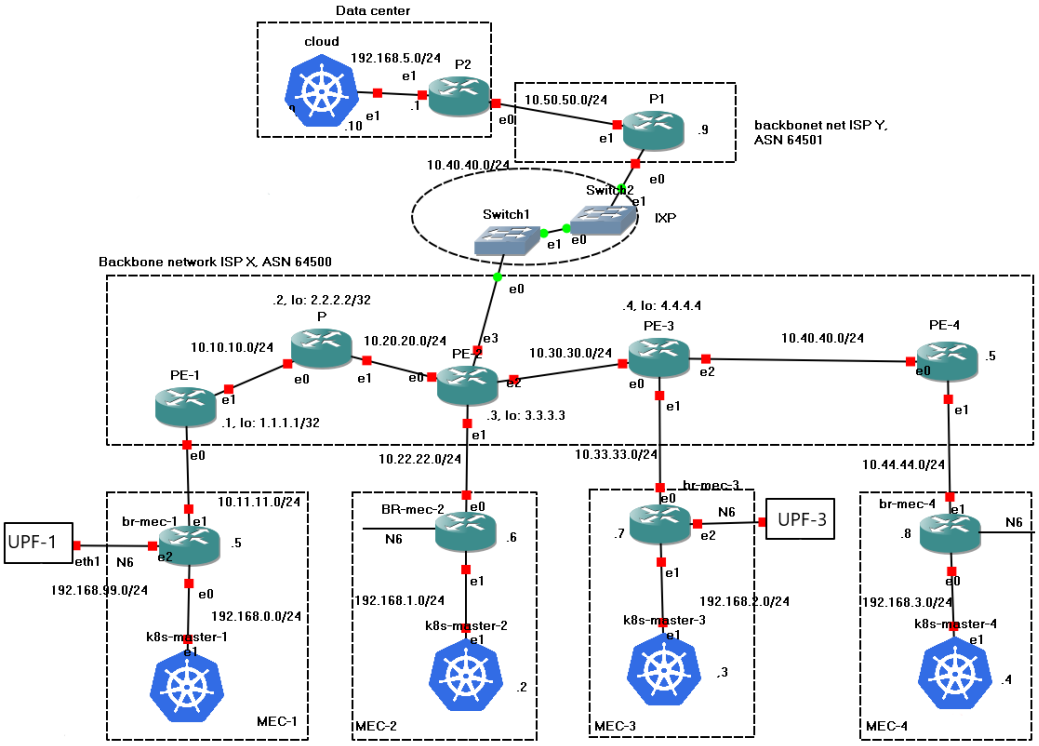

Figure 4.1: Full setup of the experiment

Hypothesis. *The events **e0** and **e1** occurring to the system will not change the system behavior from the steady state 4.2*

Experiment metrics need to be measured to disprove the Hypothesis. We are interested in how far the results deviate from the Hypothesis.

4.3 Metrics Measurement

The experiment aims to analyze whether the proposed solution answers the research questions mentioned in the Section 1.1. To see how effective the proposed solution answers the research question **RQ1**, which asks how the MEC steers the clients' traffic to the nearest MEC instances, and the research question **RQ2**, which asks how to enhance the availability, the following metrics need to be observed:

- Latency; defined as the perceived time by the client to send a request to the MEC application service and receive a response. The primary

focus is how a failure or disruption affects the latency perceived by clients.

- Percentage of failed requests; the failed request is defined as the request from the client that receives no response from the MEC application services. The main focus is to observe the number of failed requests in case of a failure.

- Mean Time To Repair (MTTR); defined as the average time of a system to recover from a failure [13]. We will observe how long the MTTR is when a failure happens.

The goal of the measurement is to observe how the proposed solution improves the resiliency of MEC systems. By observing the resiliency improvement, we can infer the enhancement in the overall availability and the quality of service that MEC systems may offer. As previously mentioned, we will measure and observe several metrics.

The first metric to measure is the base latency and failure percentage. The base latency and failure percentage are latency and failure percentage that clients perceived in a normal and optimal condition. These are the basis used to define the Steady State.

The second metric to measure is the MTTR of an application with the plain Kubernetes-based failover. This metric will provide insights for how long the application is unavailable before applying the solution. We can also say that this is the base case for the failure.

After applying the solution, the failover time is measured to observe the time it takes to shift the traffic to another MEC. The failover time affects the overall MTTR after the solution has been applied.

Lastly, two major events are introduced to disrupt the system, and the measurement will observe how the system reacts. The first event is the failure event, which is separated into two sub events. The first sub event is the application instance failure, and the second sub event is the MEC hosts failure. The second event is an enormous traffic that resembles a DoS attack.

4.3.1 Measuring Latency from Clients to Sample Application Instances in Different MECs

This measurement aimed to get the latency of the system in an optimal condition that defined the Steady State. We measured the base latency from the client to MEC application instances by measuring the time it took to complete a request to every MEC application instances. One application

instance was also deployed on the cloud without any failover mechanism, intended as the sink of all traffic in case of all MECs fail.

We measured the latency from a client connected to UPF-1 to the MEC application instances in various locations. The nearest application instance in an increasing order were MEC-1, MEC-2, MEC-3, MEC-4, and cloud, as it is shown in Figure 4.1. The distance is determined by the number of hops from the UPF to the MECs. Table 4.2 shows the latency from the clients to the same sample application located in different MECs and cloud measured with 33 times of repetition. In addition to the latency data in the table, there was no failure happened in this base case scenario.

UPF-1 to	**Average latency**	**Median latency**	**Standard deviation latency**
MEC-1	6.09	6.0	0.9
MEC-2	60.35	60.0	0.95
MEC-3	82.91	83.0	1.29
MEC-4	142.09	106.5	43.82
Cloud	90.03	90.0	0.94

Table 4.2: Latency for the Steady State

4.3.2 Measuring Mean Time to Repair From an Application Instance Failure

The first case to consider was when all of the application instances, which were instantiated as `Pod`s, fail. During the experiment, a client made an HTTP GET request to the MEC application instance every 100 milliseconds. Then, the application instance (`Pod`) in the nearest MEC was deliberately deleted.

Kubernetes provides resiliency by maintaining the number of `Pod`s running as specified in the Kubernetes `Deployment`. In this way, clients do not perceive noticeable availability issues unless all `Pod`s are deleted. If all `Pod`s are deleted, clients will notice a failure until all `Pod`s are ready. The time window between the failed `Pod` to restore is the time to repair, which is affected by the Kubernetes system's ability to re-spawn the `Pod`s and the application-dependent startup time.

Figure 4.2 shows the time to repair of the sample application. Based on the experiment with our sample application with 13 samples in the experiment,

the MTTR for the pod deletion was 17171.92 milliseconds with a median of 17138.0 milliseconds and a standard deviation of 538.28 milliseconds.

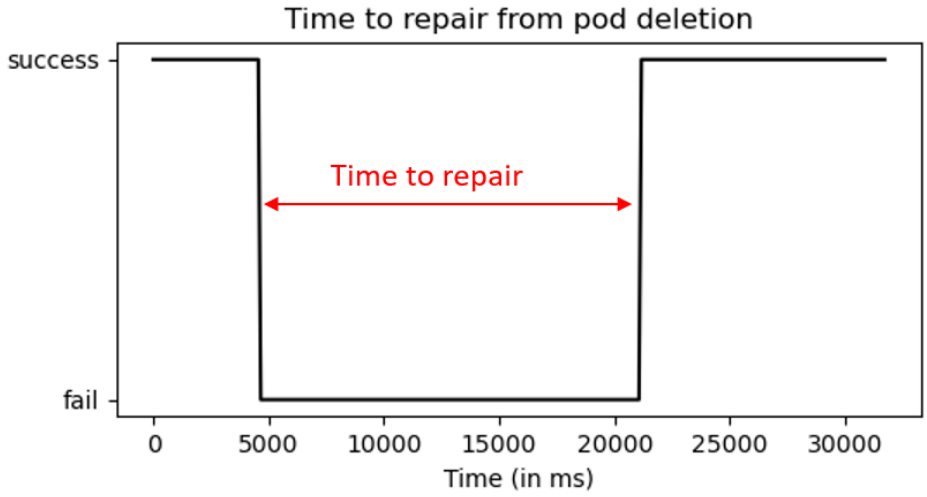

Figure 4.2: Time to repair from a pod deletion

4.3.3 Measuring Failover Time

One of the ways to measure the failover time is by observing the number of packet loss during a constant stream of data sent from a client to an application instance [37]. Figure 4.3 illustrates the technique to measure failover time.

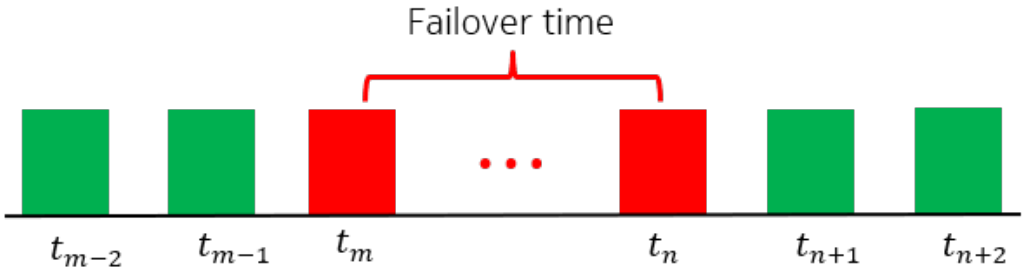

Figure 4.3: Measuring failover time

To measure the failover, a client sends a single User Datagram Protocol (UDP) packet at a constant rate, which is every ten milliseconds in this case.

The data sent must not be larger than the Maximum Transmission Unit (MTU) of an IP datagram to make the packet loss observation easier. The client sends a unique, incremental identifier to the server. The packet loss is detected by the number of missing identifiers received by the application instance. The failover time is the number of packet loss multiplied by the transmission interval. Equation 4.1 formulates the failover time measurement.

$$Failover\ time = lost\ packets * transmission\ interval \tag{4.1}$$

Based on the experiment, the average failover time from the nearest MEC-1 to another nearest MEC is less than 1 second. If the next nearest healthy MEC application instance is in MEC-2, the average failover time is 596.67 millisecond with a standard deviation of 159.21 millisecond. The failover to MEC-3 takes an average of 793.33 millisecond with a standard deviation of 184.79 millisecond. The failover to the farthest MEC, MEC-4, takes an average of 678.34 milliseconds with a standard deviation of 277.59 milliseconds. Figure 4.4 shows the bar plot of the failover time with the standard deviation as the error plot.

The bar plot in Figure 4.4 does not show incremental failover time despite the incremental distance of the alternative MEC locations. The reason is because a route advertisement does not come from the distant alternative MEC every time there is a failure. Instead, it comes from the PE router that is connected to the MEC border router. For example, consider the routing table for the destination address of 11.11.11.11 for a particular MEC application below. Every PE router is connected to MEC forms a full mesh network. As a result, they have the same AS path even though the OSPF path is different, which then affects the best path selection due to IGP metrics difference [17]. Because PE router already keeps the path to the other MEC location, if the directly connected MEC withdraws the route, this PE router will advertise the next best route to the directly connected MEC border router.

```
    Network          Next Hop      LocPrf    Path
*>  11.11.11.11/32   10.11.11.5              65533 64512 ?
*  i                 5.5.5.5       100       64513 64512 ?
*  i                 4.4.4.4       100       65535 64512 ?
*  i                 3.3.3.3       100       65534 64512 ?
```

Listing 4.1: AS paths to an anycast endpoint on the routing table of the PE router

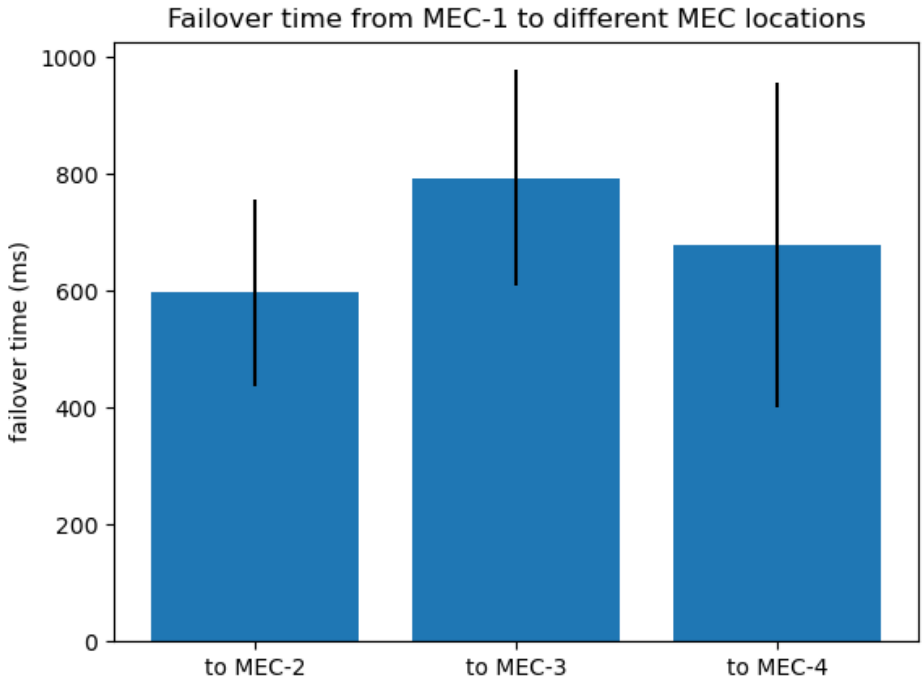

Figure 4.4: Failover time to different MEC locations

4.3.4 Application Instances Failure

When an application fails, the agent fails to probe the heartbeat of the application, and patches the `Service` type from `LoadBalancer` type to `ClusterIP` type. In this way, the MetalLB BGP speaker stops advertising the VIP address, and the new route advertisement will come from the backbone network's PE router. During the transition time, the clients may experience "[Errno 101] Network is unreachable" error from the border router. This is because when the new route has not arrived yet from the PE router, the border router replies ICMP packet type 3 ("destination is unreachable") to clients to inform that the IP address of the MEC application instance is not in the routing table. Figure 4.5 shows how the client perceives the latency when the nearest MEC application `Pod` fails, and how the latency is restored once the failed instance went up.

The latency perceived by a client shown in Figure 4.5 is the same as the latency shown in Table 4.2. When the nearest MEC application instance fails, the client's traffic goes to the second nearest MEC application instance. As

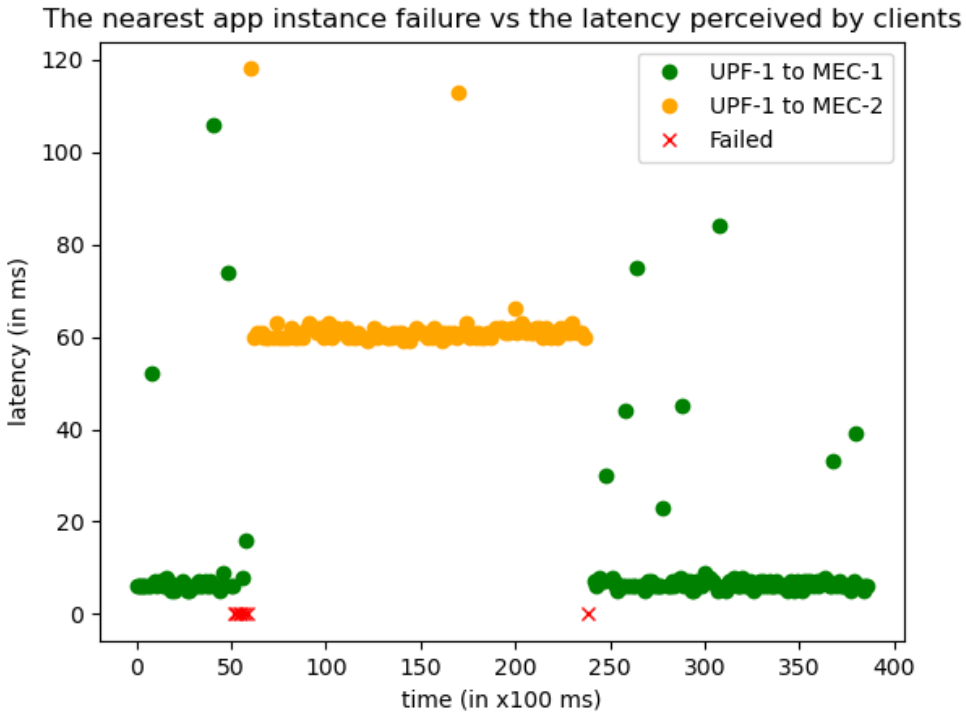

Figure 4.5: The effect of latency perceived by the client when the nearest MEC application instance fails

the topology shown in Figure 4.1, there are six hops before the traffic reaches the MEC hosts in the second nearest MEC. In the backbone network, there is 6 millisecond additional delay per hop. As a result, when the application instance in the MEC-1 fails, and the traffic goes to the MEC-2 instead, the client experiences an additional 54.26 millisecond in latency on average, and 2.34% failed requests. The average duration of the additional latency perceived by clients will be as long as the MTTR of `Pod` failure, which is 17138.0 millisecond. The failure is due to the "destination is unreachable" error returned by ICMP when the route advertisement has not arrived yet on the MEC's border router. This disproves the hypothesis by deviating 54.26 milliseconds in the latency and an increasing number of errors as large as 2.34%.

4.3.5 MEC Hosts Failure

The consequence of MEC host failure vary depending on different situations. When a node failure happens, yet there is a redundant `Pod` running in another healthy node, the border router will evict the route to the failed node, and the traffic will be routed to the `Pod` in the healthy node. However, if all nodes hosting the `Pod`s fail, the traffic must be redirected to the next nearest MEC as a failover. The `srvbend` agent is not useful if it is also down along with the node. In this case, the BFD plays the primary role in telling the border router about the failure. Figure 4.6 shows how the client perceives the latency when the node fails and never goes up again. Unlike the application instance failure scenario where the MTTR approximation is possible, hosts failure's MTTR is more challenging to estimate since this thesis does not rely on any bare metal management framework. In this case, a host failure must be handled and mitigated manually.

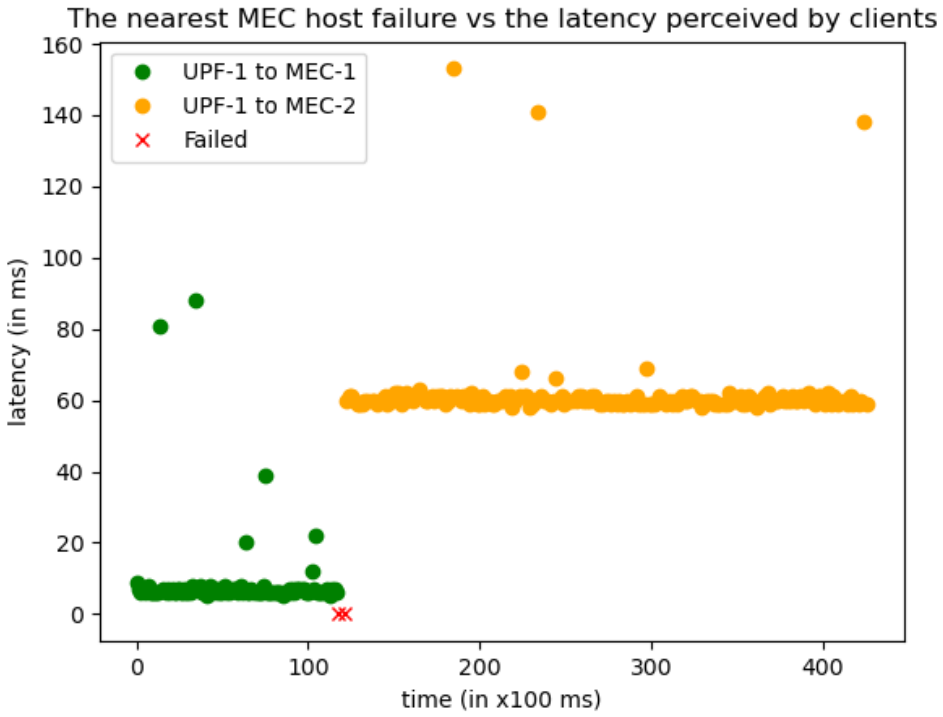

Figure 4.6: The effect of latency perceived by the client when the nearest MEC node fails

In this scenario, what clients perceived is almost the same as the previous

scenario when the `Pod` fails. The only difference is in the first scenario, the resiliency is managed by the Kubernetes scheduler, while MEC hosts failure is not managed.

4.3.6 HTTP Flooding DoS Attack Mitigation

This experiment investigated whether this traffic failover solution was suitable for mitigating flooding DoS attacks by switching the traffic to another healthy MEC. This experiment instantiated the Steady State as a client sending a constant requests rate of 5 Requests Per Second (RPS). To introduce an anomaly event **e1**, there was one malicious user who attacked the MEC application instance by saturating its computation resource with DoS attack. The attacker sent a continuous 2000 RPS while the client still did its regular operation of sending five RPS. The experiment was run for two minutes. This experiment was limited to the HTTP flooding DoS attack only. We then measured the latency and success rate of legitimate traffic to see if the Hypothesis still held.

Table 4.3 shows the measurement of the quality of service perceived by a client in several different conditions. Under a normal condition, the MEC application can serve the client within 6 milliseconds without any error request. However, under a DoS attack, the quality of the service drops significantly.

	No attack	**Attacked, no mitigation**	**CPU load-based failover**	**Rate-limiter, 5 RPS**
Median latency (ms)	6	15000	49	8
Failed requests (percent)	0	65.79	82.95	0

Table 4.3: The client's perceived quality of service during normal and application-attacked conditions

When undergoing a flooding attack, the median latency perceived by the client is 15000 milliseconds, with 65.79% of `504 gateway timeout` error requests. This shows how a simple flooding attack can overwhelm the MEC application and disturbs the availability of the service. The MEC application cannot handle the requests because under an attack, the CPU usage saturates and cannot process incoming requests.

The traffic failover mechanism, which shifts the traffic from the client to the next nearest healthy MEC, mitigates this problem by examining the

CPU usage continuously every 3 seconds. If, after 15 milliseconds, the usage is 100%, the traffic is shifted to the next nearest MEC to distribute the load. However, the result does not show a satisfying outcome. While the median latency improves significantly, the number of failed requests deteriorates from 65.79% to 82.95%.

The reasons behind this result are the following. First, if the second nearest MEC is as powerful as the first victim, shifting the traffic to that MEC will not improve anything. Second, while the traffic shifts, there is a possibility that the request fails due to "destination unreachable" as the previous experiments already explained. Third, the system does not differentiate attack traffic to the legitimate traffic. Hence, the effort to tame the attack by failing the attacker's traffic will affect the legitimate traffic too.

A rate limit mitigates this type of attack successfully. The rate limit is implemented as an `Nginx` proxy acting as the Kubernetes `ingress controller`. Table 4.3 shows how successful the rate limit in mitigating HTTP flooding attack is. With the limit of 5 RPS set on the rate limiter, the client perceives the median latency of 8 millisecond without any error request. The result disproves the hypothesis by only 2 millisecond additional latency from the steady state. This is because the rate limiter used the Leaky-Bucket algorithm to hold the incoming traffic and returned HTTP 503 if the bucket is full. Since the traffic coming to the application is limited, the CPU usage of the application does not reach the maximum threshold of CPU usage. Figure 4.7 shows that during the experiment, the CPU usage does not reach the maximum threshold set.

From this experiment, we can see that there is no magic number for the rate limit that can work nicely with all cases. Instead, an experiment before deploying the application may help to determine the appropriate rate limit to set. The use of The Utilization Saturation and Errors (USE) Method [31] may help in finding the saturation in the utilization to conclude an appropriate maximum rate limit set.

The use of rate limiter is only one of the approaches to mitigate DoS attacks. Other techniques might be combined to build a more secure system, such as inspecting `User-Agent` header of HTTP, blocking the IP address of the attacker automatically, and so on.

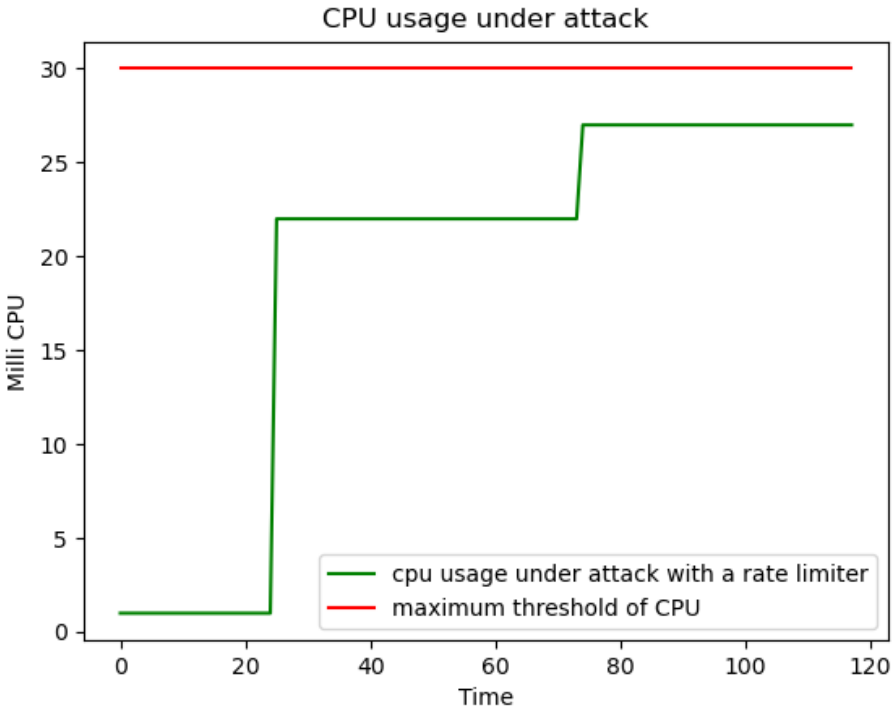

Figure 4.7: CPU usage under an HTTP flooding attack (maximum of 30 milli CPU)

Chapter 5

Discussion

This chapter discusses the implication of the proposed solution to the availability as the answer of the research questions defined in the Section 1.1. In addition, it also discusses the security considerations of the proposed solution.

5.1 MEC Availability Enhancement

Availability is defined as the ratio of the uptime to the sum of uptime and the downtime, which makes a value between 0 and 1 [47]. In other words, availability is the probability that a system runs as defined by the requirement or Service Level Agreement (SLA). The uptime is defined as the average operation time between any two failure events, known as Mean Time Between Failure (MTBF). MTBF is a term used for a repairable system, while Mean Time To Failure (MTTF) is used for a non-repairable system [38]. We consider MTBF since the MEC platform will repair the failure automatically. The downtime is the average duration of the failure, known as MTTR. Equation 5.1 shows how to calculate the availability.

$$Availability = \frac{MTBF}{MTBF + MTTR} \tag{5.1}$$

RQ1 questions the mechanism to help clients connect to the nearest MEC application instances. This thesis answers **RQ1** by proposing a MEC system that leverages IP anycast. With the topological setting shown in Figure 4.1, clients connected to UPF-1 connect to the application instance in the MEC-1. If that application fails, those clients will connect to MEC-2, which is the next nearest instance, with the failover time shown in Figure 4.4. Compared to GeoDNS, the proposed system does not need to infer the clients' locations,

thus it avoids requiring information that might be sensitive. The drawback is that it is more complicated to deploy since it relies on the network infrastructure at a large scale to make routes between MECs. Another significant advantage is that it has a better failover time, which we leverage to enhance the availability.

RQ2 challenges MEC to enhance the applications' availability on the infrastructure level. The proposed solution improves the availability by shortening the repair time and extending the time between failure. Since the $MTTR$ is smaller and the $MTBF$ is larger, according to the Equation 5.1, the *Availability* is increased. Without the proposed solution, the MTTR of `Pod` deletion is 16.505 seconds. Meanwhile, with the proposed solution, the failover time (MTTR) is less than 1 second, as shown in Figure 4.4. The equation below shows how to find the availability improvement that the solution offers.

$$\begin{aligned} \textit{Availability improvement} &= A_{\textit{with solution}} - A_{\textit{no solution}} \\ &= A' - A \\ &= \frac{MTBF'}{MTBF' + MTTR'} - \frac{MTBF}{MTBF + MTTR} \end{aligned}$$

The $MTTR$ before applying the solution is not always predictable. While the $MTTR$ for a `Pod` deletion time is predictable, on the vanilla bare metal environment, MEC hosts failure duration is difficult to predict because some physical failures may take longer to repair. In both failure cases, the solution still works, and the failover result is not dependent on neither failure cases. Therefore, the availability improvement will seem much higher on a system with a high $MTTR$ value before the solution applied.

The availability of the system defines the QoS offered by the application owner. For example, while an increment of 0.5% in availability does not seem significant, the failure time of 800 milliseconds versus 17000 millisecond per failure event is significant in terms of user experience. For time-critical applications such as Industrial IoT and vehicular communications, the failure time difference matters.

5.2 Security Considerations

This section discusses the security aspects that need to be considered following the design decision this thesis took. This thesis leverages the distributed deployment of applications and network layer techniques, especially IP any-

cast, to build a highly available MEC platform. These two properties that the system relies on have several inherent security implications.

IP anycast localizes an area that applications serve because traffic from a particular source always goes to the nearest destination relative to the source's location. This property has an advantage for mitigating DoS attack to localize the attack surface. However, this is vulnerable to route hijacking since any entity on the backbone network can also advertise the anycast address. Last but not least, localized applications open up possibilities to reveal clients' locations, which is privacy sensitive.

5.2.1 Denial-of-Service Attack Mitigation

An anycast MEC can serve as a sink to a DoS attack [36]. If the users are highly distributed across different areas, a DoS attack from a single area only affects the availability of the nearest MEC corresponding to the attacker's location, and the attack does not impact the other users connected to different MEC. It also has a better scaling property since it is relatively simple to add more anycast instances in different regions. One aspect that must be carefully considered is that when using anycast, MEC instances should not withdraw VIP addresses during an ongoing DoS attack as it can bring the attack to another healthy anycast instance [20].

We experimented with the availability of the MEC applications by attacking them with an HTTP flooding attack with a rate-limiter as the first layer of protection. Although rate-limiter was effective in mitigating the flooding attack, MECs should also consider other types of attacks, such as the SYN flood attack, smurf attack, and Slowloris attack. For example, the proxy may use Transmission Control Protocol (TCP) SYN cookies to mitigate SYN flood attack and connection timeout as well as maximum connections limiter to mitigate the Slowloris attack, which implementation is widely available with existing technologies, such as with a particular Linux setting for TCP SYN cookies and HAProxy [12]. The border router can also filter the packet, such as with Linux's `netfilter`, to drop incoming ICMP packets with broadcast address as their destination to mitigate the smurf attack [22].

5.2.2 Route and Service Hijacking

Malicious nodes can advertise the anycast VIP address on the network so that the route to the malicious nodes will be installed on the ISP backbone routers [36]. This attack is similar to the BGP hijacking attack to a certain extent. The attacker can absorb the traffic from the clients to impersonate

the legitimate MEC application instances, tamper the packets, or drop the packets (blackhole attack).

Since the path to the MEC instances across the backbone router is determined by the AS path of the BGP, the BGP advertisement should be authenticated and integrity protected using the mechanism as specified in `RFC 2385` [33]. Besides, the application can also be enhanced with an end-to-end security protocol such as TLS and IPSec to improve the security in terms of confidentiality, integrity, and authentication.

5.2.3 Privacy Issues

Since a MEC serves a specific geographical area, clients' related logs in MEC can be used to infer the clients' location, movements, and activities. This logging activity may lead to privacy-related issues. Well-known network layer techniques to work around this problem, such as using onion routing or independent proxies [60], might not be suitable since it may route the traffic farther than it should be, eliminating the benefits of MEC. One of the most effective ways to limit this privacy issue is a law enforcement that ISPs or MNOs must abide so that they cannot log and reveal clients' activities and locations [60].

Chapter 6

Conclusion and Future Work

This chapter summarizes the contributions of this thesis, outlines the possible impacts given from the findings of the experiment, and discusses the direction for future work.

6.1 Conclusion

This thesis aims to study, suggest, and analyze the availability-enhanced MEC platform with traffic failover. This thesis addresses two problems: finding the nearest MEC application instance and mitigating failures of MEC instances. We have designed, built a proof of concept, and run the experiment to evaluate the performances.

We found that the IP anycast addressing strategy was effective in connecting the clients in the mobile network to the nearest MEC instance. Anycast also extends the possibility of a faster failover than DNS-based failover because clients do not have to wait for the cache to expire to reach the backup instance. Anycast also makes the failover transparent to clients. It also does not rely on DNS ECS, which is recommended by the RFC 7871 [16]. However, the IP anycast relies on the infrastructure at the ISP level. MEC owners must set up a dedicated MEC network with MPLS, for instance.

MEC system mitigates the failure by connecting clients to the next nearest healthy MEC instance. IP anycast allows clients to find the next nearest instance via the routing protocol's best path. Each application instance has a VIP address that is advertised by a BGP speaker to the public network. An agent is responsible for checking the heartbeat of the MEC application instance. If the agent finds that the application fails, it stops the BGP speaker from advertising the failed application instance's VIP address. We also used BFD to check the links and nodes connectivity, allowing a quick link or node

failure detection.

With the proposed solution, we found that the failover time was less than one second on average, and the client perceived an increase in latency and minor failed requests if the nearest MEC fails. The latency increase grows linearly as the latency to the next nearest MEC grows. This result shows a significant improvement in the availability percentage of the system.

We also found that CPU usage-based failover was too aggressive and did not differentiate between legitimate and destructing traffic. Therefore, we found the CPU usage-based failover not practical to mitigate a massive traffic spike, such as when undergoing an HTTP flooding attack. Instead of CPU usage-based, we recommend the use of rate-limiting inside the MEC. Rate limiting was proven to be more effective in mitigating this problem with a minor decrease in the quality of the service perceived by clients.

6.2 Future Work

This thesis builds a foundation for a highly available MEC platform. We suggest the following future work to improve and extend a more reliable MEC platform.

Instead of LDP for distributing MPLS labels, more scalable protocols and sophisticated technologies, such as Resource Reservation Protocol-Traffic Engineering (RSVP-TE) or Software-Defined Networking in a Wide Area Network (SD-WAN), are recommended. With these technologies, operators can have a more flexible and customizable network behavior to connect multiple MECs.

The Kubernetes set up in this thesis is only responsible for a single MEC. Each of them runs independently, and it limits the view of the overall orchestration for multiple MECs. We suggest the use of `Kubernetes Cluster Federation` to orchestrate multiple MECs in a more scalable way. This approach allows operators to have more capabilities, such as provisioning MECs automatically and purging MEC application automatically if there is no client within that area.

Bibliography

[1] Airframe open edge server — nokia. https://www.nokia.com/networks/products/airframe-open-edge-server/. (Accessed on 03/10/2020).

[2] Frequently asked questions — public dns — google developers. https://developers.google.com/speed/public-dns/faq. (Accessed on 02/05/2020).

[3] K3s: Lightweight kubernetes. https://k3s.io/. (Accessed on 03/09/2020).

[4] Kubeedge. https://kubeedge.io/en/. (Accessed on 03/09/2020).

[5] Metallb, bare metal load-balancer for kubernetes. https://metallb.universe.tf/. (Accessed on 03/09/2020).

[6] Sensing and analytics of air quality — university of helsinki. https://www.helsinki.fi/en/researchgroups/sensing-and-analytics-of-air-quality. (Accessed on 02/13/2020).

[7] About megasense — sensing and analytics of air quality — university of helsinki. https://www.helsinki.fi/en/researchgroups/sensing-and-analytics-of-air-quality/about-megasense, 6 2019. (Accessed on 02/13/2020).

[8] 3GPP. Study on enhancement of support for Edge Computing in the 5G Core network (5GC). Technical Report (TR) 23.748, 3rd Generation Partnership Project (3GPP), 12 2019. Version 0.2.0.

[9] 3GPP. System architecture for the 5G System (5GS); Stage 2. Technical Specification (TS) 23.501, 3rd Generation Partnership Project (3GPP), 12 2019. Version 16.3.0.

[10] Abts, D., and Felderman, B. A guided tour of data-center networking. *Communications of the ACM 55*, 6 (2012), 44–51.

[11] Andrew S. Tanenbaum, D. J. W. *Computer networks*, 5ed. ed. Pearson Prentice Hall, 2011.

[12] Assmann, B. Use a load balancer as a first row of defense against ddos - haproxy technologies. `https://www.haproxy.com/blog/use-a-load-balancer-as-a-first-row-of-defense-against-ddos/`, 2 2012. (Accessed on 06/09/2020).

[13] Azure, M. Overview of the resiliency pillar - azure architecture center — microsoft docs. `https://docs.microsoft.com/en-us/azure/architecture/framework/resiliency/overview`, 10 2019. (Accessed on 04/13/2020).

[14] Bonaventure, O., et al. *Computer Networking: Principles, Protocols and Practice*. Citeseer, 2011.

[15] Brodkin, J. Senate votes to let isps sell your web browsing history to advertisers — ars technica. `https://arstechnica.com/tech-policy/2017/03/senate-votes-to-let-isps-sell-your-web-browsing-history-to-advertisers/`, 3 2017. (Accessed on 02/05/2020).

[16] C. Contavalli, W. van der Gaast, D. L. W. K. Client subnet in dns queries. RFC 7871, RFC Editor, 5 2016.

[17] Cisco. Bgp best path selection algorithm - cisco. `https://www.cisco.com/c/en/us/support/docs/ip/border-gateway-protocol-bgp/13753-25.html`, 9 2016. (Accessed on 05/19/2020).

[18] Cloudflare. What is anycast? how does anycast work? — cloudflare. `https://www.cloudflare.com/learning/cdn/glossary/anycast-network/`. (Accessed on 05/12/2020).

[19] D. Katz, D. W. Bidirectional forwarding detection (bfd). RFC 5880, RFC Editor, 6 2010.

[20] D. McPherson, D. Oran, D. T. E. O. Architectural considerations of ip anycast. RFC 7094, RFC Editor, 1 2014.

[21] De Vries, W. B., van Rijswijk-Deij, R., de Boer, P.-T., and Pras, A. Passive observations of a large dns service: 2.5 years in the life of google. *IEEE transactions on network and service management* (2019).

[22] Documentation, N. Netfilter extensions howto: New netfilter matches. `https://netfilter.org/documentation/HOWTO/netfilter-extensions-HOWTO-3.html`. (Accessed on 06/09/2020).

[23] Dutt, D. *BGP in the Data Center*. O'Reilly Media, 2017.

[24] Eisenbud, D. E., Yi, C., Contavalli, C., Smith, C., Kononov, R., Mann-Hielscher, E., Cilingiroglu, A., Cheyney, B., Shang, W., and Hosein, J. D. Maglev: A fast and reliable software network load balancer. In *13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16)* (2016), pp. 523–535.

[25] ETSI. Multi-access Edge Computing (MEC); Edge Platform Application Enablement. Group Specification (GS) MEC.011, European Telecommunications Standards Institute (ETSI), 11 2019. Version 2.1.1.

[26] ETSI. Multi-access Edge Computing (MEC); Location API. Group Specification (GS) MEC.013, European Telecommunications Standards Institute (ETSI), 9 2019. Version 2.1.1.

[27] ETSI. Multi-access Edge Computing (MEC); Radio Network Information API. Group Specification (GS) MEC.012, European Telecommunications Standards Institute (ETSI), 12 2019. Version 2.1.1.

[28] ETSI. Multi-access Edge Computing (MEC); UE application interface. Group Specification (GS) MEC.016, European Telecommunications Standards Institute (ETSI), 4 2019. Version 2.1.1.

[29] Fabrikant, A., Syed, U., and Rexford, J. There's something about mrai: Timing diversity can exponentially worsen bgp convergence. In *2011 Proceedings IEEE INFOCOM* (2011), IEEE, pp. 2975–2983.

[30] FRRouting. Ldp ? frr latest documentation. `http://docs.frrouting.org/en/latest/ldpd.html`. (Accessed on 06/04/2020).

[31] Gregg, B. Thinking methodically about performance. *Communications of the ACM 56*, 2 (2013), 45–51.

[32] Guba, O., and Ivanov, A. Dropbox traffic infrastructure: Edge network — dropbox tech blog. `https://blogs.dropbox.com/tech/2018/10/dropbox-traffic-infrastructure-edge-network/`, 10 2018. (Accessed on 03/09/2020).

[33] Heffernan, A. Protection of bgp sessions via the tcp md5 signature option. RFC 2385, RFC Editor, 8 1998.

[34] Hevner, A. R., March, S. T., Park, J., and Ram, S. Design science in information systems research. *MIS quarterly* (2004), 75–105.

[35] Hofmann, M., and Beaumont, L. R. *Content networking: architecture, protocols, and practice*. Elsevier, 2005.

[36] J. Abley, K. L. Operation of anycast services. RFC 4786, RFC Editor, 12 2006.

[37] James, T. Y. Measuring failover time for high availability network. In *Proceedings of the International Conference on Scientific Computing (CSC)* (2018), The Steering Committee of The World Congress in Computer Science, pp. 34–39.

[38] Jens Lienig, H. B. *Fundamentals of Electronic Systems Design*. Springer, 2017.

[39] kernel.org. https://www.kernel.org/doc/documentation/networking/ip-sysctl.txt. `https://www.kernel.org/doc/Documentation/networking/ip-sysctl.txt`. (Accessed on 06/04/2020).

[40] Kintis, P., Nadji, Y., Dagon, D., Farrell, M., and Antonakakis, M. Understanding the privacy implications of ecs. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment* (2016), Springer, pp. 343–353.

[41] Lam, C. F., Liu, H., and Urata, R. What devices do data centers need? In *Optical Fiber Communication Conference* (2014), Optical Society of America, pp. M2K–5.

[42] Le Boudec, J.-Y. *Performance Evaluation of Computer and Communication Systems*. EPFL Press, Lausanne, Switzerland, 2010.

[43] Lee, J., Moon, S.-J., Bae, B., and Lee, J. Local area data network for 5g system architecture. In *2018 IEEE 5G World Forum (5GWF)* (2018), IEEE, pp. 141–146.

[44] MetalLB. Metallb in bgp mode. `https://metallb.universe.tf/concepts/bgp/`. (Accessed on 06/06/2020).

[45] Nginx. Module ngx_http_limit_req_module. `http://nginx.org/en/docs/http/ngx_http_limit_req_module.html`. (Accessed on 06/04/2020).

[46] Nginx-ingress. Annotations - nginx ingress controller. `https://kubernetes.github.io/ingress-nginx/user-guide/nginx-configuration/annotations/#rate-limiting`. (Accessed on 06/04/2020).

[47] Pressman, R. S. *Software Engineering: A Practitioner's Approach*. McGraw-Hill Education, jan 2014.

[48] Prince, M. Load balancing without load balancers. `https://blog.cloudflare.com/cloudflares-architecture-eliminating-single-p/`, March 2013. (Accessed on 05/12/2020).

[49] Prince, M. Announcing 1.1.1.1: the fastest, privacy-first consumer dns service. `https://blog.cloudflare.com/announcing-1111/`, 4 2018. (Accessed on 02/05/2020).

[50] Ren, J., Guo, H., Xu, C., and Zhang, Y. Serving at the edge: A scalable iot architecture based on transparent computing. *IEEE Network 31*, 5 (2017), 96–105.

[51] Rosenthal, C., Hochstein, L., Blohowiak, A., Jones, N., and Basiri, A. *Chaos Engineering*. O'Reilly Media, Incorporated, 2017.

[52] Sabella, D., Sukhomlinov, V., Trang, L., Kekki, S., Paglierani, P., Rossbach, R., Li, X., Fang, Y., Druta, D., Giust, F., Cominardi, L., Featherstone, W., Pike, B., and Hadad, S. Developing software for multi-access edge computing. *ETSI white paper 20* (2019), 1–38.

[53] Schlinker, B., Kim, H., Cui, T., Katz-Bassett, E., Madhyastha, H. V., Cunha, I., Quinn, J., Hasan, S., Lapukhov, P., and Zeng, H. Engineering egress with edge fabric: Steering oceans of content to the world. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (2017), pp. 418–431.

[54] Sechkova, T., Paolino, M., and Raho, D. Virtualized infrastructure managers for edge computing: Openvim and openstack comparison. In *2018 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)* (2018), IEEE, pp. 1–6.

[55] Steenbergen, R. Mpls for dummies. *Recuperado el 11* (2016).

[56] Taleb, T., Samdanis, K., Mada, B., Flinck, H., Dutta, S., and Sabella, D. On multi-access edge computing: A survey of the

emerging 5g network edge cloud architecture and orchestration. *IEEE Communications Surveys & Tutorials 19*, 3 (2017), 1657–1681.

[57] Trabelsi, S., Pazzaglia, J.-C., and Roudier, Y. Secure web service discovery: overcoming challenges of ubiquitous computing. In *2006 European Conference on Web Services (ECOWS'06)* (2006), IEEE, pp. 35–43.

[58] Weiden, F., and Frost, P. Anycast as a load balancing feature. In *LISA* (2010), vol. 10, pp. 1–6.

[59] Y. Rekhter., T. Li, S. H. A border gateway protocol 4 (bgp-4). RFC 4271, RFC Editor, 1 2006.

[60] Yanes, A. Privacy and anonymity. *arXiv preprint arXiv:1407.0423* (2014).

[61] Yap, K.-K., Motiwala, M., Rahe, J., Padgett, S., Holliman, M., Baldus, G., Hines, M., Kim, T., Narayanan, A., Jain, A., et al. Taking the edge off with espresso: Scale, reliability and programmability for global internet peering. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (2017), pp. 432–445.