Testbed for Multi-access Edge Computing V2X applications prototyping and evaluation ("Regular paper")

Bilel Cherif[1,2], Pascal Berthou[1,2], Nicolas Riviere[1,2], Yann Labit[1,2]

1 CNRS, LAAS, 7 avenue colonel Roche, F-31400 Toulouse, France
2 Univ de Toulouse UPS, F-31400 Toulouse, France
Email:{bcherif, pberthou, nriviere, ylabit}@laas.fr

Abstract—Multi-access Edge Computing (MEC) is one of the key enablers behind intelligent transportation systems (ITS) futuristic applications. To address and evaluate applications that aimed at offering a service in this context, researchers and developers need a prototyping tool to abstract this system aspects. However, currently available tools do not model this environment nor give the possibility of running under development MEC services for evaluation purposes. In this paper, we propose a MEC vehicular application testbed, combining virtual technologies and network emulation tools. The goal is to help through the process of evaluating proposed solutions and applications destined to offer MEC ITS services.

Keywords— Multi-access Edge Computing, V2X applications, MEC Emulation environment, MEC testbed, ITS, Network Emulation.

I. Introduction

The shift toward next-generation automotive systems driven by intelligent transportation systems (ITS). ITS concept introduces a new vehicular software model. ITS systems are intended to offer a whole new set of services to improve the automotive system's safety, comfort, and efficiency. One primary concept under the umbrella of ITS is the connected vehicles (CV) concept, which aims at equipping vehicles with communication capabilities. Connected vehicles are intended to collaborate through a network infrastructure and computing platform in order to realize the aforementioned goals behind the concept of ITS. A Cloud-based centralized model has become the default approach for network-based services [1]. However, many vehicular applications demand stringent latency, reactivity, reliability, and bandwidth requirements that a centralized Cloud environment could not satisfy [2] [3] [4]. Current proposals [5] [6] are promoting a significant shift towards a distributed architecture to overcome the significant limitations of Cloud IoT platforms regarding connected vehicle applications, such as mobility, location awareness, and ultra-low latency.

Multi-access edge computing (MEC) technology is promising to offer ultra-low latency, high bandwidth, and real-time access to radio network information that can be leveraged by a new set of on Cloud vehicular safety and real-time applications. The core idea of MEC is to move computation closer to the user, whereby computing capabilities (Micro Data Centers or small servers) that can host Cloud applications at the mobile network edge. The "edge" term refers to the fusion of base stations (or access points) and data centers close to the mobile radio network. Operators can open their RAN edge to authorized third-parties, which will motivate connected vehicles industry to deploy innovative applications and services flexibly and rapidly.

The key components that can provide processing and storage capabilities at the edge of the network are called MEC hosts [7]. The MEC hosts are involved in the data processing of different V2X applications. The MEC hosts offer virtualized resources located between centralized vehicular Cloud and the vehicle's onboard system as a middle point. Depending on the application requirement, they can play the role of relaying points or support advanced tasks, including data caching, Geo-localization, real-time analysis, load balancing, resource management, and security. Regarding the software model, the MEC paradigm pushes critical service away from the central nodes, e.g., Datacenter (DC) or Cloud, to the logical extremes of a network. Ideally located a "single-hop" away from the user, the closer the application and the content, the better Quality of Experience (QoE) is delivered to the end-user [8]. By pushing the critical services away from the centralized environment, the MEC paradigm removes bottlenecks and potential points of failure, thus making it more preferably resilient to failure.

However, The MEC system needs management software that globally coordinates between MEC infrastructure hosts and services deployment. The MEC management software will, for sure, have an impact on the deployed services and their interaction with the end-user requests. Very recently, fundamental work [9], [10], [11], [12] by ETSI have proposed a reference architecture, design goals, and the essential components of the mobile edge computing platform. In such an environment, the creation of MEC services involves steps such as: the development of the service functions; the services distribution between MEC and Cloud levels; its integration with the system that performs life cycle management and orchestration; the implementation and test of

management interfaces; the implementation and validation of service-specific management components.

A current issue in this area is the lack of supporting tools to prototype and test services in MEC scenarios. To the best of our knowledge, there are no readily available MEC testbeds that can help researchers and developers to design and verify distributed algorithms destined to run in a similar environment. At the development phase of V2X MEC service components, supporting tools can reduce deployment time, save costs, and improve the quality of the offered services. These tools should allow testing MEC services interaction regarding the end-user and the other hosts, such as exchanging different vehicle onboard sensor data through the network and also validating its interaction with the management system.

In this context, Continental Digital Service France (CDSF) and LAAS-CNRS started the eHorizon project (2017-2021) for addressing the research and technological issues of ITS systems. This paper deals with presenting the global ITS system architecture, as well as an analytical view of the system's architecture and the requirements of the applications in MEC vehicular applications (V2X) use case. Finally, a testbed is proposed to offer a prototyping platform for the MEC V2X application.

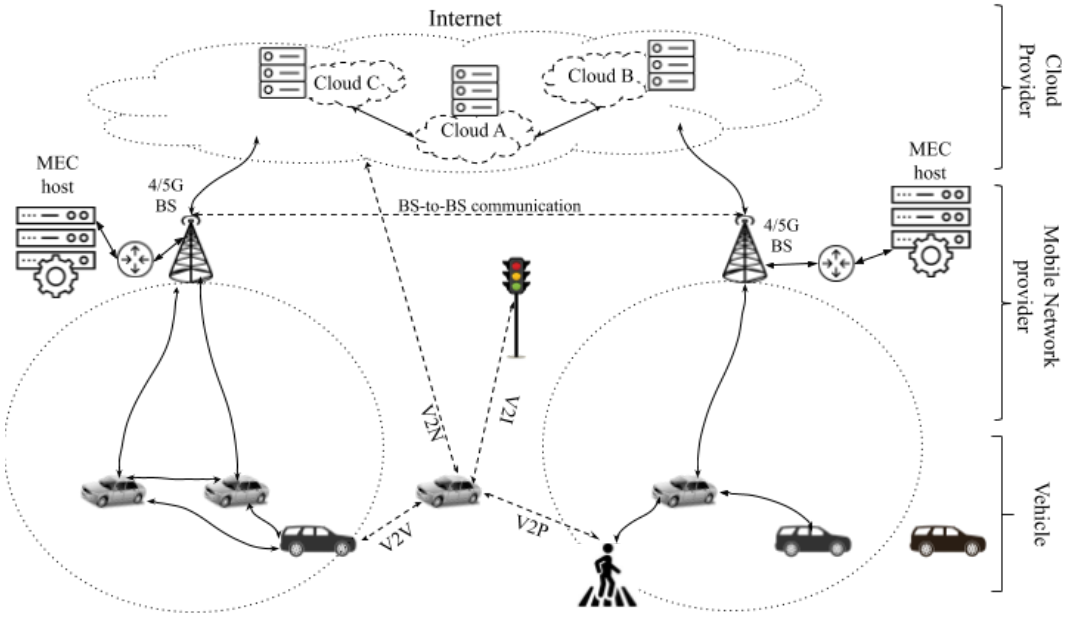


Figure.1 *Intelligent transportation system global architecture*

II. The global architecture of ITS system

In this section, we describe the global architecture of an ITS system that will serve later in our system analysis to justify our technical solutions. First, we present the main components of the architecture, their corresponding interactions, and the used technological standards. Finally, we discuss the main challenges of developing MEC V2X application.

A. Architecture Main Components

The global architecture of an ITS system consists of four main parts as shown in ***Figure.1*** (1) Vehicles, (2) Network infrastructure, (3) Cloud platform, and (4) MEC platform:

Vehicles: A vehicle is equipped with a set of sensors and systems (GPS, Radar, Lidar, Advanced Driver Assistance System (ADAS), camera, etc.), enabling the surrounding environment perception (position, speed, neighboring vehicles, temperature, etc.).

Network infrastructure (cellular network): The upcoming Fifth Generation cellular network (5G) is one of the leading technologies that may support different vehicular communication technologies. It promises to grant a very high network capacity that guarantees high throughput/bandwidth for demanding applications. According to [13], 5G networks will natively include mobile edge computing capabilities by design. Moreover, cellular networks are characterized by a wide communication range, which allows a base station to maintain connectivity with a network node (vehicle) as long as possible, which means fewer handover operations. In addition, it offers Multicast/Broadcast transmission services (MBMS/eMBMS) and device to device (D2D) communication technologies since the Fourth Generation.

Cloud computing infrastructure: The Cloud offers a centralized high storage and processing capabilities to collect and process massive data volumes to provide customized ITS services to different vehicles.

Mobile edge computing infrastructure: The MEC infrastructure offers a distributed limited storage and processing capabilities in

the vicinity of the end-users (vehicles) to cope with the Cloud issues regarding real-time services ultra-low latency requirements.

B. Communication types

In the context of ITS, a vehicle could interact with surrounding systems through various types of communication as specified in [14], and shown in ***Figure.1*** :

V2V (Vehicle-to-Vehicle): A type of communication where two vehicles could communicate with each other through the network infrastructure or using direct communication.

V2P (Vehicle-to-Pedestrian): A type of communication where a vehicle could communicate with a pedestrian cellphone through the network infrastructure or using direct communication.

V2I (Vehicle-to-Infrastructure): A type of communication where a vehicle could communicate with a roadside unit or infrastructure (traffic lights, traffic signals, etc.) through the network infrastructure or using direct communication.

V2N (Vehicle-to-Network): A type of communication where a vehicle could communicate with a serving entity through a network (internet to access to Cloud/Edge hosts).

C. MEC V2X applications challenges

ITS applications should deal with various data provided through various communication links. The density and the mobility of the vehicles are the major factors to take into account when developing V2X applications that are destined to run in mobile edge computing distributed environment. In a high-density scenario, the application must use the available bandwidth efficiently in order to deliver its service continuously to the subscribed nodes. The application should also deal with nodes joining/leaving the network during their mouvement and guarantee the service continuity. The MEC infrastructure is intended to offer a management system and a couple of standard services that facilitate the aforementioned properties. The MEC infrastructure introduces a whole new service model that requires an adapted development paradigm.

An important aspect that should be taken into account while developing a V2X application that targets MEC as a deployment environment is to ensure the proper interaction of the application with the management entities and the system components. At the development phase of a V2X MEC service components testing in real environment increase deployment time, costs, and complexity. In the next section, we analyze the standard deployment architecture proposed by the European Telecommunications Standards Institute (ETSI) to extract the main components and their functionalities.

III. Multi-access edge computing reference architecture

ETSI MEC reference architecture, as described in [15], is mainly composed of functional blocks and reference points, allowing interactions among them. It is essential to understand that in the proposed architecture, functional blocks may not necessarily represent physical nodes in the mobile network, but rather software entities running on top of the virtualization infrastructure. The reference architecture is split into two main levels: (1)the host level and (2)the system level. The system-level management is interconnected to the host level management over reference points. On the one hand, the system-level management manages mainly the life cycle, rules and services authorization, and traffic rules of the application. On the other hand, the host level management entity ensures the allocation, management, and release of virtualized resources provided by the virtualization infrastructure located on the MEC server. By analyzing the reference architecture, the main building blocks and their respective functions that are relevant to our work are:

Mobile Edge Platform Manager: Mobile Edge Platform Manager (MEPM) is responsible for Mobile Edge platform management, application lifecycle management (instantiation and termination), application rules management (authorization, DNS configuration, and resolving...) and application requirement management functions.

Mobile Edge Orchestrator: Mobile Edge Orchestrator (MEO) is the core functionality of the MEC system-level management layer. The MEO maintains an overall view on available computing, storage, and networking resources and services. The MEO handles the task of the appropriate ME host selection for requested services deployment by taking into account the application requirements, the available resources, and the nodes' positions. Finally, The MEO is also responsible for scaling up and down the available resources as required by the running applications.

Virtualization Infrastructure Manager: Virtualization Infrastructure Manager (VIM) obviously responsible for virtualized resources management. The VIM tasks consist of allocating and releasing the storage, networking, and compute resources offered by the virtualized infrastructure. VIM could also store application images for faster instantiation procedure when it is required. VIM also provides support for fault and performance monitoring by collecting virtual resources and running application data and transmitting them to the system level management entities.

IV. Mobile edge computing prototyping testbed requirement and technologies

In order to evaluate MEC V2X applications, the prototyping testbed should provide an easy way to install, configure, manage, upgrade, and terminate the running services. Termination of services must not affect other services or resources within the concerned hosts. Edge/Cloud hosts model should have different resources limitation and management models too. Each host should include specific resources limits. Services deployment should be possible on any host as long as free resources are available. Vehicles mobility induces the creation and destruction of network topology links at runtime following a mobility model. The platform should handle the vehicle's mobility by dynamically adapting the network topology. The different communications types (V2V, V2N, etc.) should have various link types. Different links that relay different nodes should be attributed with the proper properties (bandwidth, delays, etc.).

Based on previously detailed analysis of MEC based V2X application deployment environment and global ITS system, we retrieved the main requirement for our testbed platform. Our

testbed architecture is based on technological solutions that meet the following requirements: (1)low cost deployment, (2)setup flexibility, (3)network topologies and computing resources emulation and management (corresponding to ETSI specification), (4)applications management, (5)node mobility behavior handling, and (6)the support of real-world protocols and services. We targeted Open source frameworks because they are the most cost-effective solutions. However, as far as we know, the existing solutions do not model MEC environments and V2X systems, where services can be deployed distributed and managed on edge (MEC) and Cloud resources. Furthermore, simulators makes it difficult to support real-world communication protocols and services interactions. Therefore, in this section, we present the available Open Source solutions that satisfy our platform requirements partially and allow the execution of v2x applications in a dynamic and controllable environment.

A. Mininet network emulator

Mininet [16] is an open-source software emulator for prototyping a large network on a single machine. Mininet is widely used by the research community for network architecture topologies emulation [17], [18], [19]. Mininet can be used to quickly create realistic virtual network topologies running actual software application code on a personal computer. Mininet allows the user to create, interact with, and customize a network architecture. It uses the Linux kernel tools to create virtual hosts, switches, and links. Unlike the simulators, it allows the deployment of protocols and real applications code in a close to real-world setup. Virtual hosts can run standard Linux software using Linux namespaces. Furthermore, virtual hosts are isolated through network namespaces with their own set of network interfaces, IP addresses, and a routing table. Virtual switches connect one or more virtual hosts via the creation and destruction of virtual network links in runtime, which makes it suitable for modeling nodes mobility and different communications types modeling. Mininet, by default, does not handle any host mobility models natively but could be extended for that purpose. Mininet allows link properties configuration regarding bandwidth, loss rate, and delay through an extensible Python API. However, the Mininet virtual host approach downside is the sharing of the host machine file system by default.

B. Docker Containers engine

Docker [20] is a software platform that allows the creation of containers. Containers are a lightweight, standalone executable package that includes application code and its dependencies, such as executable system libraries and configuration settings. Docker container uses the operating system's level virtualization and offers a lightweight virtualization infrastructure. However, unlike VM images, the containers do not bundle a full operating system. Furthermore, Docker containers isolate the instances resources using the same technologies that Mininet does for its virtual hosts, based on Linux cgroups and kernel namespaces. They both use virtual Ethernet interface pairs to connect the virtual hosts to the virtual switches.

C. Docker Swarm

Docker swarm mode allows the orchestration and management of a cluster of Docker Engines, natively within the Docker platform. Docker swarm, easily deploy application services to a swarm and manage swarm behavior. The main function of docker swarm are (1) Coordinate between containers and allocate tasks to groups of containers, (2) Perform health checks and manage the lifecycle of individual containers, (1) Provide redundancy and failover in case nodes experience failure, (3) Scale the number of containers up and down depending on load, and (4) Perform rolling updates of software across multiple containers. Which makes it a perfect candidate for MEC hosts management in our case.

V. Testbed Architecture

In this section, we present the global architecture of our emulation tool. In order to build a flexible tool capable of emulating the MEC environment, we propose an architecture built on top of some existing tools, and we extended some of their functions to fit our targeted system. For the purpose of developing our testbed, we extended the Mininet framework to allow the management of Docker containers as virtual nodes, handle host mobility, allow resources management, and connectivity management. The developed testbed provides capabilities to run, manage, and orchestrate MEC vehicular applications under various conditions and network typologies.

The general architecture and the essential components of our proposed testbed, as well as their interconnections, are illustrated in ***Figure.2***. The MEC testbed API is the core component of the testbed. It implements the required functions and interfaces to create the emulation environment. The MEC testbed API implements an abstraction for Mininet core functions that are necessary to build an emulated network environment (topology definition, virtual nodes/switches instantiation, and links creation). Furthermore, it allows the user to apply limitation models that define the available resources for each host, and interact with the emulated components (execute commands on a certain host, record log files, etc.).

The emulated environment is built of virtual nodes, virtual switches, and virtual connections. The MEC testbed API interacts with Mininet to instantiates the virtual nodes from pre-created Docker container images. A container image can be instantiated more than once in an emulated environment. A virtual instance is a collection of virtual nodes and their respective resources model to emulate a group of MEC hosts or a Cloud environment. The virtual instance abstraction allows the management of the related set of virtual nodes and virtual switches as a single entity. The V2X services could run in one or more virtual nodes inside a virtual instance.

The developed MEC testbed API enables adding/removing, and connecting/disconnecting containers to virtual instances dynamically within the created network topology. These extended features allow the emulation of real-world MEC and Cloud infrastructures in which it is possible to start and stop services instances at any point in time. Also, it allows the adjustment of the containers and the virtual instances resource limitations (CPU allocated resource, and memory resources) at runtime by interacting with the Docker engine. The application running

environment can be created by deploying virtual nodes and virtual instances into a virtual network environment running on a host machine. Preconfigured container images could be launched as virtual nodes in the emulated environment once it is created. Each container image comprises part of a distributed application, as well as the required services and protocols.

The mobility API interacts with the Mininet core and the virtual hosts in order to initiate/update vehicle nodes position, create/destroy communication's virtual links, and manage mobility models. A real-world map could be imported to the SUMO mobility simulator to model an accurate mobility model that simulates vehicle movements. For this purpose, we designed a custom client interface that interacts with SUMO through the Tracii interface. The Tracii interface follows a client-server model where the mobility model API instance requested the up to date vehicle positions from the running SUMO instance at each simulation step.

The management system connects the emulated environment using the MEC testbed developed API. The management system starts the required process and services in each virtual node to run the under test application. The different emulation processes are interconnected to this entity to ensure the coordination between the emulation components. The Docker swarm, when used, it takes in charge the task of orchestrating the different Docker virtual nodes. Docker swarm implements a node check function and nodes replacement in case of failure. When a node fails, Docker swarm replaces this node by a new instance and communicates with the management system to set up the appropriate updates. The management systems also coordinate between the emulated environment and the docker swarm to add or eliminate virtual nodes from virtual instances. The management system also keeps track of the available resources on each virtual instance. The amount of available resources is checked every time before instantiating a new virtual node to determine if the operation is possible or not. In order to set up an appropriate environment for evaluating MEC V2X services, our testbed architecture was designed to offer the following features:

Topology flexibility: Through the use of Mininet virtual switches to connect virtual hosts, the MEC testbed API creates easily flexible network typologies to include the different host's interconnections links (V2V links, V2I links; etc.). The API also uses the native traffic control offered by (TC) Linux kernel to attribute different properties to the created network typologies (delay, bandwidth, etc.).

Nodes mobility support: The developed mobility API implements easily different mobility models. The nodes positions could be extracted through the interaction with a real-world map offered by the interconnection with SUMO emulator, or with a mobility model that calculates the successive node position following the implemented mobility model. The mobility support is achieved through the use of the mobile nodes' positions regarding the fixed host (Edge hosts) to create and destroy communication links in runtime.

Virtual instances resources management, adjustment, and orchestration: The management system and the testbed API interacts with the virtual instances to ensure the allocated resources limitation, and keep track of used resources quantity of each virtual instance. On the one hand, the MEC test API manages the launching of Docker's application instance on the appropriate hosts and offer their lifecycle management. On the other hand, it groups the virtual nodes under different virtual instance with tagged labels to ease their orchestration using Docker Swarm.

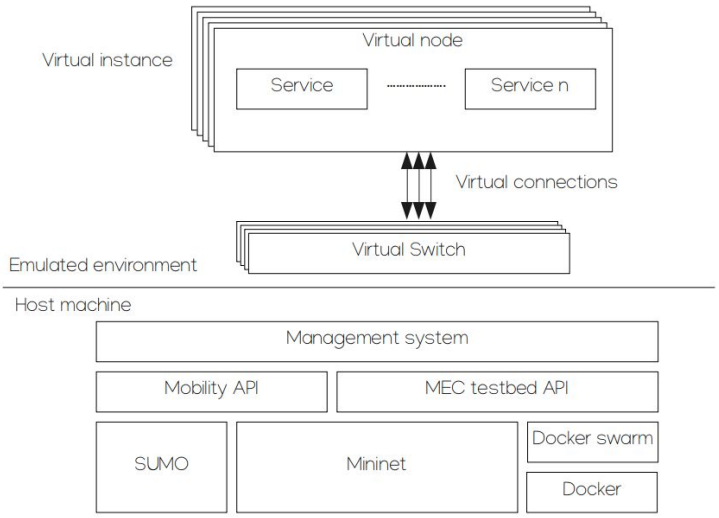


Figure.2 *An overview of MEC testbed architecture*

Fully functional application deployment environment: Our platform is capable of modeling a close to real-world emulation environment by modeling the network topology, the host's resources, and locations, the running application chunks (the MEC host services, the Cloud host services, and the onboard application).

Services and host isolation: Deploying virtual nodes in the form of Docker instances offers a virtualized isolated environment for service execution at each host level. Using the cgroups, and namespaces each node is isolated from the others that are executing on the same virtual instance. The use of virtual interfaces and virtual links allows different nodes and instances to communicate easily.

Real-world protocols support: It is possible to use virtual interfaces to interconnect running docker instances with real-world protocol stacks. The use o*f* real-world protocols is very useful in the process of *validating* the different service's interactions among each other, and with the management instances.

VI. MEC testbed workflow

In this section, we provide full details about the steps required to use our testbed tool. At each step, the underlying details to achieve the specified step are briefly explained. A MEC testbed emulation using our testbed platform involves many steps to describe the desired network architecture and the different configurations. An example of a MEC environment emulation in the context of ITS is illustrated in *Figure.3*, where three types of containers were used to instantiate the different node actors in this context. Each container image bundles a part of the distributed application (service), required protocols, could run a particular Linux distribution, and could even emulate a specific processor architecture.

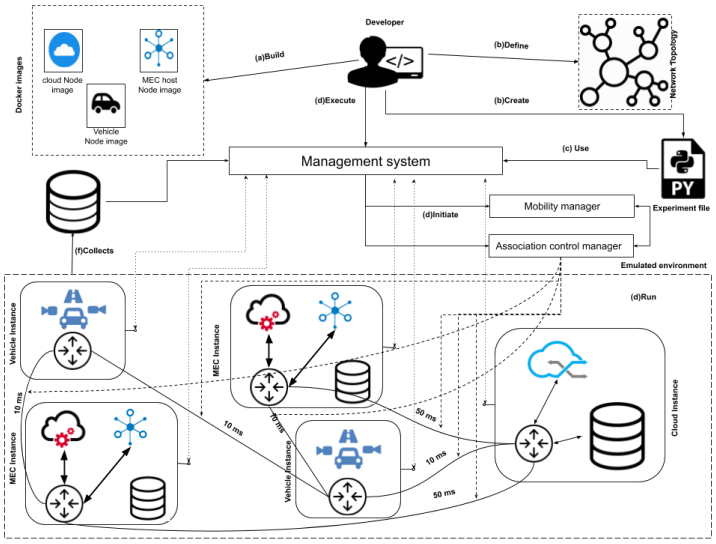


Figure.3 *MEC testbed workflow*

Figure.3 shows the high-level workflow and the different steps to launch an emulated environment using our emulator to evaluate a particular scenario. (a) First, the developer should provide the container images of the different actors that are relevant to his scenario. Each container should include the application slice that runs on each type of node and its dependencies, such as the executable code, required scripts, configuration files, required libraries, and so on. (b) Next, the developer should define the network topology and the link's properties of the fixed part of the network (Cloud hosts and Mobile Edge Cloud hosts) of the under test scenario. (c) The developer defines the mobility model for the mobile nodes (vehicles in our case). At this step, the developer provides a scenario file that contains all the previous steps configurations (Network topology, docker images, mobility model, etc.). (d) The next step is to launch the management system that connects to the emulated environment by using the MEC testbed API. At this stage, the management system initiates the mobility model and deploys the application on the platform by starting the required processes and services on each virtual node. Furthermore, the management API launches the association control manager, which creates and destroys dynamic network links based on the mobility of the nodes. In order to fulfill the previous operation, the management system uses the scenario description file provided by the developer. (e) The application starts running inside the platform and interacts with the virtual network. (f) Finally, the network flow statistics and the application interactions could be collected and stored for analysis purposes. Furthermore, the developer could interact with the environment using the provided CLI interface and check each instance state using docker swarm provided web GUI.

To simulate a scenario using our testbed, an experiment file that describes the network topology (MEC hosts, Cloud hosts, vehicles ….), the used mobility model, the different network links properties. The creation of a scenario file could be done using the following steps:

A. Host images building: at this step, after coding and verifying the application code that needs to be deployed on each node, a docker file should be created for each type of nodes. A docker image should be created for each type of node bundling each node application, scripts, and dependencies with a specific docker image name.

B. Experiment file definition: This step consists of writing an experiment file that describes all the emulation parameters. This file should describe the whole emulated environment. The experiment definition file is taken by the system manager to build the virtual environment and lunches the scenario emulation. The experiment file should contain the following elements:

- Resource model specification: The resource tables are an abstraction that holds the allocated resources for each host in our scenario. This step consists of defining the resources tables for each type of the participating nodes(MEC hosts, Cloud hosts). Default resource tables are provided through our MEC testbed code. If the user defines none, the default resource tables are used instead.

- Topology and nodes instantiation: this step consists of creating an experiment instance through our specified MEC testbed API. The topology class is an abstract class that stores the hosts, their resources tables, and vehicles participating in our scenario. The topology instance uses the resources tables to calculate the remaining resources at each host after deploying services on them. Specific service deployments on a particular host could be rejected if the remaining host resources are not enough for that purpose.

- Topology Definition: This step consists of defining the fixed network topology that links the nonmobile nodes (MEC hosts and Cloud hosts). Our testbed API provides the necessary functions to create virtual switches and virtual links between the different hosts to create a virtual network. The virtual network is created using our API by calling Mininet emulator core and instantiating the necessary components to create the described network topology.

- Link specification: Our testbed API provides a link abstraction that uses the Linux traffic control (TClink) tool to create a virtual link with specific parameters (bandwidth, delay …..). TClink tool is a part of the Netemu package that gives the user the ability to configure the kernel packet scheduler. The TClink tool controls a specific link bandwidth and delay through the queueing discipline (Qdisc) approach in Linux. A Qdiscs in Linux is a packet scheduler (e.g., FIFO), which gives a flexible way to control different links properties (such as bandwidth and delay). At this step, we specify each type of links parameters for each type of link (e.g., vehicle to MEC host link or MEC host to Cloud link etc ..).

- Mobility model definition: We provide two ways to define the mobility model for the instantiated vehicle nodes. This could be done through an external mobility simulator, SUMO mobility simulator in our case. If we choose to use mobility traces from SUMO mobility simulator is launched with a provided map template. The second way supported by our testbed platform is to instantiate a thread that holds a mathematical mobility model to calculate the vehicle nodes positions at each emulation step. Our code bundles some mathematical mobility models like the random mobility model or the Gauss-Markov model that could be used directly just by specifying the model's required parameters.

- Association control Model: The association control class is an abstraction of the rules that are used to create a virtual link

between two nodes. A simple association control model could use the algebraic distance between nodes to create a new link or destroy an existing one at each emulation step. This step consists of defining the association control model parameters, such as the communication range. The association control thread when launched, it verifies at each step the position of mobile nodes toward the fixed hosts' nodes to create new links or destroy existing ones.

- Experiment definition: The last step consists of instantiating an experiment object that takes all the previously configured parameters. The experiment instance is used by the management system to lunch the Emulation scenario with the set parameters and manages the emulation at runtime. Our API provides different methods to configure each parameter taken by the experiment object.

VI. Use case

To study the benefits of using our testbed, we designed a use case application of real-time traffic monitoring where vehicles communicate with MEC deployed services to post/update their locations and speed. The MEC host service analyzes the collected vehicles' data to determine the vehicles' traffic flow. Additionally, the MEC services post/update traffic flow information to remote Cloud service of its deployment region. Finally, the Cloud services store the traffic flow information in a database and make them accessible to the other hosts.

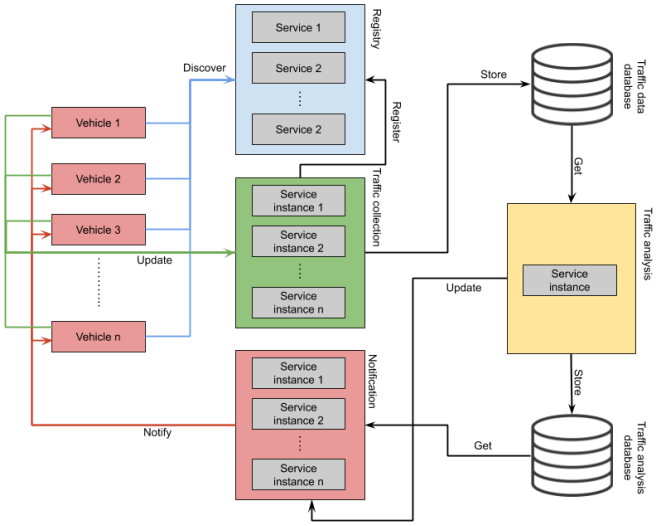


Figure.4 *Vehicles traffic monitoring service architecture*

B. Use case application

As illustrated in ***Figure.4***, the architecture of the use case application is split into five microservices. The traffic collection microservice provides a restful API to the vehicles to register into the service and update their respective positions at each timestamp. It communicates periodically with the traffic database microservice. Collected data are triggered by the traffic analysis microservice to compute the traffic density factor that is then stored in a dedicated database. Finally, the notification microservices compare the up-to-date traffic density factor with the recorded history of the traffic density factor at the same hour of the day. If the notification microservice detects an abnormal augmentation of traffic density factor at a specific time, it sends a notification to a group of vehicles to inform them about the probability of having a road hazard on a specific zone.

B. Use case implementation and discussion

In order to evaluate the testbed architecture, we implemented a full application prototype. The implementation of the prototype applications was developed with a microservices oriented architecture pattern as detailed in the previous section. We developed two scenarios. In the first, we generated an emulated environment with a single Cloud host. In the first scenario, all the application microservices are deployed on the Cloud host. The vehicles communicate only with the Cloud host during this scenario to update their positions or to receive notification messages from the notification service. The second scenario was implemented in a mixed Cloud MEC environment where part of the microservices are deployed on the MEC hosts and the rest on the central Cloud. The services partitioning criteria was based on resources that are available on each type of resource (MEC host are resource-constrained compared to Cloud hosts) and on the network link delay. Table 1 resumes the characteristics of the used computer and the simulation parameters for both scenarios.

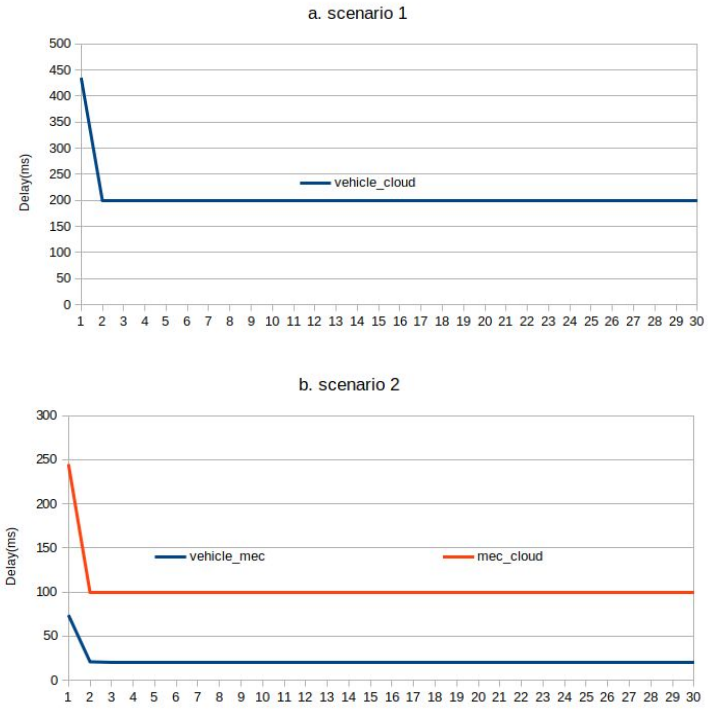
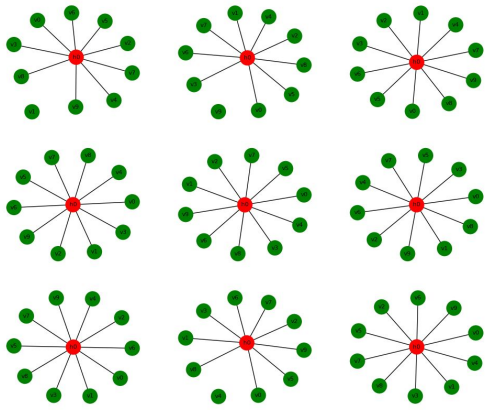


Figure.5 *Different link types round trip delay*

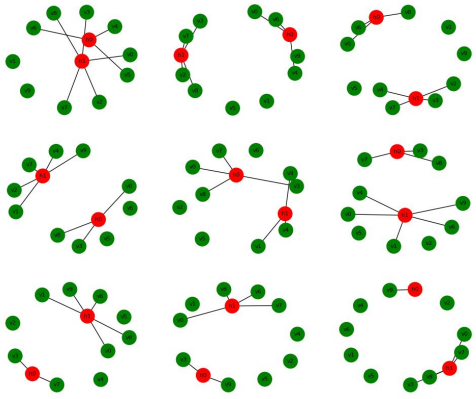
In the following, we present the data that we collected after running our application in both scenarios. We chose to present only logged data that are relevant to our testbed platform and excluded the data related to the running service functionalities. Our test platform provides various logging capabilities that could be used to evaluate services implementation through the whole development cycle. The data that we considered relevant to our testbed architecture are the vehicles' positions, network topologies, and delay time data.

First, we collected the different network round trip delay on both scenarios network architectures to evaluate the accuracy of our modeled virtual environment network. ***Figure.5.a*** shows the collected round trip delay values for the first scenario using only

Cloud host architecture. As shown in the Figure, first, the delay is higher than the configured delay parameter, then it remains stable at the theoretical value (20 ms). ***Figure.5.b*** shows the round trip delay value collected on the second scenario, where we have a mixed MEC Cloud network topology. The collected results show that the round trip delay between vehicle and MEC host is very close to the configured value (20 ms).



a. Scenario 1



b. Scenario 2

Figure.6 *Different link types round trip delay*

Next, we took mobility snapshots from our test scenario in order to verify that the testbed creates links between nodes in the range of each host accurately. As previously mentioned in *Table 1*, we implemented a Gauss-Markov mobility model to evaluate our scenarios. ***Figure.6*** shows a snapshot of the network configurations took from random scenarios' execution at random time steps. ***Figure.6.a*** shows a sample of the collected topologies logs during the first scenario execution. We see that using a single edge host with a wide range result a persistent communication for a long time between the Cloud host and the in range vehicle. In some topologies, some nodes are not connected to the edge host because they are out of the communication range.

We intentionally deployed a scenario where the edge host range does not cover the whole simulation area to test the association control manager. In ***Figure.6.b,*** we notice that the association control manager successfully changes the network links of vehicles regarding their distance to the MEC host. A new link is created/destroyed each time a vehicle enters/leaves the communication range of one of the scenario hosts. Some vehicles are disconnected from both MEC hosts at some time stamps because they are out of the range of both hosts. Table 1 also shows the placement of each microservice at each simulation setup. At the experimentation runtime, we verified that each microservice is deployed at the specified host.

VII. Conclusion and future work

With the aim of offering a prototyping environment to address V2X MEC application's developments and its evaluation process, we provide an architecture of a testbed for this purpose. In order to provide such a tool, in this paper, we demonstrated that this task could be achieved through the extension of existing network emulation and software virtualization tools. The proposed testbed offers a flexible and easy to use tool to model the MEC V2X services deployment environment. The tool usage could be customized according to the user through the exploitation of execution log records. On the one hand, user evaluation could be more oriented to a performance analysis by evaluating the communications delay, and the hosts' load. On the other hand, user evaluation could be more oriented towards the validation of the application's behavior and interactions.

Our future work will focus on offering a more accurate network model that supports a more realistic cellular network propagation and handover model. Further, we want to investigate a way to use our platform logging capabilities with an automated log analysis tool to automatically validate the execution results of full-service implementation under various scenarios.

	Scenario 1	Scenario 2
CPU	Intel(R) Core(TM) i7-4750HQ CPU CPU(s): 8 Thread(s) per core: 2 Core(s) per socket: 4 Frequency: 2.00GHz Max frequency: 3.2 GHz	Intel(R) Core(TM) i7-4750HQ CPU CPU(s): 8 Thread(s) per core: 2 Core(s) per socket: 4 Frequency: 2.00GHz Max frequency: 3.2 GHz
RAM	16 GB RAM Speed: 1600 MT/s	16 GB RAM Speed: 1600 MT/s
Allocated resources per host (CPU in cpu numbers) (Memory in Megabyte)	Vehicles: {"cpu": 0.25, "memory": 64} Cloud: {"cpu": 2, "memory": 2048}	Vehicles: {"cpu": 0.25, "memory": 64} Cloud: {"cpu": 2, "memory": 2048} MEC: {"cpu": 1, "memory": 512}
Link delay	Vehicle-Cloud: 100 ms Vehicle-Vehicle: 10 ms	Vehicle-MEC: 10 ms Vehicle-Vehicle: 10 ms Cloud-MEC: 50 ms
Simulation parameters	**Mobility:** Gauss-Markov model velocity_mean =3 0 alpha = 0.9 variance=0.5 Dimension = (300, 10) Number of nodes = 10 Number of Cloud hosts = 1 Cloud host position = (100, 5) **Association control model:** Algebraic distance communication range = 100	**Mobility:** Gauss-Markov model velocity_mean = 30 alpha = 0.9 variance = 0.5 Dimension = (300, 10) Number of nodes = 10 Number of MEC hosts = 2 MEC host 1 position = (0, 5) MEC host 1 position = (0, 100) **Association control model:** Algebraic distance communication range = 50
Microservices placement	Everything on the Cloud host	**MEC host:** Registry. Traffic collection microservice. Notification microservice

Table.1 Simulation Setup configuration

References

[1] F. Ana Juan, M. Joan Manuel, and J. Josep. 2019. Towards the Decentralised Cloud: Survey on Approaches and Challenges for Mobile, Ad hoc, and Edge Computing. ACM Comput. Surv. 51, 6, Article 111 (January 2019), 36 pages.

[2] S. Maheshwari, D. Raychaudhuri, I. Seskar, and F. Bronzino, "Scalability and Performance Evaluation of Edge Cloud Systems for Latency Constrained Applications," 2018 IEEE/ACM Symposium on Edge Computing (SEC), Seattle, WA, 2018, pp. 286-299.

[3] D. Sabella, A. Vaillant, P. Kuure, U. Rauschenbach and F. Giust, "Mobile-Edge Computing Architecture: The role of MEC in the Internet of Things," in *IEEE Consumer Electronics Magazine*, vol. 5, no. 4, pp. 84-91, Oct. 2016.

[4] H. Truong and M. Karan, "Analytics of Performance and Data Quality for Mobile Edge Cloud Applications," 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), San Francisco, CA, 2018, pp. 660-667.

[5] M. Emara, M. C. Filippou and D. Sabella, "MEC-Assisted End-to-End Latency Evaluations for C-V2X Communications," 2018 European Conference on Networks and Communications (EuCNC), Ljubljana, Slovenia, 2018, pp. 1-9.

[6] S. Zhou, P. P. Netalkar, Y. Chang, Y. Xu and J. Chao, "The MEC-Based Architecture Design for Low-Latency and Fast Hand-Off Vehicular Networking," *2018 IEEE 88th Vehicular Technology Conference (VTC-Fall)*, Chicago, IL, USA, 2018, pp. 1-7.

[7] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta and D. Sabella, "On Multi-Access Edge Computing: A Survey of the Emerging 5G Network Edge Cloud Architecture and Orchestration," in IEEE Communications Surveys & Tutorials, vol. 19, no. 3, pp. 1657-1681, third quarter of 2017.

[8] Y. Cao and Y. Chen, "QoE-based node selection strategy for edge computing enabled Internet-of-Vehicles (EC-IoV)," 2017 IEEE Visual Communications and Image Processing (VCIP), St. Petersburg, FL, 2017, pp. 1-4.

[9] ETSI MEC ISG, “Mobile Edge Computing (MEC); Technical Requirements,” ETSI, DGS MEC 002, 2016.

[10] ETSI MEC ISG, “Mobile Edge Computing (MEC); General principles for Mobile Edge Service APIs,” ETSI, DGS MEC 009, July 2017.

[11] ETSI MEC ISG, “Mobile Edge Computing (MEC); Study on MEC Support for V2X Use Cases,” ETSI, DGS MEC 009, September 2018.

[12] S. Dario, S. L. Vadim, T. Linh, K. Sami, P. Pietro, R. Ralf, XinhuiLi, F. Yong gang, D. Dan, G. Fabio, C. Luca, F.Walter, P. Bob, and H. Shlomi, ETSI White Paper No. 20 : Developing Software for Multi-Access Edge Computing, February 2019.

[13] G. Slawomir, "Next generation ITS implementation aspects in 5G wireless communication network," 2017 15th International Conference on ITS Telecommunications (ITST), Warsaw, 2017, pp. 1-7.

[14] 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; Study on LTE support for Vehicle to Everything (V2X) services (Release 14).

[15] ETSI GS MEC 003 V1.1.1. Mobile Edge Computing (MEC); Framework and Reference Architecture, 2016.

[16] Mininet, “project home page”, Available Online at: http://mininet.org/

[17] L. Bob, H. Brandon, and M. Nick. A Network in a Laptop: Rapid Prototyping for Software-Defined Networks. 9th ACM Workshop on Hot Topics in Networks, October 20-21, 2010, Monterey, CA.

[18] Y. Lisa, M. Nick. Learning Networking by Reproducing Research Results. SIGCOMM CCR, April 2017.

[19] H. Nikhil, H. Brandon, J. Vimal, L. Bob, and M. Nick. Reproducible Network Experiments using Container-Based Emulation. CoNEXT 2012, December 10-13, 2012, Nice, France

[20] Docker, “project home page”, Available Online at https://www.docker.com