Scalable Mobile Edge Computing: A Two-tier Multi-Site Multi-Server Architecture with Autoscaling and Offloading

Ying-Dar Lin, Widhi Yahya, Chien-Ting Wang, Chi-Yu Li, and Jeans H. Tseng

Abstract

Mobile Edge Computing (MEC) provides computation resources within 5G networks hosting applications that are close to a user equipment (UE). For scalability, MEC servers can be placed behind the base stations of an access network (AN) and also inside the core network (CN) of a cellular system, which results in a two-tier architecture. A scalable MEC system reveals a management problem because keeping all servers on as traffic fluctuates wastes operational expenditure. On the other hand, traffic can become unbalanced, with hotspots in some base stations. This work proposes a two-tier multi-site multi-server architecture and integrates Latency Satisfaction Aware Autoscaling (LSAA) and Dynamic Weight Offloading (DWO) to address the above two problems. Offloading is a short-term solution to hotspot traffic, while autoscaling is a long-term solution to traffic fluctuation. A two-tier MEC testbed was implemented in the framework of OpenNESS with 3GPP integration, with experimental comparisons of one-tier vs. two-tier, uniform vs. hotspot, transient vs. persistent hotspot traffic, with or without offloading and autoscaling. Under heavy hotspot traffic, two-tier MEC satisfies 86 percent, 73 percent, and 21 percent traffic with both offloading and autoscaling, offloading only, and without offloading and autoscaling, respectively, while one-tier MEC only satisfies 32 percent, 32 percent, and 21 percent traffic.

Introduction

5G cellular communication standards are expected to fulfill various services which are categorized as Enhanced Mobile Broadband (eMBB), Ultra-reliable and Low-latency Communications (uRLLC), and Massive Machine Type Communications (mMTC). Mobile Edge Computing (MEC) provides computation resources which are closer to user equipment (UE) to accommodate services with tight delay requirements such as URLLC [1]. There are two kinds of MEC, termed Infrastructure MEC and Application MEC. Infrastructure MEC holds software-defined networking (SDN) and Network Functions Virtualization (NFV) services for supporting physical network infrastructure, for example, creating network slicing for various services. Application MEC hosts consumer applications in virtual machines (VM) and containers [2].

MEC is a kind of cloud system with a cluster of servers across access networks (AN) and core networks (CN). In terms of scalability, a cloud system has a vast number of servers in a centralized area while MEC has tens of servers in some AN and CN areas closer to UEs [3]. Providing scalability in a distributed system such as the MEC system is not trivial because each location may require a different capacity.

Several MEC studies have been conducted with and without taking into account the scalability issue. Studies [4] and [5] did not address scalability and focused on traffic redirection to MEC applications and GTP handling to develop MEC behind a base station. The authors of [6] considered a service placement optimization to alleviate a single MEC scalability problem. Wang *et al.* [7] took account of UE's device as system capacity. Studies [8–13] discussed scalable MEC. Moradi *et al.* [8] proposed SoftBox, a scalable infrastructure MEC framework at CN sites, while Sonkoly *et al*. [9] focused on optimizing the orchestrator to manage distributed scalable MEC. Studies [10, 11] developed scalable application MEC at AN sites. Scalable MEC simulations were conducted in [12] and [13]. Some of these papers did note scalability but did not consider any traffic patterns, such as hotspot traffic that may arise in a sporting event or a music concert that overloads an AN-MEC site. Most of them implemented one-tier MEC architecture, which deploys MEC servers only at AN or CN sites. The authors of [13] considered two-tier MEC architecture, but only used a single server for each MEC site.

This article proposes a two-tier multi-site multi-server MEC architecture, which places a cluster of servers at each AN and CN site to accommodate some traffic scenarios, including hotspot traffic. There are three challenges in a two-tier multi-site multi-server MEC architecture system. The first lies in deploying infrastructure, and that consumer applications of distributed MEC systems must be carried out centrally and automatically to reduce deployment complexity. The second is how to minimize the number of active servers automatically, appropriate to the various traffic scenarios while still addressing the latency satisfaction constraint. The third challenge is distributing the arriving traffic to available resources to minimize latency violation, which is categorized as a control plane challenge. Traffic can be offloaded verti-

Digital Object Identifier:
10.1109/MWC.111.2100004

Ying-Dar Lin, Widhi Yahya, Chien-Ting Wang, and Chi-Yu Li are with the National Yang Ming Chiao Tung University; Jeans H. Tseng is with Lanner Inc.

Paper	Scalability		Offloading	Scaling	Objective	Constraint	3GPP Integration	Implementation framework
	Multi-site single-server	Multi-site multi-server						
[8]	–	CN	Heuristic	Heuristic	Minimize Signaling overhead	Latency	O	Softbox
[9]	–	AN	Heuristic	Heuristic	Minimize VM mapping time	Latency	X	OpenStack, Docker
[12]	AN	–	PSO	PSO	Minimize cost-effectiveness	Latency	X	Simulation
[13]	AN, CN	–	Heuristic and ILP	Heuristic and ILP	Minimize active nodes	Latency	X	Simulation
[10]	–	AN	Contextual based	Location based services	Optimize utilization	Response time	X	OSM, OpenStack
[11]	–	AN	Load balanced	Mixed integer programming problem	Minimize active servers	Response time	X	SMOKE, Docker
Ours	–	AN,CN	Dynamic weight offloading (DWO)	Latency satisfaction aware autoscaling (LSAA)	Minimize active servers	Latency satisfaction percentage	O	OpenNESS

TABLE 1. Scalable MEC developments.

cally, from an AN site to CN sites, or horizontally, between AN sites or between CN sites.

This research proposes auto-deployment, autoscaling, and offloading modules to address the three aforementioned challenges, respectively. An auto-deployment module is a management plane module that tackles the distributed MEC deployment challenge. This module is represented in the form of a server-proxy-orchestrator architecture. The proxy is an orchestrating agent placed at each AN and CN MEC site. An autoscaling module is a management plane module that adjusts the number of active servers to satisfy arrival traffic. A Latency Satisfaction Aware Autoscaling (LSAA) algorithm is derived to determine the number of active servers. We propose Dynamic Weight Offloading (DWO) as a control plane module's algorithm to overcome the control plane challenge. A control plane module distributes traffic to servers based on their weight, which is dynamically adjusted based on their latency satisfaction percentage. Offloading and autoscaling modules are executed in the order of seconds and minutes, respectively, which are the short-term and long-term solutions to hotspot traffic.

The main contributions of this research are summarized as follows:

- This article proposes a scalable two-tier multi-site multi-server MEC architecture.
- The auto-deployment module was proposed to tackle the distributed MEC deployment challenge using a server-proxy-orchestrator architecture.
- The LSAA algorithm was proposed to address the autoscaling issue on the management plane.
- The DWO algorithm was designed to address traffic distribution among servers on the control plane.

We investigated and compared a two-tier multi-site multi-server with a one-tier multi-site single-server MEC in accommodating some traffic scenarios. The OpenNESS framework was used to implement and orchestrating a two-tier multi-site multi-server MEC testbed. The generated traffic was addressed to face detection applications that were deployed in edge nodes. To expand our understanding, we also investigated the effect of uniform vs. hotspot traffic; the effect of transient hotspot vs. persistent hotspot traffic; and with vs. without offloading and autoscaling modules in terms of cost and latency satisfaction percentage.

The remainder of this work is organized as follows. The following section presents the related works regarding the scalable MEC. We then describe the problem statements and solutions. Details of implementation are then described. Following that we provide the experiment results and analysis. The final section concludes this article.

Related Work

Table 1 compares the studies related to scalable MEC. A multi-site architecture was implemented in [8–13] to serve traffic from multiple base stations. The authors of [12] and [13] utilized a single server for each MEC site. The single-server MEC cannot handle bursty or heavy traffic. To solve this problem, the authors of [9–11] implemented multi-site, multi-server MEC only at AN sites, while Moradi *et al*. [8] implemented it only in CN sites. For better scalability, we implement multi-site multi-server MEC not only at AN sites but also at CN sites that employed a two-tier architecture. So, if hotspot traffic overloads all AN-MEC sites, then the traffic can be offloaded onto CN-MEC sites.

In a scalable MEC system, the management plane bonded to the control plane for some objectives such as minimizing the signaling messages of a cellular network [8], maximizing cost-effectiveness or system utilization [10, 12], minimizing VM mapping time for better system response time [9], and minimizing the resources [11, 13]. In [8] and [10], active container and VM adjustment was carried out, respectively. Neither of the studies considered deactivating an idle server, which also results in inefficient power consumption. Sonkoly *et al*. [9] only considered service placement and disregarded server activation and deactivation. The authors of [11–13] considered server activation and deactivation, while Tonini *et al*. [13] mini-

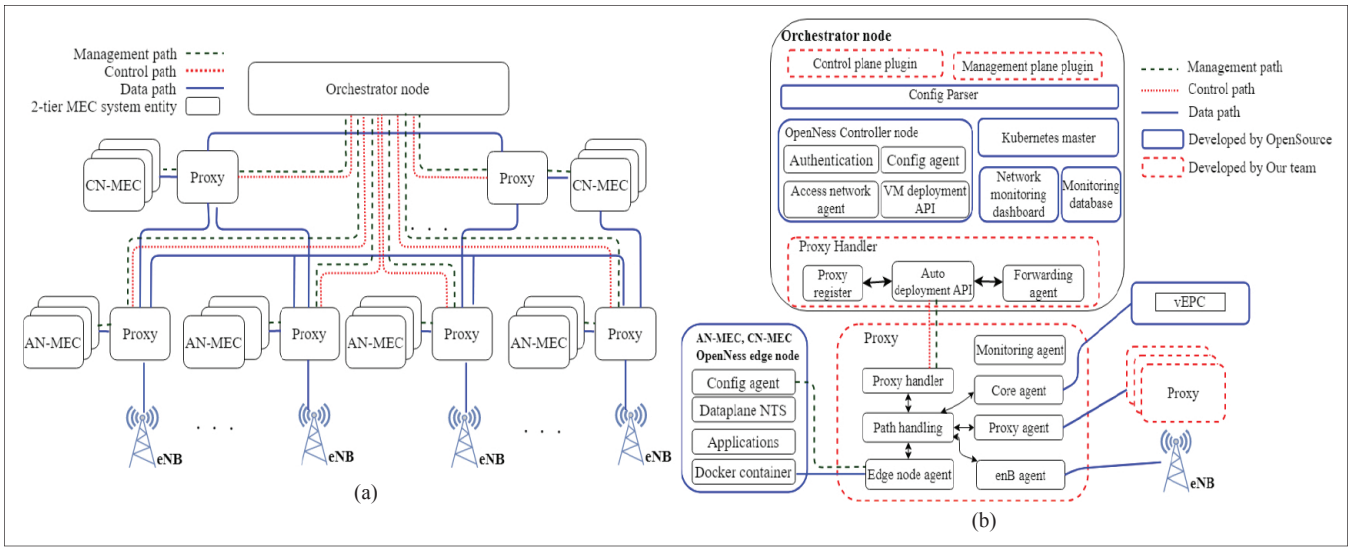

FIGURE 1. The two-tier multi-site multi-server MEC system: a) the two-tier multi-site multi-server MEC Architecture; b) implementation on top of OpenNESS.

mized the infrastructure MEC server while providing high availability. Rodrigues *et al*. [12] carried out scalable MEC simulations with scaling and an offloading mechanism that considered fixed latency constraint. Avegris *et al*. [11] implemented a SMOKE autoscaling framework, which uses a fixed response time as its constraint. Achieving a hundred percent satisfaction is difficult since a management plane runs in the order of minutes while traffic can fluctuate over time. The papers mentioned above did not address offloading and scaling onto control and management planes, which have different time domains. Our work proposes control plane and management modules that employ DWO and LSAA algorithms. Both algorithms use latency satisfaction percentage as its constraint instead of fixed latency constraint.

The implementation papers [9, 10] built MEC on top of OpenStack. OpenStack is a cloud development framework with no 3GPP integration. [10] utilized Open Source Mano (OSM) to manage edge networking. OSM is an open-source software to manage and orchestrate NFV. OSM enables 3GPP integration in OpenStack [14]. We implement a scalable MEC architecture on top of an OpenNESS framework which is specifically made for building MEC. OpenNESS also has 3GPP integration (https://www.openness.org/api-documentation/?api=cups). An OpenNESS controller utilizes Kubernetes for managing a container's life cycle, and is also equipped with some micro-services for edge networking.

Two-Tier Multi-Site, Multi-Server Architecture Design

This article proposes a two-tier multi-site, multi-server MEC architecture, shown in Fig. 1a. This architecture distributes a cluster of servers to each AN site behind the base station, and at each CN site of a cellular system. In 5G networks, a CN-MEC site connects to multiple AN-MEC sites. The CN-MEC shares its capacity if some of the AN-MEC sites become overloaded by hotspot traffic. Since hotspot traffic occurs rarely and randomly at any AN-MEC site, it is better to have more capacity at the centralized CN-MEC with wider coverage than to have more capacity at distributed AN-MEC sites.

In a distributed MEC system, the orchestrator's management module manages everything, servers, applications, traffic, and infrastructure services, centrally and efficiently via management paths. Proxy extends the orchestrator to manage the edge nodes automatically. The AN-MEC's proxy, which is the first hop user plane of the UEs, distributes traffic to some servers at its site or servers at another MEC site through the data paths. The term traffic that is used in this article is a service's requests which are generated by UEs. Traffic is distributed by each of AN-MEC's proxy based on the offloading weights dynamically adjusted by the control plane module via control paths.

Design Issues

Auto-Deployment: Managing a distributed two-tier MEC system is cumbersome because servers are situated in different geographic locations. In deploying a new MEC server, a system engineer installs and configures all the required packages manually by remoting or directly visiting a newly deployed server. Furthermore, deploying services manually by humans can cause configuration inconsistency, which can result in catastrophe. An auto-deployment module configuring a new deployed machine automatically and centrally is needed to reduce the deployment complexity and avoid configuration inconsistency.

Autoscaling: A two-tier multi-site, multi-server MEC architecture uses very many servers. Not all servers should be switched on all the time, as computing requirements will change dynamically. Some idle servers must be switched off to minimize operating expenditure (OpEx) in light arrival traffic volume, while in heavy arrival traffic rates, an exact number of servers must be turned on to satisfy computation requirements. The adjustment of the number of active servers is a part of a management plane problem which can be described as follows. Given several active servers at each site, arrival traffic rate, and latency information, the output is the number of active servers that satisfies a

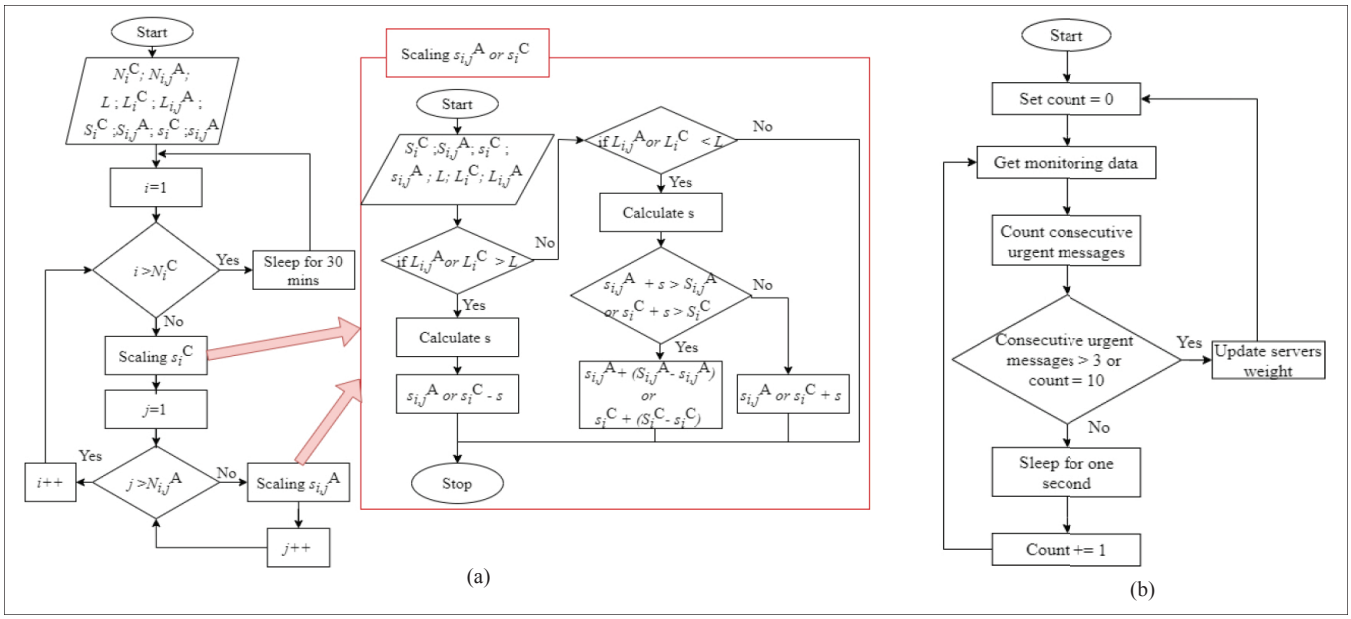

FIGURE 2. Autoscaling and offloading algorithms: a) latency satisfaction aware autoscaling (LSAA); b) dynamic weight offloading (DWO).

defined constraint. This work uses latency satisfaction percentage as a constraint, with the objective of minimizing the number of active servers. Latency satisfaction percentage is the percentage of the traffic which satisfies the latency constraint.

Offloading: The offloading module distributes traffic based on the server's load condition. Distributing traffic to overloaded servers can increase delay violations. The offloading module addresses the control plane problem, which can be described as follows. Given the number of AN and CN-MEC sites, the number of active servers, the servers' latency information, the output is the offloading weight based on the server's latency satisfaction percentage, with the objective of maximizing the latency satisfaction percentage at the currently active servers.

Solution Ideas

Orchestrator-Proxy-Server Architecture: The auto-deployment mechanism is applied by placing the proxy between the orchestrator and server, which establishes the orchestrator-proxy-server architecture, and which is shown in Fig. 1a. Generally, the orchestrator manages the servers directly. Since the proposed autoscaling module scales and configures physical servers automatically, that cannot be done by a current orchestrator such as Kubernetes, we propose a proxy as an orchestrator agent to do so. The orchestrator provides some information such as the orchestrator's IP address, edge applications, application-related configuration, edge hostname, and edge IP address to the registered proxy. When we add new servers to AN-MEC or CN-MEC sites, a proxy will bypass the configuration information between the new servers and the orchestrator, so the new machine's configuration can be set automatically.

The proposed proxy hosts several functions, such as proxy handler, path handling, edge node agent, core agent, proxy agent, eNB agent, and monitoring agent. As shown in Fig. 1b, proxies are controlled by a proxy handler which consists of a proxy register, an auto-deployment API, and a forwarding agent. The autoscaling and offloading modules manage a site through an auto-deployment API and forwarding agent, respectively. The proxy handler receives the orchestrator's commands regarding scaling and offloading decisions. The edge node agent handles the new deployment and scaling processes. After an edge node agent automatically carries out application configuration, Kubernetes can deploy the application automatically. Path handling responsible for traffic offloading is integrated with the cellular network path through core and eNB agents. A proxy can communicate with other proxies via a proxy agent. A specific application monitoring agent is also placed in a proxy for monitoring application latency by inspecting and probing approaches.

Latency Satisfaction Aware Autoscaling (LSAA): The proposed autoscaling module runs in orchestrator node in the order of minutes and consumes the server latency information, which is owned by the monitoring module. Figure 2a shows that the autoscaling algorithm has outer and inner iterations to scale the system. The outer iteration scales the active servers at CN-MEC (s_i^C), with the limit of the number of CN-MEC sites, N_i^C. In the inner iteration, the algorithm scales the active servers of each jth AN-MEC site, which is connected to the ith CN-MEC site. The total AN-MEC sites of each ith CN-MEC site is denoted by $N_{i,j}^A$.

The scaling function considers the latency satisfaction percentage of each AN and CN-MEC site which is denoted as $L_{i,j}^A$ and L_i^C, respectively. These satisfaction percentages represent the traffic that satisfies the latency constraint. The latency satisfaction percentage threshold, L, is used as the scaling threshold. The algorithm calculates s as the number of servers that will be turned on or off at a site. The capacity of a server (x) is derived by querying the traffic rate of the server, which has a satisfaction percentage a little bit greater than, or equal to, the defined satisfaction percentage threshold from the monitoring database. Assuming we carry out benchmarking for the deployed system in the system initiation, we can always derive x. The arriv-

Category	Parameter	Values
Two-tier multi-site multi-server architecture	AN-MEC sites	2
	CN-MEC sites	2
	The number of servers in each AN-MEC site	2
	The number of server in each CN-MEC site	1
One-tier multi-site single-server architecture	AN-MEC sites	2
	The number of server in each site	1
Hardware	AN and CN-MEC servers	HTCA 6200
	Orchestrator node	Core i7 with 32GB RAM
Latency setting	Latency constraint	300 ms
	Latency satisfaction percentage constraint	80 percent
Traffic of each AN-MEC	Light uniform	50 reqs/s
	Light with hotspot 1:2	50 and 100 reqs/s
	Light with hotspot 1:4	30 and 120 reqs/s
	Heavy uniform	100 reqs/s
	Heavy with hotspot 1:2	100 and 200 reqs/s
	Heavy with hotspot 1:4	60 and 240 reqs/s

TABLE 2. Parameter settings.

al traffic rates of a site $\lambda'_{i/i,j}$ in an interval time are predicted using the moving average algorithm. Assuming the MEC system contains homogeneous servers, the required number of servers at a site for the next interval time ($s'^{C/A}_{i/i,j}$) can be derived by taking the ceil of $\lambda'_{i/i,j}$ divided by x. So, $s = |s'^{C/A}_{i/i,j} - s^{C/A}_{i/i,j}|$ where $s^{C/A}_{i/i,j}$ is number of current active servers at a site. Each AN-MEC and CN-MEC site has a maximum number of servers that is denoted by $S^{A}_{i,j}$ and $S^{C}_{i,}$ respectively.

Dynamic Weight Offloading (DWO): Figure 2b shows the dynamic weight offloading (DWO) algorithm in detail. This algorithm runs in orchestrator node and applies the weight to the proxies placed at AN-MEC sites. A proxy will distribute the arrival traffic to available servers at AN-MEC and CN-MEC sites based on their weight, which is bound to the latency satisfaction percentage. The server's weight is calculated based on the ratio of its latency satisfaction to the sum of all the servers' latency satisfaction values. The idea is to distribute more traffic to servers with a high satisfaction ratio. The offloading module is executed in the order of seconds or, if the monitoring module receives three consecutive urgent messages, urgent messages are generated if the incoming traffic causes consecutive latency violations.

Implementation

The two-tier multi-site multi-server MEC implementation is grouped into sections as orchestrator or OpenNESS controller, OpenNESS edge node, monitoring system, and proxy implementation.

Orchestrator Node: OpenNESS Controller and Kubernetes

The system orchestrator was built on top of a server with an Intel Core i7 CPU and 32 GB RAM. The orchestrator was constructed by running the OpenNESS v.20.03 Playbook with the Network Edge option, and was integrated with the Kubernetes to manage containerized services. Some functions, such as authentication, config agent, network access agent, and VM deployment API, were added as container-based micro-services. The offloading and autoscaling modules were written in Python and ran in the orchestrator node every 10 seconds and 30 minutes. Both modules consumed the latency information from the monitoring database.

Edge Nodes: OpenNESS Edge Nodes

The edge nodes were built on top of four Lanner HTCA-6200 boxes. Each HTCA-6200 box consisted of two servers. Each server used an Intel Xeon processor, with 128 GB RAM, and a 32-port 10 GbE. The OpenNESS edges were also deployed through the OpenNESS Playbook. All OpenNESS edge nodes were connected to the orchestrator through a layer-two switch. The OpenNESS edge nodes were integrated with the Evolved Packet Core (EPC), which used the Open Air Interface (OAI).

Monitoring System

The developed monitoring system used inspecting and probing agents. The inspecting agent was placed in the proxy, and inspected and counted traffic flows that were going to the application container through the proxy. Pcap and dpkt were used in the inspection module. The probing agent generated a request to a specific application for measuring its latency. Probing was required because the inspecting module could not obtain latency information of nodes that were not the destination of current flows. The probing messages were sent to all servers every second. The latency and traffic information were saved in MongoDB, which ran in the orchestrator node.

Proxy

In extensive field deployment, we could not configure the machines individually or set the routing rules and connections manually, which was very inefficient. We thus designed a proxy to communicate AN-MEC, CN-MEC, orchestrator, EPC, and eNB. The proxy was implemented as a pod in the Kubernetes system. The proxy handler, path handling, edge node agent, core agent, proxy agent eNB agent, and monitoring agent were written in Python. The edge node agent turns the server on by using the Wake on LAN function or turns the servers off by shutdown command. The auto-deployment API and edge node agent were written in Python and used the Remote Procedure Call (RPC) approach. HAProxy was implemented as a proxy's path handling. Basically, the HAProxy is a load balancer that uses a round-robin or least-connection algorithm. In this work, the HAProxy was extended by the proxy handler to perform DWO.

Experimental Results

This section describes the parameter settings in evaluating a two-tier multi-site multi-server MEC architecture. Some scenarios are also determined for autoscaling and offloading modules. Results provide numerical results to answer research questions.

Parameter Settings

The parameter settings are shown in Table 2. The two-tier multi-site multi-server MEC testbed consisted of four sites, two AN-MEC sites and

two CN-MEC sites. All MEC sites were connected through a layer-two switch. The switch also connected all MEC sites to an orchestrator server. Each AN-MEC site had two servers, and each CN-MEC site had one server. The proposed system was compared to a one-tier MEC system, which only consisted of AN-MEC sites with one server for each site.

Two PCs were connected to AN-MEC sites for emulating the UEs. We used Jmeter to generate multiple requests to the face detection applications deployed in AN-MEC and CN-MEC servers as containers. The deployed face detection application used a web socket as its interface.

Face detection is usually implemented as a real-time streaming application. According to an ITU-T G.1010 recommendation on latency requirements of real-time services, which is 150-400 ms, we set the latency constraint to 300 ms. The latency satisfaction percentage threshold was set to 0.8 or 80 percent. Since overloading the MEC server is hard to achieve by only generating requests, the Stress tool was also used to overload the servers and increase node latency.

Scenarios

The two-tier MEC testbed was evaluated in six traffic scenarios: light uniform traffic, light with hotspot 1:2, light with hotspot 1:4, heavy uniform traffic, heavy with hotspot 1:2, and heavy with hotspot 1:4. Details of traffic rates can be seen in Table 2. The light and heavy with hotspot traffic scenarios are a kind of non-uniform traffic with different traffic ratios for each AN-MEC site. As noted above, the scalable MEC testbed has two AN-MEC sites. The ratio represents the ratio of arrival traffic of two AN-MEC sites. All of those requests were generated within one hour.

Results

Comparisons between the two-tier multi-site multi-server MEC and the less scalable one-tier multi-site single-server were also carried out in this section, addressing three key questions.

Uniform vs. Hotspot Traffic: Figure 3a shows the CDF of traffic latency for light traffic scenarios. 100 percent of light uniform traffic had latency less than the defined latency constraint. Both one-tier and two-tier MEC satisfied all the arrival traffic because the light uniform arrival traffic rate did not overload the AN-MEC servers. In serving uniform light traffic, both one-tier and two-tier MEC have one active server at each AN site. When the traffic changed to a hotspot 1:4, with offloading only, two-tier MEC still had a latency satisfaction of 66 percent. After scaling up the system by turning on one more AN-MEC server, two-tier satisfied 99 percent of hotspot traffic 1:4. In contrast, one-tier's satisfaction percentage dropped to 61 percent for hotspot 1:4. The one-tier with limited resources could not adjust its capacity to avoid server overload, which caused increasing node latency and latency violation.

In two-tier MEC, the autoscaling module responded to the heavy traffic by turning on all AN-MEC servers. As shown in Fig. 3b, two-tier MEC had 92 percent latency satisfaction, while one-tier only satisfied 45 percent of the arrival traffic. Without scaling, two-tier MEC still satisfied about 73 percent of the arrival traffic in the hotspot 1:4 scenario. The autoscaling module decided to turn on two CN-MEC servers since the heavy with hotspot overloaded all AN-MEC sites and could satisfy 86 percent of the traffic. In contrast, one-tier MEC with offloading only satisfied 32 percent of the arrival traffic in hotspot 1:4. The base-line system, one-tier MEC without offloading, and autoscaling module only satisfied 21 percent of the heavy with hotspot traffic. Most latency violations appeared in an AN-MEC site with hotspot traffic.

These results show that offloading was a short-term solution to non-uniform traffic, such as hotspot traffic. As shown in Figs. 3a and 3b, with only offloading, the satisfaction percentages of two-tier MEC were about 66 percent and 73 percent in light and the heavy hotspot traffic scenarios, respectively, while one-tier MEC without offloading satisfied only 25 percent and 21 percent of the traffic. Two-tier MEC systems must have an offloading module to redirect traffic from AN to CN-MEC sites. Two-tier MEC architecture with no offloading module is the same as one-tier MEC architecture, and therefore their latency satisfaction percentage of heavy hotspot 1:4 was the same, at 21 percent for both architectures. Bounded by offloading and autoscaling mechanisms, the two-tier MEC satisfied more than 80 percent of arrival traffic. The two-tier MEC has CN-MEC servers which are activated only if AN-MEC sites are overloaded, for example in serving heavy hotspot traffic. Uniform traffic decreases the latency satisfaction percentage according to its rate, with a low impact on the satisfaction percentage at a low rate and a high impact on high arrival rate. The autoscaling module played a key role in heavy uniform traffic. Changing the offloading ratio did not much affect the latency satisfaction percentage because all sites were equally loaded.

Transient Hotspot vs. Persistent Hotspot: The difference between transient hotspot and persistent hotspot lay in the occurrence time. The transient hotspot appeared in the system in short intervals, while the persistent hotspot occupied the system for a long time. The offloading module was responsible for distributing the hotspot traffic to all available active servers to minimize latency violation. As shown in Figs. 3c and 3d, the transient hotspot decreased the latency satisfaction percentage by only 7 and 10 percent in light and heavy traffic scenarios, respectively. Changing the hotspot ratio from 1:2 to 1:4 also did not much affect the satisfaction percentage because the offloading module distributes the arrival traffic based on the server's latency satisfaction, leading to fair traffic distribution in terms of latency.

The transient hotspot had a low probability of bringing latency satisfaction percentage below 80 percent. This probability depended on how long we generated hotspot traffic. In our case, the transient hotspot was generated for a total of 15 minutes in the first half of the experiment (one hour for each scenario). When the transient hotspot's latency satisfaction percentage was still higher than the latency satisfaction threshold, it did not trigger the scaling up mechanism. The persistent hotspot, as shown in Figs. 3a and 3b, decreased the satisfaction percentage equal to or less than 80 percent, which triggered the autoscaling module to scale the system up.

With vs. Without Offloading and Autoscaling: Figure 4a shows that the two-tier MEC with

Uniform traffic decreases the latency satisfaction percentage according to its rate, with a low impact on the satisfaction percentage at a low rate and a high impact on high arrival rate. The autoscaling module played a key role in heavy uniform traffic. Changing the offloading ratio did not much affect the latency satisfaction percentage because all sites were equally loaded.

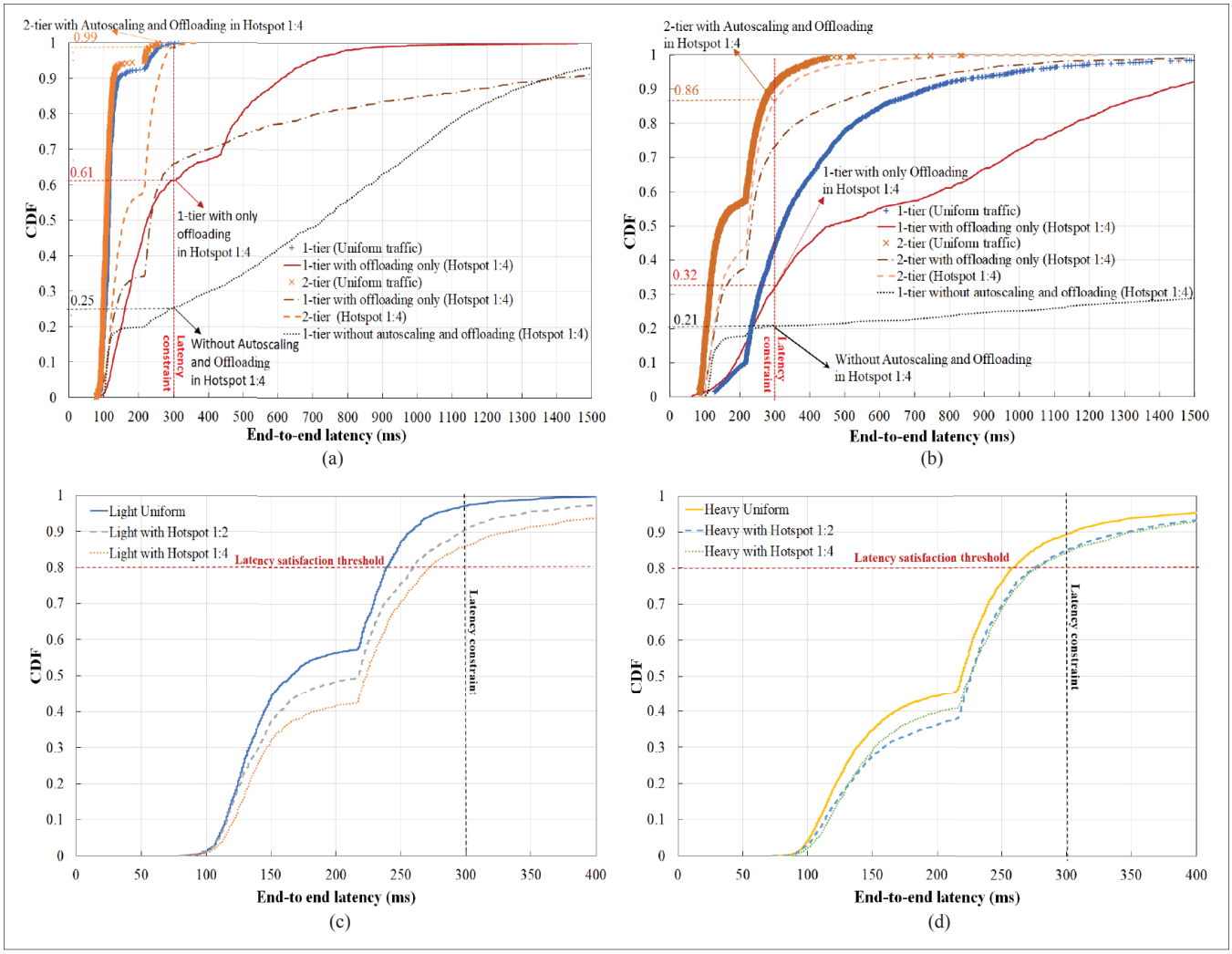

FIGURE 3. CDF of the arrival traffic latency in all scenarios: a) CDF of latency satisfaction in light traffic rate scenario; b) CDF of latency satisfaction in heavy traffic rate scenario; c) transient hotspot in light traffic scenario; d) transient hotspot in heavy traffic scenario.

an autoscaling module adjusted its active servers according to the arrival traffic rate while having a latency satisfaction percentage greater than 80 percent. As shown in Fig. 4b, without the autoscaling module, two-tier MEC had the highest latency satisfaction percentage, but it turned all servers on for all of the time, which is costly. An HTCA-6200 server consumes 1200 W on peak load and assumes 300 W on idle load. Since the relationship between CPU utilization and system power is linear [15], the MEC system with an autoscaling module used two AN-MEC servers with CPU utilization of about 30-70 percent, which consumed about 1400 W (each server consumed 700 W on average) in a light traffic scenario. Without an autoscaling module, the load was distributed to six servers with CPU utilization of about 20-30 percent and consumed 2700 W (each server consumed 450 W on average). Thus, the system with autoscaling could save up to 40 percent more energy than without an autoscaling module. In contrast, the one-tier MEC system had limited capacity to adapt to the increasing amount of arrival traffic, resulting in the worst latency satisfaction percentage. As shown in Fig. 4b, the one-tier latency satisfaction percentage dropped to 32 percent in handling 300 reqs/sec.

This study proposed a two-tier multi-site multi-server MEC architecture, which is equipped with auto-deployment, offloading, and autoscaling modules. The auto-deployment module enables the deployment of edge servers in a distributed area efficiently. Two-tier MEC with offloading (DWO) and autoscaling (LSAA) modules satisfy more than 80 percent of the traffic in all traffic scenarios, while the one-tier multi-site single-server MEC architecture only satisfies 32-61 percent of the traffic in serving heavy with hotspot traffic. Two-tier MEC shares CN-MEC servers that will be activated if the arrival traffic overloads AN-MEC sites and violates the latency satisfaction percentage constraint, while one-tier only relies on AN sites that may be overloaded in serving hotspot traffic. Offloading, which is executed every 10 seconds or when the offloading module receives three consecutive urgent messages, is a short-term solution in handling non-uniform traffic, such as hotspot traffic. Autoscaling, which runs every 30 minutes, is a long-term solution for scaling up a system when a persistent hotspot exists or scaling down a system when the latency satisfaction percentage is above the threshold. A persistent hotspot has a higher probability of bringing down the latency satisfaction percentage than a transient hotspot.

The proposed LSAA and DWO algorithms

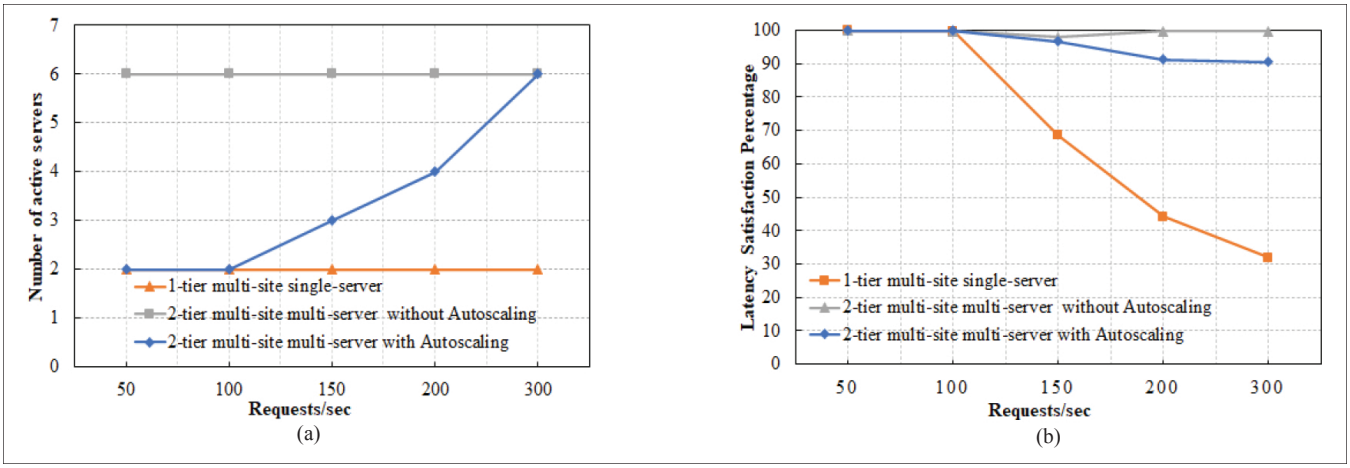

FIGURE 4. With vs. without offloading and autoscaling: a) allocated Resources; b) latency satisfaction percentage.

rely on an application latency requirement. Considering more applications should be evaluated in future studies. A combination of using a virtual machine and container for deploying applications also needs to be considered in developing an autoscaling module. Finally, further experiments in a realistic 5G environment are required for evaluating offloading and autoscaling module integration.

Acknowledgment

This work and its proof-of-concept system were supported by a project from Lanner Inc.

References

[1] Y. C. Hu *et al.*, "Mobile Edge Computing — A Key Technology Towards 5G," ETSI white paper, vol. 11, no. 11, 2015, pp. 1–16.
[2] 5G PPP Architecture Working Group and Others, "View on 5G Architecture, Version 3.0, February 2020," 5G Architecture White Paper, no. 3, 2020.
[3] V. Kumar *et al.*, "Comparison of Fog Computing & Cloud Computing," *Int'l. J. Mathematical Sciences and Computing (IJMSC)*, vol. 5, no. 1, 2019, pp. 31–41.
[4] C.-Y. Chang *et al.*, "MEC Architectural Implications for LTE/LTE-A Networks," *Proc. MobiArch '16*, 2016, pp. 13–18.
[5] C.-Y. Li *et al.*, "Mobile Edge Computing Platform Deployment in 4G LTE Networks: A Middlebox Approach," *HotEdge*, 2018.
[6] Z. Ning *et al.*, "Distributed and Dynamic Service Placement in Pervasive Edge Computing Networks," *IEEE Trans. Parallel and Distributed Systems*, vol. 32, no. 6, 2021, pp. 1277–92.
[7] X. Wang, Z. Ning, and S. Guo, "Multi-Agent Imitation Learning for Pervasive Edge Computing: A Decentralized Computation Offloading Algorithm," *IEEE Trans. Parallel and Distributed Systems*, vol. 32, no. 2, 2021, pp. 411–25.
[8] M. Moradi *et al.*, "SoftBox: A Customizable, Low-Latency, and Scalable 5G Core Network Architecture," *IEEE JSAC*, vol. 36, no. 3, 2018, pp. 438–56.
[9] B. Sonkoly *et al.*, "Scalable Edge Cloud Platforms for IoT Services," *J. Network and Computer Applications*, vol. 170, no. 4, Aug. 2020, p. 102 785.
[10] D. Spatharakis *et al.*, "A Scalable Edge Computing Architecture Enabling Smart Offloading for Location Based Services," *Pervasive and Mobile Computing*, vol. 67, no. 4, 2020, p. 101 217.
[11] M. Avgeris *et al.*, "Where There Is Fire There Is Smoke: A Scalable Edge Computing Framework for Early Fire Detection," *Sensors* (Switzerland), vol. 19, no. 3, 2019, p. 639.
[12] T. G. Rodrigues *et al.*, "Cloudlets Activation Scheme for Scalable Mobile Edge Computing with Transmission Power Control and Virtual Machine Migration," *IEEE Trans. Computers*, vol. 67, no. 9, 2018, pp. 1287–1300.
[13] F. Tonini *et al.*, "Scalable Edge Computing Deployment for Reliable Service Provisioning in Vehicular Networks," *J. Sensor and Actuator Networks*, vol. 8, no. 4, Oct. 2019, p. 51.
[14] T. Dreibholz, "Flexible 4G/5G Testbed Setup for Mobile Edge Computing Using OpenAirInterface and Open Source MANO," *Workshops of the Int'l. Conf. Advanced Information Networking and Applications*, Springer, 2020, pp. 1143–53.
[15] X. Fan *et al.*, "Power Provisioning for a Warehouse-Sized Computer," *ACM SIGARCH Computer Architecture News*, vol. 35, no. 2, 2007, pp. 13–23.

Biographies

Ying-Dar Lin is a Chair Professor of computer science at National Yang Ming Chiao Tung University (NYCU), Taiwan. He received his Ph.D. in computer science from the University of California at Los Angeles (UCLA) in 1993. His research interests include network softwarization, cybersecurity, and wireless communications. His work on multi-hop cellular was the first along this line, and has been cited over 1000 times. He is an IEEE Fellow and IEEE Distinguished Lecturer. He has served or is serving on the editorial boards of several IEEE journals and magazines, and was the Editor-in-Chief of *IEEE Communications Surveys and Tutorials* (COMST) during 2016–2020.

Widhi Yahya received his M.S. degree in computer science and information engineering from National Central University (NCU), Taiwan in 2014. He is a lecturer in computer science at the University of Brawijaya, Indonesia. He is currently pursuing a Ph.D. degree at the EECS International Graduate Program of National Yang Ming Chiao Tung (NYCU). His research interests are in the areas of network programming, software-defined networking, and multi-access edge computing (MEC) optimization.

Chien-Ting Wang received his M.S. degree in communications engineering from National Chung Cheng University (CCU) - Taiwan in 2013. He is currently pursuing his Ph.D. in computer science at National Yang Ming Chiao Tung University (NYCU). He is with the Graduate Degree Program of Network and Information Systems, National Chiao Tung University, Taiwan and Academia Sinica, Taiwan. His research interests include software-defined networking, network function virtualization, service chain placement and resource slicing.

Chi-Yu Li [M] received bachelor's and master's degrees from the Department of Computer Science, National Chiao Tung University (NCTU), Hsinchu, Taiwan, and the Ph.D. degree in computer science from the University of California, Los Angeles (UCLA), Los Angeles, CA, USA, in 2015. He is currently an associate professor with the Department of Computer Science, National Yang Ming Chiao Tung University (NYCU). His research interests include wireless networking, mobile networks and systems, and network security. He received the Award of MTK Young Chair Professor (2016), MOST Young Scholar Research Award (2017–2020), and Best Paper Award in IEEE CNS 2018.

Jeans H. Tseng is SVP, CTO and GM for Telecom Applications BU at Lanner Electronics Inc. He leads the development of vCPE benchmarking and NFV PoC with various third parties, including Verizon, Wind River and NTT Lab. He has over 30 years of experience in sales and technical management in the IT, networking, and telecom industries. Prior to Lanner, he was with Alcatel-Lucent from 1999 to 2013 as country general manager, VP of sales and President of Taiwan Alcatel Technology Co. Ltd. Prior to that, he was with Xylan Corp. as country general manager, D-Link and Siemens Telecommunication Systems as Director of the R&D Center and software engineer.