An NFV-Based Edge Platform for Low-Latency V2X Services Supporting Vehicle Mobility-Driven Auto Scaling

Wei-Chieh Hung, Shun-Ren Yang, Shan-Ni Lee, Yi-Chun Lin
Dept. of Comp. Sci. & Inst. of Info. Sys. and App. & Inst. of Commun. Eng.
National Tsing Hua University
Hsinchu, Taiwan
sryang@cs.nthu.edu.tw

Phone Lin, *Fellow, IEEE*
Dept. of Comp. Sci. & Info. Eng.
National Taiwan University
Taipei, Taiwan
plin@csie.ntu.edu.tw

2021 International Wireless Communications and Mobile Computing (IWCMC) | 978-1-7281-8616-0/21/$31.00 ©2021 IEEE | DOI: 10.1109/IWCMC51323.2021.9498707

***Abstract*—To guarantee low-latency and reliable end-to-end connectivity and provide suitable resources for vehicle-to-everything (V2X) services, the network function virtualization (NFV)-based multi-access edge computing (MEC) technology has been recognized as an effective solution. In the literature, few studies have proposed corresponding NFV-based MEC frameworks to provide V2X services. However, these works only discuss the concept of applying the NFV to the MEC to offer V2X services without implementing it in a real testbed. On the other hand, within the NFV-based MEC framework, several studies have investigated the NFV auto scaling mechanism to provide suitable resources for different services. However, these works are not suitable for V2X services, which exhibit unique traffic characteristics. In this paper, we (1) implement an NFV-based edge platform to provide vehicles with low-latency V2X services, (2) implement a vehicle mobility-driven NFV auto scaling mechanism that forecasts the resource requirements for V2X services and triggers the scaling operations if needed, and (3) use the open source software to build an NFV testbed and deploy our platform as virtual network functions to the NFV testbed. Finally, we implement a simple V2X service to validate our proposed edge platform, justifying that it, together with our NFV auto scaling mechanism, requires affordable processing delay and supports low V2X service latency.**

***Index Terms*—Auto scaling, multi-access edge computing (MEC), network virtualization function (NFV), vehicle-to-everything (V2X).**

I. Introduction

With the growing number of vehicles, more and more vehicle-to-everything (V2X) services, such as cooperative collision warning and remote vehicle diagnostics, are expected to improve drivers' comfort and safety. V2X communications, proposed by the Third Generation Partnership Project (3GPP) [1], has potential to help realize the demanded V2X services. V2X communications aim at providing low-latency and reliable information exchanges between vehicles, and between vehicles and other traffic infrastructure or pedestrians. With the V2X communications, for example, the traffic efficiency and the road safety can be improved, and the traffic congestion situation is further alleviated.

To guarantee low-latency and reliable end-to-end connectivity for V2X services, the ETSI multi-access edge computing (MEC) technology has been recognized as an effective solution, which deploys processing capabilities at the edge of the mobile network and achieves lower latency due to its close proximity to vehicles [2]. Recently, the ETSI network function virtualization (NFV) technology, which deploys network functions (including MEC) as software entities, namely virtual network functions (VNFs), in virtual machines (VMs) in commercial-off-the-shelf (COTS) servers, has been applied to realize the concept of MEC. Besides, the NFV supports auto scaling, which is a resource-management method used in cloud computing which automatically adjusts the volume of elastic computational resources allocated to applications based on their needs, thus dynamically accommodating the V2X service-demand variation.

Many studies have investigated and designed NFV-based MEC frameworks to provide scalable low-latency services [3] [4]. However, these studies have not focused on V2X services. Note that, the latency requirements of V2X services are more stringent than that of other services and the continuity of V2X services need to be considered while vehicles are moving, which causes that the aforementioned works are not suitable for V2X services. Indeed, few studies have put emphasis on providing V2X services based on the NFV-based MEC framework. For example, the work in [5] focuses on the MEC-empowered V2X services and the orchestration of the MEC using the NFV technology. However, these works only discuss the concept of applying the NFV to the MEC to offer V2X services without implementing it in a real testbed.

On the other hand, providing suitable resources for V2X services can be challenging due to the fact that the number of vehicles running on the roads varies from moment to moment and from region to region. Within the NFV-based MEC framework, several studies have investigated the NFV auto scaling mechanism to provide suitable resources for different services [6] [7]. However, these works are not suitable for V2X services, which exhibit unique traffic characteristics.

In this paper, we aim to implement an NFV-based edge platform, which embeds the ETSI MEC framework in the ETSI NFV framework, to provide vehicles with low-latency

V2X services and enable an NFV auto scaling mechanism that allocates suitable resources to V2X services. Our main contributions include:

- The design and implementation of an NFV-based edge platform for V2X services: This edge platform is modularized to realize: (1) the low-latency V2X service provisioning, (2) the NFV auto scaling for V2X services following our proposed NFV auto scaling mechanism, (3) the load balancing for service flow, and (4) the vehicle information updating.
- The design and implementation of a vehicle mobility-driven NFV auto scaling mechanism: This vehicle mobility-driven NFV auto scaling mechanism leverages the vehicle mobility information and a prediction model to forecast the resource requirements for V2X services in the next time interval and trigger the NFV auto scaling operations if needed.
- The open source module deployment in the NFV environment: We use the Open Source MANO (OSM) software and the OpenStack software to build an NFV testbed. Then, we deploy the edge platform modules as VNFs to the NFV testbed.
- The real-testbed evaluation of our NFV-based edge platform: We demonstrate that our NFV-based edge platform can provide low-latency V2X services to vehicles and our vehicle mobility-driven NFV auto scaling can provide suitable resources to V2X services in comparison to other scaling methods.

II. NFV-based Edge Platform Architecture

To enable low-latency V2X services over the MEC in NFV architecture, an NFV-based edge platform is designed. Besides, to allocate suitable resources to V2X services, a vehicle mobility-driven NFV auto scaling mechanism is proposed to be incorporated into the platform. In the following, we discuss the overall architecture of the platform.

As shown in Fig. 1, our NFV-based edge platform follows the ETSI NFV framework [8], where (1) our proposed edge platform (EP) provides V2X services and allocates suitable resources to V2X services using the NFV auto scaling mechanism, (2) the NFV management and orchestration (NFV MANO) manages the resource and lifecycle of the EP, and (3) the NFV infrastructure (NFVI) provides the virtualized resources to the EP. In EP, we merge the MEC application platform and the MEC applications, defined in the ETSI MEC host level framework [9], into the NFV environment by deploying them as VNFs. Specifically, the EP consists of four modules: the information module (IM), the load balancing module (LBM), and the auto scaling module (SM) together implement the MEC application platform to provide essential functionalities for the MEC applications, while a V2X application module (AM) acts as an MEC application to offer a V2X service. These modules are deployed as VNFs, which are composed of multiple VMs.

The EP modules cooperate with each other to provide V2X services for vehicles as well as allocating suitable resources to the services. The AM offers V2X services to vehicles. Note that, in our architecture, the serving area of the EP is divided into several sections, each of which is allocated an AM to provide services for vehicles therein. The following describes the functionalities of these modules:

AM: The AMs provide V2X services. Specifically, acting as a VNF, an AM executes a V2X application in the background and continuously waits for the service requests. If any vehicle requests a V2X service, the service request will be received by the LBM first, and then relayed to the AM. The AM will process the request and send a service response back to the vehicle, relayed also via the LBM.

LBM: The LBM relays service flow between vehicles and AMs and handles load balancing periodically. In terms of load balancing, the LBM chooses a suitable AM for each requesting vehicle based on the vehicle position and the resource usage of each AM.

SM: The SM allocates suitable VM resources to the AMs. To be specific, the SM adopts our vehicle mobility-driven NFV auto scaling mechanism, a forecasting scaling, to forecast the resource requirements of the AMs based on vehicle mobility and trigger the NFV auto scaling operations accordingly.

IM: The IM stores and updates information for other modules to query. The IM records the vehicle information, including position and velocity. Besides, the IM stores the VM information of each AM, including the number of VM each AM has launched and the number of vehicles each of these VMs serve.

III. The Platform Operations

This section details how our modules interact with each other to provide scalable V2X services for vehicles. We discuss the defined data types followed by all EP modules, then the platform operations formed by module interaction to help achieving our goal of providing scalable V2X services for vehicles, and finally our proposed NFV auto scaling mechanism.

A. Platform Data

Our EP maintains and operates on two types of data, vehicle data and VM data, to allow the V2X service provisioning and dynamic V2X service resource allocation. The vehicle data and the VM data are respectively stored in the vehicle table and the VM table, which are manipulated by our IM. The vehicle data specify some general vehicle information, including username, position, and velocity. Besides, the vehicle data leaves some space for application information, which can be defined by developers. When a vehicle registers (deregisters) to (from) our EP, the corresponding vehicle data is created (deleted) in our vehicle table by our IM with the vehicle's username as the index. When a vehicle updates its information to our EP, its vehicle data indexed by the username is updated in our vehicle table accordingly. On the other hand, to provide suitable resources for V2X services, the VM data specify the VM information of each AM, where V2X applications are

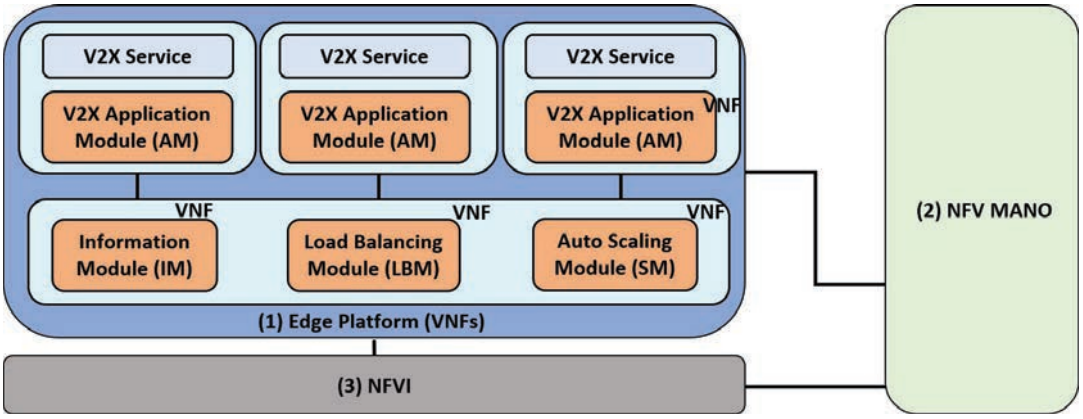

Fig. 1: System Architecture

executed, including the VM id list, the number of vehicles each VM serves, and the VM IP addresses for connections. When an AM launches (stops) a VM, the corresponding VM data is created (deleted) in our VM table by our IM with the VM id as the index. When a request from a vehicle is passed to an AM, one of the AM's allocated VMs is selected (under our LBM) to serve the vehicle, and the number of vehicles served by the selected VM is updated in the VM table.

B. Platform Operations

To achieve the goal of V2X service provisioning and resource allocation, our EP performs the following main operations: registration/deregistration, vehicle information update, V2X service request and response, load balancing, and auto scaling.

1) Registration/Deregistration: Before a vehicle can request a V2X service, it should first register to our platform. When our IM receives the register request from the vehicle, it retrieves the username and the position from the request and stores them in the vehicle table for the vehicle. The register request is then passed to our LBM, which selects a suitable AM, together with a designated VM, to serve the vehicle according to the vehicle's position. After selection, a connection is established between our LBM and the chosen AM so that our LBM can relay the subsequent service flow. Finally, the selected AM asks the IM to update the number of vehicles served by the designated VM so as to keep track of the latest resource utilization.

On the other hand, to stop services, the vehicle should deregister from our platform. For this, the vehicle sends a deregister request to our IM, triggering our IM to remove the username and position of this vehicle from the vehicle table. Then, the request is passed to our LBM for it to terminate the established connection with the serving AM. Finally, the disconnected AM asks the IM to update the number of vehicles served by the VM designated to the deregistered vehicle in our VM table.

2) Vehicle information update: During the service period, the vehicle should periodically update to our IM its position and velocity information and specified application information for V2X applications.

3) V2X service request and response: The vehicle can send a V2X service request to our LBM, which relays the request to the serving AM. After processing, the AM sends a service response back to our LBM, which then relays the response back to the vehicle.

4) Load balancing: Our platform periodically selects a more suitable AM (with a VM) to provide service for each moving vehicle, achieving load balancing. Our LBM requests the latest vehicle position from our IM and selects a suitable AM based on the position. In case a new AM is selected, our LBM disconnects from the old AM and establishes a connection with the new AM. Besides, the old AM and the new AM will both ask the IM to update the numbers of vehicles served by their designated VMs in our VM table.

5) Auto scaling: Our platform periodically applies our proposed NFV auto scaling mechanism, which will be detailed in the next subsection, to dynamically allocate suitable resource to each AM. Specifically, in the mechanism, our SM requests the vehicle information and the VM information from our IM and then forecasts the resource requirement of each AM to see whether to perform scale-out operation, which means adding the number of VMs, or scale-in operation, which means reducing the number of VMs.

C. Vehicle Mobility-Driven NFV Auto Scaling Mechanism

We implement a vehicle mobility-driven NFV auto scaling mechanism which is periodically adopted by the SM to forecast the resource requirements based on the vehicle mobility information and dynamically scales the resources accordingly.

Our vehicle mobility-driven NFV auto scaling mechanism periodically executes the following four steps:

1) Obtain vehicle and resource information: The mobility information and current position of each vehicle are requested from the IM for predicting the vehicle's position in the next time interval in Step 2. The information of resources allocated to each AM in the last time interval is obtained for scaling decision.

2) Do vehicle position prediction: The position of each vehicle in the next time interval is predicted using a specific machine learning model. For the demonstration purpose, in our current implementation, each vehicle is assumed to move with fixed velocity, where the vehicle position can readily be anticipated. Note that, other more sophisticated models can also be applied in our mechanism to capture the vehicle mobility in reality. After this vehicle position prediction, the maximum number of vehicles to be served by each AM in the next time interval is counted.

3) Calculate whether resources are suitable: The maximum resources required by each AM in the next time interval is calculated based on the predicted maximum vehicle number. By comparing the calculated required resources with the allocated resources in the last time interval, obtained in Step 1, the mechanism can decide whether to perform scale-out or scale-in operation. If the resources are enough, nothing is adopted. Otherwise, the NFV auto scaling operations are adopted. If the resources are too much, the scale-in operation is adopted; if the resources are not enough, the scale-out operation is taken.

4) Update resource information: Finally, the information of the latest allocated resources after scaling is updated by the IM, which will then be utilized for the calculation in the next time interval. Following our NFV auto scaling mechanism, the resources provided for each AM would not be wasted and can be provided in time.

IV. Module Deployment

To verify the feasibility of our EP, a testbed is constructed as the proof-of-concept (PoC). In this section, we first introduce the testbed construction in which an NFV environment is created. Then, we introduce the way to deploy the EP modules to the testbed.

A. Testbed Construction

To construct an NFV environment, we need an NFV MANO software, which manages the whole NFV system, and a cloud computing software, which realizes the NFVI functionalities and acts as VIM to control and manage the virtualized resources for VNFs. After constructing the NFV environment, we need to handle the network settings so that these two pieces of software can collaborate with each other. In the following, we provide more details for the NFV environment construction first and then the network configuration.

1) NFV environment construction: The Open Source MANO (OSM) and the OpenStack are chosen for the NFV MANO software and for the cloud computing software respectively. In our work, the OSM is installed using the shell scripts it provides, while the OpenStack is installed using the RDO package [10]. Besides, the hardware equipment to install the two pieces of software are as follows:

- NFV MANO software: Intel Xeon Silver 4110, 16GB RAM, and Ubuntu 18.04 LTS 64-bit.
- Cloud computing software: AMD Ryzen 5 2400G 3.6Hz, 16GB RAM, and centOS 7 64-Bit.

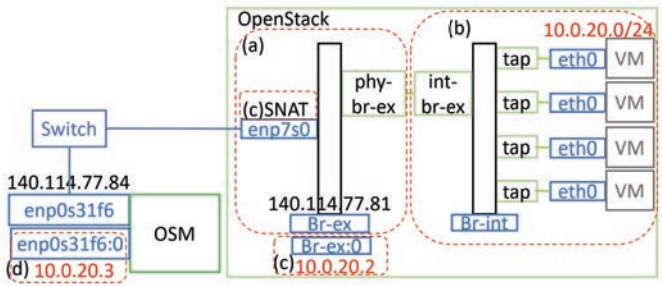

Fig. 2: Network Configuration

2) Network configuration: We should configure the network of the OpenStack to fulfill the following two requirements: (1) the Internet accessibility of our EP modules, which are VNFs deployed over VMs in the OpenStack, and (2) the reachability from the OSM to the OpenStack, which enables the OSM to configure the VNFs.

As shown in Fig. 2, we have only two public IP addresses, one for the OpenStack, and the other for the OSM. To fulfill (1), we create a public network with a subnet in the OpenStack. Then, we perform the source network address translation (SNAT). As for (2), we give the OSM an alias IP address in the same subnet created in the OpenStack.

B. Module Deployment

To deploy the EP modules to our testbed, we first package the implementation of these modules into customized VM images. Then, we use the OSM VNF onboarding to form these VM images as VNFs and deploy the VNFs to our testbed. The OSM VNF onboarding should fulfill three lifecycle stages, including (a) Day 0: VNF Instantiation & Management Setup, (b) Day 1: Service Initialization, and (c) Day 2: Runtime Operations. In the following, we show what we do in each stage to include all the requirements, instructions, and elements, following the guidelines provided by the OSM document [11].

1) Day 0: In this stage, the VNFs are instantiated and the management is set up via VNF packages so that the VNFs can be configured at the following stages. Besides, the VNFs need to be packed into a network service (NS) via a NS package so as to be instantiated by the OSM system. Therefore, we first design several VNF packages, each of which contains a VM image to be launched, namely our customized image, a network interface to connect to the outside network, and some management settings for Day 1 and Day 2, which will be discussed in the following stages. Then, we design an NS package, which connects all our VNF packages to our established public network in the OpenStack through their network interfaces to allow communications among them. Finally, we instantiate the NS, which launches all our VNFs as VMs in the OpenStack.

2) Day 1: In this stage, the VNFs are configured so that the services inside the VNFs can be automatically initialized. A proxy charm, a set of scripts for deploying and operating

software through ssh, is built to initialize the services. Therefore, we add ssh configurations, including ssh-hostname, ssh-username, and ssh-password, in the VNF packages to allow the management from the OSM to the VM. Note that, we do not need to initialize the VNF services, because we have enabled service automation while creating customized images. However, we still need to add the ssh configurations, because the OSM needs to use ssh in the next stage.

3) Day 2: In this stage, reconfiguration of the services in the VNFs, including the auto scaling operations of the VNFs are taken. The OSM adopts the threshold-based auto scaling method, which monitors a specific metric to see whether the metric exceeds the scale-out threshold or falls below the scale-in threshold and triggers auto scaling accordingly. However, we need the OSM auto scaling process to follow our forecasting results. Therefore, we set our SM output as 1 or 0 as the metric to be monitored and both the scale-out and the scale-in thresholds as 0.5 in the VNF packages. In this way, while our proposed auto scaling mechanism forecasts that the resources are too much or not enough, the OSM will trigger scale-out or scale-in accordingly.

V. Performance Evaluation

To validate the performance of our proposed EP, we implement a simple V2X service and a demonstrative program (DEMO) to simulate vehicles requesting the V2X service in our testbed. In this section, we first describe the simple V2X service and the DEMO. Then, we follow the traffic flow around National Tsing Hua University (NTHU), Taiwan, to set up the simulation of the DEMO. Finally, we evaluate the performance of our EP, and compare our vehicle mobility-driven NFV auto scaling with other scaling mechanisms.

A. The Simple V2X Service and the DEMO

We implement a clustering V2X service to divide the vehicles into multiple groups, where vehicles in the same group can communicate with each other directly, while vehicles in different groups can only communicate via their cluster heads. This clustering V2X service allows to further realize sophisticated cluster-based V2X protocols. Note that, when a vehicle requests our clustering service, our service will randomly select a suitable cluster head for it depending on its position, its moving direction, and its nearby vehicles. The DEMO, on the other hand, simulates vehicles moving and periodically requesting our clustering service from our EP, which performs platform operations to take corresponding actions for each vehicle.

B. Simulation Setup

We set up the simulation in terms of the serving area, vehicle mobility, and service request, according to the statistics analysis gathered by the government of Hsinchu City [12]. Without loss of generality, we proportionally scale down the time and the serving range based on the density of vehicles.

1) Serving area: We consider a serving area with size 1000 $\times$ 1000 m^2, partitioned into four square sections, each of which is allocated an AM to provide services. Moreover, each AM contains a VM initially, which can serve 200 vehicles.

2) Vehicle mobility: We consider a total number of 350 vehicles, each of which is randomly distributed and randomly given a direction in the serving area initially, and then moves for ten minutes with a constant speed of 10 m/s without direction changes.

3) Service request: Each vehicle generates a clustering request per second while moving, upon which our EP will respond to the vehicle by assigning a suitable cluster head.

C. Experiment Results

This subsection shows the V2X service latency measurements, and the latency and power consumption performance of our vehicle mobility-driven NFV auto scaling mechanism compared with (1) static scaling, a method that consistently provides resources that meet the maximum requirement, and (2) threshold-based scaling, a method that performs scale-out or scale-in based depending on whether the resource requirements exceed or fall below the defined threshold.

1) V2X service latency measurements: We first measure the average latency of our clustering V2X service $\overline{L}$, which is defined as follows:

$$\overline{L} = \sum_{i=1}^{n} (P_i + V_i + R_i)/n,$$

where n represents the total number of service requests from vehicles, P_i represents the processing time of our EP modules for request i, V_i represents the VM launching time request i has to wait for a new VM activation when the resources of the serving AM are not enough, and R_i represents the round-trip transmission time of a pair of request and response packets for request i. When a service request i is sent to our EP, we calculate the required P_i and V_i (if any) and estimate R_i using existing statistical data. Note that, to calculate the V_i values, we assume that if a vehicle does not receive a service response within 1000 ms (i.e., the interval between two consecutive service requests from the same vehicle in our experiment) after it sends a service request, its V_i is set as 1000 ms.

Our experiment results show that for our clustering V2X service, the average $\overline{P}$ is 0.99 ms, while the average $\overline{V}$ is 0 ms. The reason for the zero average $\overline{V}$ is that under the assumption that each vehicle moves with an unchanged speed and direction, we can perfectly predict the resource requirements and allocate suitable resources in advance. Moreover, given the negligible $\overline{P}$ and $\overline{V}$, we note that the service latency is thus dominated by the round-trip packet transmission time R_i. Considering the case that LTE is used as the transmission technology, R_i can be roughly estimated as 32.5(+-0.16), based on the statistics of 2019 Chunghwa 4G Latency counted by OpenSignal [13].

2) The NFV auto scaling method comparison: We compare the performance of our vehicle mobility-driven NFV auto

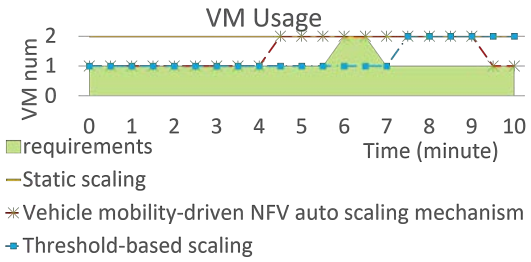

Fig. 3: VM Usage of Different Auto Scaling Methods

scaling mechanism with that of the static scaling method and the threshold-based scaling method.

In terms of the average launching time, the experiment results show that the average launching time of the static scaling method is 0 ms, the same as that of our scaling mechanism, while the average launching time of the threshold-based scaling method is 7.3 ms, much more than that of our scaling mechanism. This is because the static scaling method continuously provides enough resources, requiring no VM launching operations. However, the reactive threshold-based scaling method uses a fixed scale-out threshold and cannot adapt to the real-time changes of resource requirements by preparing the demanded resources beforehand, leading to a higher launching latency faced by each request.

As for the VM usage, Fig. 3 indicates that the VM usage of the static scaling method is more than that of both the threshold-based scaling method and our scaling mechanism, while the VM usage of the threshold-based scaling method is more than that of our mechanism. This result for the static scaling method is obvious since it consistently provides resources that meet the maximum requirement. For the threshold-based scaling method, we note that although it can perform the scale-out/scale-in operations to adjust the amount of allocated resources, as Fig. 3 shows, the provided resources can hardly match the corresponding requirements and a significant portion of the resources can be wasted. Therefore, we claim that our scaling mechanism achieves a better VM utilization, thus reducing the overall power consumption than the other two methods.

VI. Conclusion

To guarantee low-latency and reliable end-to-end connectivity and provide suitable resources for V2X services, the NFV-based MEC technology has been recognized as an effective solution. This paper (1) implemented an NFV-based edge platform to provide vehicles with low-latency V2X services, (2) implemented a vehicle mobility-driven NFV auto scaling mechanism that forecasts the resource requirements for V2X services and triggers the scaling operations if needed, and (3) used the open source software to build an NFV testbed and deploy our platform as VNFs to the NFV testbed. Moreover, we implemented a simple V2X service and a demonstrative program in our testbed to validate our proposed edge platform. Our experiment results indicated:

- Our proposed edge platform takes affordable processing latency and VM launching latency to support V2X services, and consequently the total service latency is mainly dominated by the round-trip transmission latency.
- In terms of VM launching latency, our proposed NFV auto scaling mechanism performs similar to the static scaling method but obviously better than the threshold-based scaling method, while our mechanism outperforms the other two methods from the perspective of VM utilization, thus leading to the least power consumption.

Acknowledgment

This work was supported in part by the Ministry of Science and Technology (MOST), Taiwan, under Contracts 106-2923-E-002-005-MY3, 107-2221-E-007-117-MY3, 108-2221-E-007-034-, and 108-2823-8-002-007, the Ministry of Education Higher Education Sprout Project, and National Taiwan University under Grant NTU-109L883503.

References

[1] 3GPP, "Study on enhancement of 3gpp support for 5g v2x services (v16.2.0, release 16)," 3rd Generation Partnership Project (3GPP), Technical Report (TR) 22.886, Dec. 2018. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3108

[2] M. Emara, M. C. Filippou, and D. Sabella, "Mec-assisted end-to-end latency evaluations for c-v2x communications," in *2018 European Conference on Networks and Communications (EuCNC)*, 2018, pp. 1–9.

[3] A. Boubendir, E. Bertin, and N. Simoni, "On-demand, dynamic and at-the-edge vnf deployment model application to web real-time communications," in *2016 12th International Conference on Network and Service Management (CNSM)*, 2016, pp. 318–323.

[4] S. Li, Z. Guo, G. Shou, Y. Hu, and H. Li, "Qoe analysis of nfv-based mobile edge computing video application," in *2016 IEEE International Conference on Network Infrastructure and Digital Content (IC-NIDC)*, 2016, pp. 411–415.

[5] F. Giust, V. Sciancalepore, D. Sabella, M. C. Filippou, S. Mangiante, W. Featherstone, and D. Munaretto, "Multi-access edge computing: The driver behind the wheel of 5g-connected cars," *IEEE Communications Standards Magazine*, vol. 2, no. 3, pp. 66–73, 2018.

[6] B. Yang, W. K. Chai, Z. Xu, K. V. Katsaros, and G. Pavlou, "Cost-efficient nfv-enabled mobile edge-cloud for low latency mobile applications," *IEEE Transactions on Network and Service Management*, vol. 15, no. 1, pp. 475–488, 2018.

[7] I. Sarrigiannis, K. Ramantas, E. Kartsakli, P. Mekikis, A. Antonopoulos, and C. Verikoukis, "Online vnf lifecycle management in an mec-enabled 5g iot architecture," *IEEE Internet of Things Journal*, vol. 7, no. 5, pp. 4183–4194, 2020.

[8] ETSI, "Network functions virtualisation (nfv); architecture framework (etsi gs nfv 002 v1.1.1)," ETSI Industry Specification Group (ISG), Group Specification (GS) 002, Oct. 2013. [Online]. Available: https://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.01.01_60/gs_NFV002v010101p.pdf

[9] ——, "Multi-access edge computing (mec); framework and reference architecture (etsi gs mec 003 v2.1.1)," ETSI Industry Specification Group (ISG), Group Specification (GS) 003, Jan. 2019. [Online]. Available: https://www.etsi.org/deliver/etsi_gs/MEC/001_099/003/02.01.01_60/gs_MEC003v020101p.pdf

[10] RedHat, "OpenStack, packaged for and tested on CentOS." https://www.rdoproject.org//, [Online].

[11] OSM, "OSM Usage," https://osm.etsi.org/docs/user-guide/05-osm-usage.html, [Online].

[12] H. C. Government, "Urban Planning Division Announcement," http://www.v523.tw/upload/fileUrl/2015-11/05/616d1834-07f1-4ac4-865d-60a2f7ab5e8b.pdf, [Online].

[13] OpenSignal, "Taiwan Mobile Network Experience Report," https://www.opensignal.com/reports/2019/06/taiwan/mobile-network-experience, Jun. 2018, [Online].