Cooperative Load Balancing of Neighboring Edge Clouds with LISP

Toru Furusawa*† , and Akihiro Nakao†
*Toyota Motor Corporation, Tokyo, Japan
†The University of Tokyo, Tokyo, Japan

***Abstract*—Cooperative load balancing effectively prevents service quality deterioration, particularly under elevated loads in edge offloading services of resource-limited edge clouds. It does this by transiently dispersing the load across multiple geographically proximate edge clouds. Our study introduces a method that enables quick load redistribution among adjacent edge clouds, facilitating rapid load distribution through the dynamic redirection of requests from terminals based on the load status of the respective edge cloud. This is achieved by updating the route information within the relay network. To validate our approach, we developed a prototype of an edge cloud infrastructure using Kubernetes and a controller responsible for routing control between edge clouds utilizing Locator/Identity Separation Protocol (LISP). Through experimentation, we demonstrate an 80% reduction in median response time when distributing the load across neighboring edge clouds.**

***Index Terms*—edge computing, container, Kubernetes, LISP, load balancing**

I. Introduction

Numerous studies are underway to explore the practical implementation of edge computing, which is characterized as a processing approach that takes place on servers strategically positioned between the cloud infrastructure and connected devices [1]. This emerging paradigm facilitates low-latency responses and mitigates relay traffic. An exmemplary use case that highlights the potential of edge computing is found in connected cars, which demand not only substantial data transmission and swift processing but also rapid processing capabilities [2]. One specific area of focus within this research is computation offloading. This involves the continuous transmission of camera images and sensor data from onboard devices to a server for processing, emphasiziing the need for extensive data transmission and minimal latency in response times [3].

The infrastructure supporting edge computing, enabling the dynamic allocation of server resources through virtualization technology akin to traditional large-scale clouds, is referred to as an edge cloud. Notably, the implementation of a microservices architecture (MSA) with container virtualization technology and Kubernetes[1] on the edge cloud is anticipated to yield high scalability and agility. This mirrors the capabilities of traditional clouds, operating at the microservices level, where each unit provides distinct functions [4].

However, owing to the dispersed nature of edge clouds across diverse locations, the computing resources at their disposal are inherently more limited when compared with those present in public clouds or on-premises environments in extensive data centers. In addressing the challenges associated with the limited computing resources of geographically dispersed edge clouds, particularly as their distribution widens, the risk of overload becomes more pronounced. This is especially critical in edge offloading scenarios where continuous data uploadsand low-latency responses are paramount.

To mitigate this risk, we are developing a method for cooperative load balancing among neighboring edge clouds for edge offloading services [5]. This collaborative approach involves multiple geographically proximate edge clouds and public clouds. If a specific edge cloud, operating at full computational capacity, encounters a load exceeding its stable operational threshold, the method temporarily redirects surplus requests to neighboring edge clouds or available public clouds, ensuring the stable operation of edge offloading services.

While the cooperative load balancing proposed in [5] effectively distributes the computational load of edge offloading, it presents certain challenges. First, it does not address the distribution of the network load on the edge cloud gateway, which could be problematic due to the continuous transmission of extensive data from multiple terminals potentially consuming a significant portion of the edge cloud's bandwidth. Second, data transfers between edge clouds and public clouds result in increased response times compared with methods involving direct connections between terminals and each edge cloud or public cloud.

In this study, we propose a cooperative load distribution method among neighboring edge clouds to overcome these challenges. This method dynamically updates the routing information of the network, dispersing the access destination edge cloud from the terminal. Edge edge cloud's offloading services are assigned a common IP address to which terminals send their requests. If the requests per second (RPS) for a particular edge cloud's offloading service exceeds a certain threshold, the controller utilizes the Locator/Identity Separation Protocol (LISP) [6] to dynamically alter the relevant router's mapping registry. This temporarily redirection routes excess requests to a neighboring edge cloud with available capacity.

The contributions of this study are as follows:

- By employing a method that disperses the terminal's access point to the service in the relay network, we enable

[1]https://kubernetes.io/

979-8-3503-7099-7/24/$31.00 ©2024 IEEE

the distribution of both computational and network loads in the cooperative load distribution of edge offloading services among neighboring edge clouds.
- Through evaluation experiments, we demonstrate that, during periods of overload on a specific edge cloud's offloading service, the median response time of terminal requests is reduced by up to 80% compared to methods that do not employ our proposed approach.

The rest of this paper is structured as follows. The following section explores related research. The proposed method along with its implementation are detailed in Sections III and IV, respectively. Section V presents the evaluation of the proposed method through experiments. Finally, we draw our conclusions in VI.

II. Related Work

In edge and fog computing, individual edge or fog environments are characterized by limited available resources. Consequently, in scenarios where a substantial load is present, coordinated task offloading and load distribution among multiple environments become crucial to ensure the continuous provision of stable services [7]. Cooperative load balancing among neighboring edge server resources is a recognized approach to mitigate edge server overload [8]. In this cooperative load balancing framework, multiple edge servers in close geographic proximity collaborate. When the number of requests to a particular edge server surpasses a predetermined threshold, the excess requests are temporarily redirected to neighboring edge servers, thereby maintaining stable the edge server processing. Previous research [9] has proposed collaborative resource management and load distribution methods for multiple fog computing environments, primarily focusing on mathematical models for achieving resource management and load distribution. However, for the realization of load distribution between geographically dispersed edge computing environments connected via wide area networks, methods encompassing the network layer are imperative.

Compute-first networking (CFN) and compute aware networking (CAN) are methodologies that disperse the terminal's access destination by dynamically determining the routing destination of the network based on real-time computational and network loads of edge clouds [10]. Similarly, the method proposed in [11] disperses the terminal's connecting edge cloud by exchanging BGP control messages between relay routers, incorporating both computational and network load metrics. Another approach presented in [12] determines the terminal's access destination edge cloud based on each edge cloud's computational load, simultaneously allocating the necessary bandwidth to the route connecting the terminal to the access destination edge cloud. In contrast to these methods, the approach presented in this study distinguishes itself by implementating LISP-compliant routers at the gateway connecting edge clouds and terminal groups.

A related method proposed in [13] utilizes LISP to distribute access from terminals to the same service deployed across multiple edge clouds, deploying services with the same IP address on Kubernetes clusters on edge clouds and the determining of the connecting edge cloud from terminals using LISP. While sharing similarities in these aspects, our study stands out by introducing a method that dynamically switches the destination edge cloud, presenting a unique contribution.

III. Proposed Method

In this section, we discuss the proposed cooperative load distribution method for edge offloading services among neighboring edge clouds.

A. System Overview

The proposed system is illustrated in Figure 1. This system operates instances of the same edge offloading service, accessible via a shared IP address, across all geographically distributed edge clouds. The edge offloading service, a stateless service, is designed to sequentially receive vehicle data from the on-board terminal via the mobile network, conduct computational processing such as inference, and promptly return the results to the terminal. A typical service instance performs sequential object detection processing on images transmitted at regular intervals from the on-board terminal's camera, promptly sending the results back to the terminal.

XTRs, which is a LISP-compliant routers, are strategically placed at the gateway router (referred to as the client gateway) connecting terminals in each region and at the access point (referred to as the edge gateway) of each edge cloud from the outside. LISP, an encapsulation protocol, separates and manages device identification (ID) information from location indication (Locator) information. Within routers that support LISP, there are two primary types: Ingress Tunnel Router (ITR) and Egress Tunnel Router (ETR). The ITR encapsulates packets from a LISP site and sends them to the destination LISP router's IP address, while the ETR receives encapsulated packets and sends the decapsulated packets to the LISP site. Generally, both ITR and ETR functionalities are implemented on the same router, commonly referred to as an XTR.

The client gateway is positioned after the UPF of the 5G mobile network. Each terminal periodically sends a request to the common IP address of the edge offloading service, and the intermediate client gateway forwards the request from the terminal to one of the edge clouds. When the load is light, each terminal's request is forwarded to the geographically closest edge cloud's offloading service. However, with an increase in the edge cloud's load and an anticipation of potential service degradation owing to longer response times or service interruptions from server overload, the client gateway is dynamically managed to distribute a portion of requests to neighboring edge clouds that are not overloaded. The controller determines the allocation of requests to neighboring edge clouds by assessing the load on each edge cloud's offloading service and regulary monitoring resource availability. This information is subsequently recorded in the mapping registry of the client gateway.

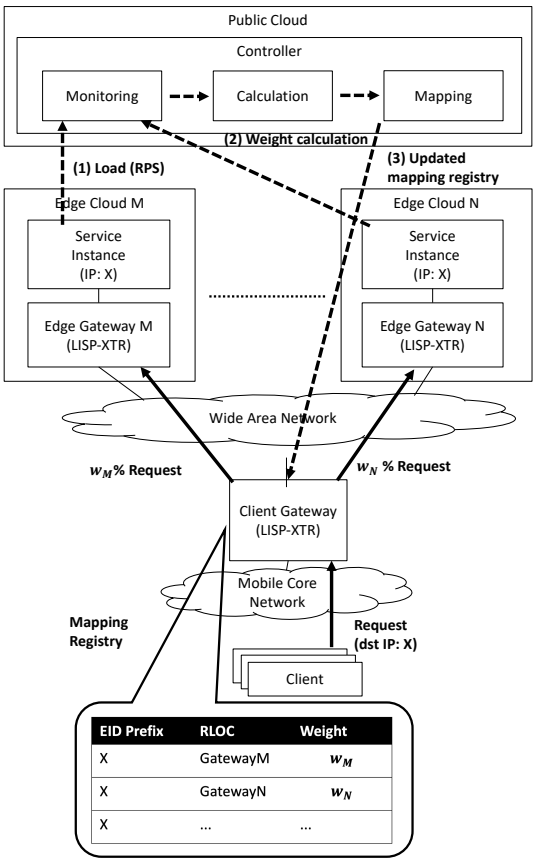


Fig. 1. Overview of proposed architecture

B. Controller Operations

Table I presents a compilation of data models utilized by the controller for monitoring, static configuration, and dynamic control tasks. The edge clouds to which the client gateway connects are defined as $edge_i$, where i is the index of the edge cloud assigned based on the prioirity level, with lower values indicating higher priority. Those priorities are determined by factors such as geographic distance and the extent of communication delay. For each edge cloud $edge_i$, the maximum RPS of requests for stable operation of the edge offloading service is assumed to be pre-measured and defined as C_i. The controller repetitively executes the following three operations at intervals of $interval$ seconds:

1) Acquire the RPS of requests to the client gateway (defined as r).
2) Calculate the weight values for the gateway routers using Algorithm 1.
3) Register the calculated transfer ratio in the mapping resolver and promptly update the mapping registry of the client gateway.

C. Weight Calculation Algorithm

The weight calculation algorithm is described in Algorithm 1. If the RPS directed toward $edge_1$ exceeds the maximum receivable RPS C_1, then w_1 is calculated, such that the RPS for requests to $edge_1$ becomes C_1. Subsequently, for the remaining requests $r_{left} = r - C_1$, w_2 is calculated to distribute r_{left} to $edge_2$, providing it fits within C_2. Similarly, w_3, w_4, ... are calculated for $edge_3$, $edge_4$, ... and so forth. If $r_{left} > 0$ after distributing the RPS to all edge clouds $edge_i$ in accordance with C_i, then w_1 is recalculated to distribute the remaining r_{left} to the highest-priority edge, $edge_1$.

TABLE I
CONTROLLER UTILIZED METRICS

Type	Metrics	Description
Static setting	$interval$ [seconds]	The time interval at which the controller retrieves monitoring-type metrics and updates the mapping registry.
Static setting	C_i [RPS]	The maximum RPS of requests that the edge offloading service on edge cloud $edge_i$ can process stably.
Monitoring	r [RPS]	The RPS of requests sent to the client gateway for the edge offloading service from clients.
Dynamic control	$weight_i$	The proportion of requests from terminals that reach the client gateway and are forwarded to the edge offloading service on edge cloud $edge_i$

Algorithm 1 Weight calculation algorithm

Input: $r, edges$ (a list of $edge_i$)
Output: $weights$ (a list of $weight_i$)

```
r_left = r
for edge_i in edges do
    if r_left > C_i then
        weight_i = C_i / r
        r_left = r_left − C_i
    else
        weight_i = r_left / r
        r_left = 0
        break
    end if
end for
if r_left > 0 then
    weight_1 = weight_1 + r_left / r
end if
return weights
```

D. Updating the Mapping Registry

Upon the controller registering a new transfer ratio in the mapping resolver, the updating process of the client gateway's mapping registry ensues, adhering to the sequence delineated in Figure 2. This prompt update is crucial for initiating load balancing at the client gateway. In standard scenarios, the

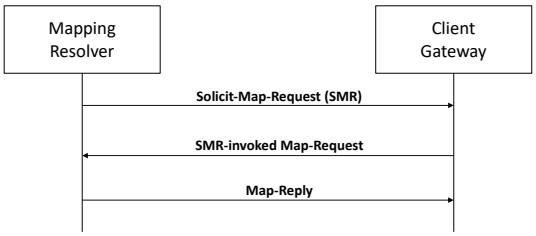


Fig. 2. Mapping Registry Update Sequence

mapping registry of the LISP's XTR is undergoes updating when the XTR submits a query to the map resolver promptly if no entry corresponding to the packet is found in the mapping registry. The update is completed upon receiving the response. Entries recorded in the mapping registry are cached for a designated period, and on condition that the cache persists, packets are continuously forwarded according to the entries stored in the mapping registry. In this study, a methodology involving the transmission of a solicit map request (SMR) message from the mapping resolver to the XTR is implemented to accelerate the update of the client gateway's the mapping registry immediately upon the controller computing the transfer ratio.

IV. Implementation

This section provides an overview of the software employed in the implementation of this study.

Table II provides a comprehensive list of the software along with their respective versions employed in this study. Each edge cloud is established as an independent Kubernetes cluster. For measuring the RPS of the edge offloading service, we use Prometheus[2]. Centralized monitoring is achieved by deploying Prometheus agents on each Kubernetes cluster and utilizing a Prometheus insntance with the Federation feature enabled in the controller,

Vector Packet Processor (VPP)[3] is employed for both the client and edge gateways. VPP is a software providing network packet forwarding functionality on Linux. It features LISP functionality as a plugin and serves as the data plane of XTR.

OpenDaylight[4] is used for the mapping resolver. While OpenDaylight is recognized as an SDN controller, it also features LISP mapping resolver functionality as a plugin. In this study, we exclusively leverage this plugin functionality to utilize Opendaylight as a mapping resolver.

The weight calculation function of the controller is implemented through a custom application using Python3. By integrating multiple open-source software, the newly developed software components in the implementation system are confined solely to the weight calculation function.

V. Evaluation

In this section, we establish an environment to simulate multiple geographically dispersed edge cloud environments and deploy a system implementing the proposed method. We incrementally escalate the rate of RPS from terminals to the edge offloading service until the RPS reaches a threshold at which the nearby edge cloud can no longer maintain stable operation. Our verification process confirms that, under such circumstances, requests are temporarily redistributed to other nearby edge clouds, effectively preventing performance degradation caused by overload.

[2]https://prometheus.io/

[3]https://s3-docs.fd.io/vpp/23.02/

[4]https://www.opendaylight.org/

TABLE II
Software Used for Implementation

Component	Software	OSS Version
Edge Cloud Infrastructure	Kubernetes	1.26.0
Monitoring	Prometheus	1.0.0
LISP XTR	VPP	22.10
LISP Mapping Resolver	OpenDaylight	17.0.0
Weight Calculation	Python3 Application	3.10.9

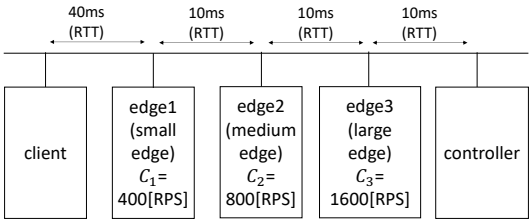


Fig. 3. Latency Settings Among Experiment Servers

A. Experimental Configuration

On an OpenStack infrastructure that we have constructed, we operate virtual machines to simulate multiple edge clouds, a controller, and a group of clients, as depicted in Figure 3. All virtual machines run on Linux (Ubuntu 20.04) as the OS. Additionally, we utilize the tc command to intentionally introduce a fixed delay between each virtual machine, as illustrated in Figure 3. Edge1 represents a small edge cloud with wide distribution, directly accessible by terminals via a wireless access line. We establish a round-trip delay of 40 ms, considering the delay value of the wireless section. Edge2 is a medium-sized edge cloud covering a broader area than Edge1, with a round-trip delay set at 50 ms with the terminal. Edge3, a large edge cloud encompassing an even larger area, has a terminal delay set to 30 ms. The communication delay of the wireless access network can in reality fluctuate significantly. However, in this study, we prioritize evaluating the feasibility of implementing the proposed method and assign a fixed round-trip delay for consistency.

The edge offloading service employed in the experiment utilizes a simple Nginx web server that only responds with a 4KB HTML page. Each Kubernetes cluster operates a single container (Pod) for the edge offloading service. The resource

TABLE III
ALLOCATED RESOURCES AND PARAMETER SETTINGS

Edge Name	Allocated Resources for Edge Offloading Service Pod	Controller's Parameter C_i
Edge1	100m CPU, 2GB Memory, 20GB Disk Capacity	400
Edge2	200m CPU, 4GB Memory, 20GB Disk Capacity	800
Edge3	400m CPU, 8GB Memory, 20GB Disk Capacity	1600

request and limit values assigned to the Pod, along with the C_i values used by the controller for weight calculation, are allocated varying values for each edge cloud, as detailed in Table III. Note that for a Pod with 100m CPU and 2GB memory, the maximum throughput of the web server utilized in this experiment is 500 transactions per second (TPS). According to the pre-verification process, if it receives more than 500 RPS simultaneously, the response time will gradually deteriorate, leading to occurence of errors.

B. Experimental Procedure

From the virtual machine client, simulating a group of clients, we initiate requests to the edge offloading service at 1 s intervals, gradually increasing the number of concurrent requests over time. Specifically, we modulate the RPS from the terminal to the edge offloading service by adjusting the number of simultaneous connections from the client, as depicted in Figure 4. Subsequently, we conduct an experiment to measure variations in the throughput with TPS of the edge offloading service and the response time of the requests as the RPS fluctuates. In a real-world scenario, the RPS of requests to edge offloading services in a certain region may change based on the road traffic volume of connected vehicles in that region. For simplicity, this experiment assumes a linear change in RPS, as illustrated in Figure 4.

In all cases, the controller's parameter values, $interval$, is set to 10 s, and n is set to 3. We also conduct experiments with a method that does not distribute the load to the nearby edge for comparative analysis.

C. Experimental Results

Figure 5 illustrates the variations in the throughput of Edge1 when load distribution to the nearby edge cloud is not implemented, along with the changes in the throughput of Edge1, Edge2, and Edge3 when employing the proposed method. Additionally, Figure 6 presents the box plot of the request response time from the client, and Table IV provides the corresponding statistical values.

In the absence of load balancing, as evident from Figure 5(a), although the client's requests continue to escalete beyond 150 s from the start of the experiment, the throughput of Edge1 remains approximately at 500 RPS. As indicated in

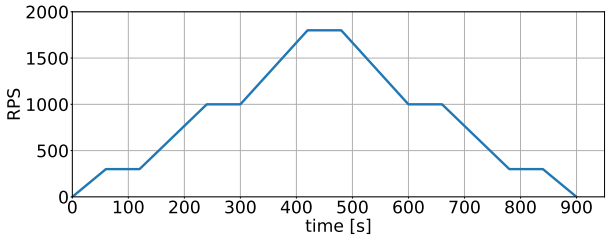
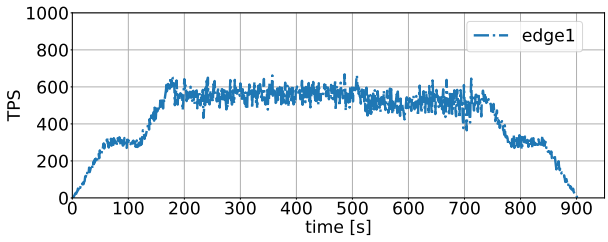
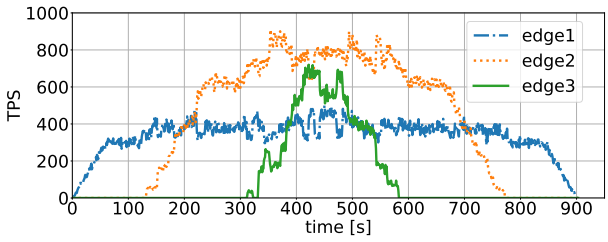


Fig. 4. Requests from Client



(a) Without Load Balancing



(b) Proposed Method

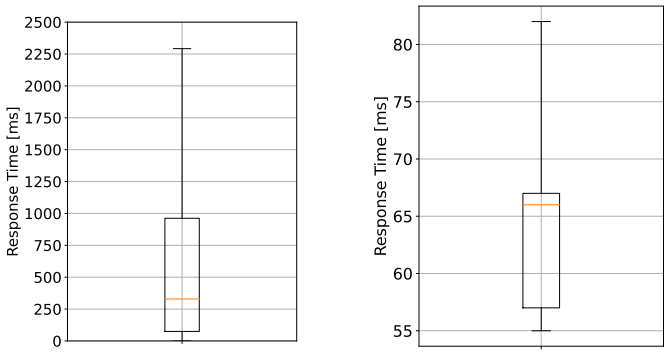
Fig. 5. Throughput

Table IV, both the median and 75th percentile values of response time have significantly increased. This is attributed to the occurrence of requests exceeding the stable management capacity of Edge1, leading to an overloaded state.

Alternatively, when implementing load balancing using the proposed method, as depicted in Figure 5(b), once the client's requests surpass the maximum throughput value of Edge1 (500 RPS), the system begins forwarding requests to Edge2. Subseqnently, immediately upon the arrival of requests exceeding the maximum throughput at Edge2, requests are further directed to Edge3. The response time for the requests remain consistently low, with the median being 1.16 times that of the 25th percentile, and the 75th percentile being only 1.18 times higher. In comparison to the scenario where load balancing is not employed, the median response time can be reduced by a substantial 80%, and the 75th percentile value can be diminished even further, by an impressive 93%.

D. Future Work

As mentioned in Section I, services utilized in edge offloading necessitate high-capacity data transmission and reception, along with low-latency responses. This study's evaluation employs a general web server as an edge offloading service. Therefore, future evaluations using services that involve the



(a) Without Load Balancing (b) Proposed Method

Fig. 6. Boxplots of Response Time

TABLE IV
RESPONSE TIME STATISTICS

Method	Proposed Method	Without Load Balancing
Mean	67.2 ms	662.1 ms
Deviation	63.4 ms	810.3 ms
Minimum	55.0 ms	55.0 ms
25th Percentile	57.0 ms	74.9 ms
Median	66.1 ms	329.2 ms
75th Percentile	67.2 ms	962.0 ms
Maximum	6361.4 ms	9385.5 ms

transmission and reception of substantial data amounts are necessary.

Furthermore, in this study, evaluations are conducted in an environment where the service load due to requests increases and decreases linearly. However, real-world load fluctuation patterns often involve non-linear changes. The expected load fluctuation patterns must be defined for each use case, and evaluations must be conducted for each fluctuation pattern.

Additionally, this study assumes an environment with a single client gateway. In the future, load balancing methods for environments with multiple client gateways will be investigated.

VI. CONCLUSION

In this study, we present a method for dynamically adjusting the mapping registry within the network using LISP XTR. This approach effectively redistributes the acess destination of each clouds from the terminal, resulting in collaborative load balancing for edge offloading services among nearby edge clouds. Our experiments illustrate that, when compared to scenarios without load balancing, this method prevents a decrease in the throughput of edge offloading services and reduces the median response time by 80%.

Our future endeavors include the following. First, we aim to enhance the load balancing method. The current study proposes a load balancing method assuming a single client gateway. However, when multiple client gateways can access the same edge cloud, a load balancing method that considers requests from each client gateway becomes necessary.

Additionally, in the proposed method, we presuppose that the maximum throughput value of the offloading service at each edge cloud is predetermined and set as a parameter (C_i). If the available resources of the edge cloud are variable, fixing the parameters may not be suitable. Therefore, a method to dynamically adjust the parameters based on the free resources and load status of the edge cloud would be beneficial.

Furthermore, as mentioned in Subsection V-D, we will continue evaluations using edge offloading services that necessitate large volumes of data and low-latency responses. Specifically, we plan to concentrate on evaluations utilizing vehicle edge offloading services that transmit vehicle camera video sequentially and conduct real-time image processing and object detection.

ACKNOWLEDGMENT

This work was partly supported by NICT, grant number JPJ012368C01101, Japan.

REFERENCES

[1] W. Shi, G. Pallis, and Z. Xu, "Edge computing [scanning the issue]," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1474–1481, 2019.
[2] Automotive Edge Computing Consortium, "General Principle and Vision White Paper Version 3.0," Available at: https://aecc.org.
[3] P. Kopelias, E. Demiridi, K. Vogiatzis, A. Skabardonis, and V. Zafiropoulou, "Connected & autonomous vehicles–environmental impacts–a review," *Science of the total environment*, vol. 712, p. 135237, 2020.
[4] M. S. Aslanpour, A. N. Toosi, C. Cicconetti, B. Javadi, P. Sbarski, D. Taibi, M. Assuncao, S. S. Gill, R. Gaire, and S. Dustdar, "Serverless edge computing: vision and challenges," in *2021 Australasian Computer Science Week Multiconference*, 2021, pp. 1–10.
[5] T. Furusawa, H. Abe, K. Okada, and A. Nakao, "Service mesh controller for cooperative load balancing among neighboring edge servers," in *2022 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*. IEEE, 2022, pp. 1–6.
[6] D. Farinacci, V. Fuller, D. Meyer, D. Lewis, and A. Cabellos-Aparicio, "The Locator/ID Separation Protocol (LISP)," RFC 9300, Oct. 2022. [Online]. Available: Availableat:https://www.rfc-editor.org/info/rfc9300
[7] G. Goel and A. K. Chaturvedi, "A systematic review of task offloading & load balancing methods in a fog computing environment: Major highlights & research areas," in *2023 3rd International Conference on Intelligent Communication and Computational Techniques (ICCT)*, 2023, pp. 1–5.
[8] R. Beraldi, A. Mtibaa, and H. Alnuweiri, "Cooperative load balancing scheme for edge computing resources," in *2017 Second International Conference on Fog and Mobile Edge Computing (FMEC)*. IEEE, 2017, pp. 94–100.
[9] N. Mostafa, "Cooperative fog communications using a multi-level load balancing," in *2019 Fourth International Conference on Fog and Mobile Edge Computing (FMEC)*, 2019, pp. 45–51.
[10] L. Tian, M. Yang, and S. Wang, "An overview of compute first networking," *International Journal of Web and Grid Services*, vol. 17, no. 2, pp. 81–97, 2021.
[11] Y. Li, Z. Han, S. Gu, G. Zhuang, and F. Li, "Dyncast: Use dynamic anycast to facilitate service semantics embedded in ip address," in *2021 IEEE 22nd International Conference on High Performance Switching and Routing (HPSR)*. IEEE, 2021, pp. 1–8.
[12] X. Han, Y. Zhao, K. Yu, X. Huang, K. Xie, and H. Wei, "Utility-optimized resource allocation in computing-aware networks," in *2021 13th International Conference on Communication Software and Networks (ICCSN)*. IEEE, 2021, pp. 199–205.
[13] K. Sun, J. Lee, and Y. Kim, "Lisp-based control plane for service connectivity in multi-cluster cloud systems," *IEEE Access*, vol. 10, pp. 24 786–24 796, 2022.