SMOTEC
(An Edge Computing Testbed for Adaptive Smart Mobility Experimentation)
Documentation

Version 0.1

Zeinab Nezami

July 20, 2023

Contents

1 Introduction

SMOTEC is an open-source edge computing testbed to support smart mobility experimentation. The testbed is a distributed system with collaborating components including Testbed, Edge Agent, Mobile Agent, Service, and Service Distributor, and relies on open-source technologies such as K3s, Docker, ZeroMQ, EPOS, Grafana/Prometheus, SUMO, and Java.

This documentation guides users and developers to conduct smart mobility experimentation with SMOTEC and extend it. This tutorial is organized as follows: Section 2 presents the software and hardware requirements for SMOTEC system. Sections 3 and 4 describe how to set up SMOTEC using pre-configured virtual machines and in a native Linux environment, respectively. These sections also guides users to run a default traffic monitoring experiment in the set up environment. Section 5 presents the implementation details of the proposed edge computing testbed including source code and Docker images and discuss how to extend the testbed. The challenges user might encounter during the testbed set up procedure are presented in Section 6.

2 Requirements

SMOTEC uses K3s, a light Kubernetes distribution, as its management layer which keeps a comprehensive view on available edge resources and running services. The K3s cluster for running SMOTEC includes at least one master node, in this documentation referred to as *orchestrator*, and at least two worker nodes. The orchestrator is defined as a host running the k3s server, with control-plane and datastore components managed by K3s. the worker node is defined as a host running the k3s agent, without any datastore or control-plane components.

K3s is available for a variety of architectures including x86_64 and arm64 which makes it suitable for resource-constrained environments such as Internet of Things (IoT) and edge computing. Each node running K3s server should meet the following minimum requirements. These requirements are baseline for K3s and its packaged components, and may increase as the resources consumed by the user's workload scale. The orchestrator has a small footprint (Minimum: 1 or 2 cores of CPU and 512 MB to 1 GB RAM) that can be as light as a Raspberry Pi. More information are available on the official page of K3s[1]. In addition, to run smart mobility experiments with

---

[1]K3s. Available at `https://docs.k3s.io/installation/requirements` (Accessed 18 July 2023).

SMOTEC you need Java. Make sure that Java JDK version 17 or higher is installed on your system. You can download it from Oracle website[2].

3 Step by Step Guide: Virtualized Environment

This section illustrates the procedure to run and configure a three-node SMOTEC testbed using the virtual machines that we have generated for the testbed. In this method, there is no need to install K3s and software packages. The edge environment is pre-configured and ready to run the SMOTEC traffic monitoring applications.

3.1 Preparation

There are three virtual machines (VM) in our Zenodo repository available at `https://doi.org/10.5281/zenodo.8167775`: Master-K3S is the orchestrator machine (K3s Master node) and two others are worker machines. User needs to import the machines to a virtualization tool such as VirtualBox. The procedure to follow for this purpose is presented below.

1. Install virtualBox 7.0.6 or later versions from `https://www.virtualbox.org/wiki/Downloads`.

2. In VirtualBox, as shown in Figure 1, go to tools menu, in the tab Host-Only Networks create a Host-Only Ethernet Adaptor with the IP range 192.168.58.1/24 and Net Mask 255.255.255.0, and then apply the changes.

3. Import all the VMs to VirtualBox by going to File, Import Appliance menu. In the new window of Import Virtual Appliance select one of the VM you downloaded. Repeat this step for the three VMs.

4. Once the import is finished and before starting the machines, you should assign the Adapter created in step 2 to the machines imported to make them available to the K3s cluster. To do this for every machine do as follows:

   (a) select the machine in the virtualBox machine menu, go to the setting tab for that machine, then go to the Network tab.

   (b) Select Adapter1 and attach it to NAT.

   (c) Select Adapter2 and attach it to the the created Adapter in step 2.

---

[2]Oracle Java. `https://www.oracle.com/java/technologies/downloads/` (Accessed 18 July 2023).

The NAT adapter enables virtual machines to connect to the Internet for downloading Docker images from remote repository. In case you use Adapters with a different range of IP addresses you will need to go through the installation of K3s described in section 4 after finishing this procedure.

Now that you assigned adapters to all machines they will be ready to start. You may continue using next section to run the SMOTEC experiment.
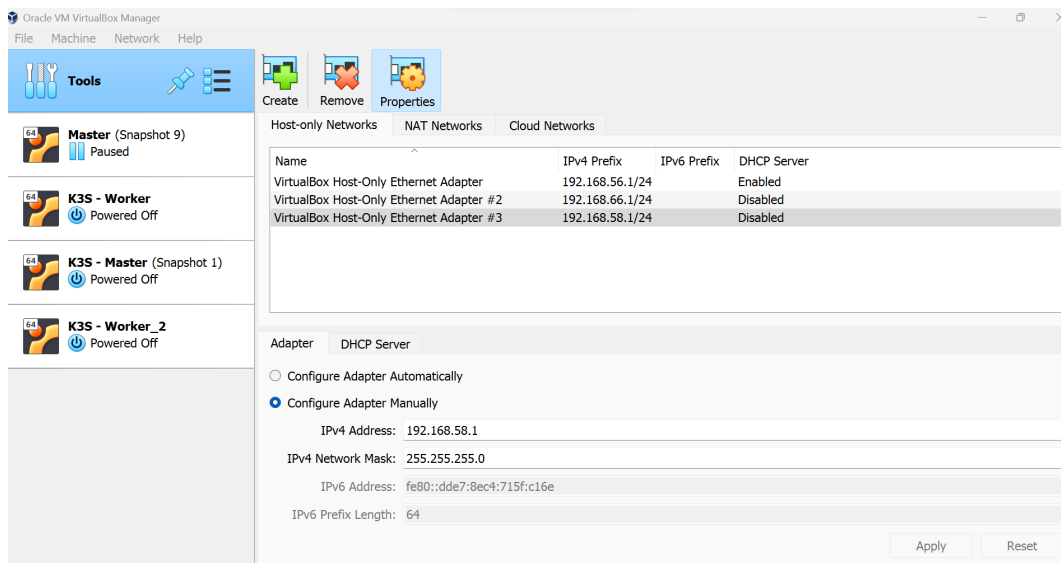

Figure 1: Creation Host-Only Adapter in VirtualBox.

Users may configure more virtual machines and join them to the cluster as worker (edge) nodes. To this end, create a new virtual machine with the hardware and software requirements mentioned in section 2. Then assign a Host-Only and a NAT adapter to it as described in this section. Then follow the instructions of section 4.2 to scale the virtual machine setup by adding the new machine.

3.2   SMOTEC traffic monitoring experimentation

This section outlines the steps you need to run the traffic monitoring application defined by testbed configuration file.

Start all the three VMs imported in the previous section. Use the login credentials listed in Table 1 to login to the machines. To reduce the space required for the machines, a minimal server version of Ubuntu 22.04.2 LTS is already installed on the virtual machines. To run the traffic monitoring

Table 1: Specification of hardware and software platform used in SMOTEC virtual machines.

Characteristic	Virtual Machine		
	Ochestrator	Worker3	Worker2
root user	root/orchestrator	root	root
root password	123456	123456	123456
IP Host-Only	192.168.58.60	192.168.58.40	192.168.58.50
Operating system	Ubuntu 22.04.2 LTS	Ubuntu 22.04.2 LTS	Ubuntu 22.04.2 LTS
RAM	4GB	2GB	2GB
DisK Space	5GB	5GB	5GB
Node label	master	worker1	worker2

experiment available in the orchestrator (Master-K3S) you need to login to the orchestrator as a root user and do the following steps.

- Open terminal in orchestrator, and then make sure K3s service is active and running using the following command:

```
systemctl status k3s.service
```

- Make sure the two worker (edge) nodes are connected and available to the K3s cluster by getting the list of available nodes in your cluster:

```
kubectl get nodes
```

As a result of the previous command you should see the list of the three machines in a ready state.

- Run the traffic monitoring experiment by navigating to the Testbed directory in /home/ochestrator/Testbed. Based on the default testbed configuration parameters of testbed in /Testbed/Conf/TestbedConfig.json, the experiment lasts for 600 seconds. There are two options to run the experiment:

    - (i) Execute the below command and wait until the experiment finishes. When the experiment time is over, the message "End of experiment!" is shown on the terminal.

    ```
    java -jar Testbed.jar
    ```

    - (ii) Direct the terminal output to a file, for example out.txt, and run the program in background by executing the java command using & symbol as follows:

    ```
    java -jar Testbed.jar >out.txt &
    ```

3.2.1 Configuration parameters

SMOTEC uses a JSON file, TestbedConfig.json, as its configuration file which defines an SMOTEC experiment using the parameters listed in Table 2. The configuration parameters are then stored in a Kubernetes ConfigMap object as key value pairs and will be available to the SMOTEC components. Users can find the configuration file in /home/orchestrator/Testbed/Conf on the orchestrator node. Parameters in this file are set based on a 3-node edge infrastructure including one orchestrator and two edge (worker) nodes, and 10 mobile nodes. You may change the values based on the settings and requirements of your experiment such as the number of edge nodes and their characteristics (e.g., location, coverage range, processing power, and memory), the number of mobile devices and their service requirements (e.g., processing power and memory), name and path to the testbed container image repository, EPOS simulation settings, etc. After altering the default values save the file and run the experiment using the configuration JSON file as the input to java command:

```
java -jar Testbed.jar
 /home/orchestrator/Testbed/Conf/TestbedConfig.json
```

If you want to use your own configuration file you can save your JSON file in the orchestrator machine and give the path to it as an input to the java run command as follows:

```
java -jar Testbed.jar <path to your config JSON file>
```

For more information on ConfigMap refer to the Kubernetes web page at https://kubernetes.io/docs/concepts/configuration/configmap/.

3.3 Monitoring of SMOTEC experiment

Users can use the three methods described in this section to view the results and output of their experimentation with SMOTEC:

To have an overview of the status of SMOTEC pods and services run the following command in the terminal during the experiment life-time:

```
kubectl get pods -n smotec -o wide
```

As a result the list of running pods with the edge node they are running on will be shown on the terminal. To see the terminal output of a pod execute the logs command with the name of pod as the input:

```
$kubectl logs <PodName> -n smotec
```

Table 2: Configuration parameters for running an experiment with SMOTEC.

Parameter Name	Description	Value Type
Max_x	X-Coordinate of top right corner of a rectangular test area	Integer
Min_x	X-Coordinate of bottom right corner of a rectangular test area	Integer
Max_y	Y-Coordinate of top left corner of a rectangular test area	Integer
Min_y	Y-Coordinate of bottom left corner of a rectangular test area	Integer
NumMobileAgents	Number of mobile agents (e.g., vehicles) in the area	Integer
AgentId	The Id of the mobile agent (same with the name of mobility profiles)	Array of Integers
CPUResourceDemand	CPU demand for requested services of mobile agents in MIPS	Array of Integers
MemResourceDemand	Mamory demand for requested services of mobile agents in MB	Array of Integers
StoResourceDemand	Storage demand for requested services of mobile agents in MB	Array of Integers
AgentMobilProfile	Directory of mobility dataset for mobile agents	String
Secret	Image repository secret key	String
ContainerImagesPath	Path to the repository containing Docker images	String
Orchestrator	Label of master node	String
NumEdgeNodes	Number of edge nodes	Integer
EdgeID	Id of edge nodes	Array of Integers
NodeLabel	Label of edge nodes	Array of Strings
AP-COVERAGE	Coverage range of edge nodes	Integer
Xpoints	X-Coordinates of edge nodes location	Array of Integers
Ypoints	Y-Coordinate of edge nodes location	Array of Integers
CpuCap	CPU capacity of edge nodes in MIPS	Array of Integers
MemCap	Memory capacity of edge nodes in MB	Array of Integers
StorageCap	Storage capacity of edge nodes in MB	Array of Integers
EPOSNumPlans	Number of service placement plans generated by every edge node	Integer
EPOSplanDim	Dimension of EPOS plans	Integer
EPOSnumSimulations	Number of simulations of EPOS	Integer
EPOSnumIterations	Number of iterations of EPOS	Integer
SrvDisLi	Service distributor listening port for service placement plans	String
SrvDisRe	Service distributor response port for selected service placement plans	String
ServiceName	Name of user service image	String
ServiceDistributorImage	Name of service distributor image	String
EdgeAgentImage	Name of edge agent image	String
MobileAgentImage	Name of mobile agent image	String
ConfigMap	Name of testbed configuration map	String
ExperimentTime	Time duration of running an experiment in second	Integer

We configured Grafana/Prometheus monitoring tools and also provided multiple Grafana dashboards in the virtual machines so that you can directly go to the URL `192.168.58.60:3000` in your local host for instant visualization during the experimentation. Use the username "admin" and password "123456" to login to Grafana. Go to Home, Dashboards, and then General, you will see three dashboards: Kubernetes Nodes, Kubernetes Cluster (Prometheus), and Node Exporter Full. For an instance, in Node Exporter Full dashboard you can see the resource utilization of cluster nodes such as CPU, memory, disk, and network. You can select the node you want to monitor in this dashboard by going to "job" dropbox and selecting one of the following values for the nodes: "node_exporter" for the master node, "node_exporter_worker_2", and "node_exporter_worker_3" for the worker nodes. This dashboard together with the two others give you total visibility over all edge devices in your infrastructure and the containers running on them.

In addition to the real-time monitoring of edge resources, an output directory will be available after successful completion of experiment in the directory /root/Documents/output on orchestrator. To check the results, the output directory may be downloaded using secure copy (scp) to your local machine by running the following commands on the terminal of your host machine:

```
scp -r root@192.168.58.60:/root/Documents/output
<path to your local directory>
scp root@192.168.58.60:/home/orchestrator/Testbed/out.txt
<path to your local directory>
```

The output directory contains:

- The output of service distributor module is available in "epos-out" directory.

- CPU and Memory utilization of edge nodes in ".csv" format.

- A copy of service placement plans selected by EPOS for every edge node in "S-plans" directory.

- Service deployments and releasing of resources performed by SMOTEC and K3s are listed in the "deprel" directory, with the corresponding deployment files in the "deployment" directory.

- Datasets of service placement plans generated by every edge node is available in "Datasets" directory.

4   Step by Step Guide: Native User Environment

The steps below illustrate the procedure to install and configure SMOTEC in a Linux environment. The Testbed uses K3s, a light-weight Kubernetes distribution, as its orchestration layer. To set up the testbed, user must configure a Kubernetes cluster (K8s or K3s) with at least one orchestrator node and two worker nodes (user can add as many worker nodes as she wants), then, add labels to the nodes in your cluster and run SMOTEC experimentation.

We built SMOTEC using low-cost Raspberry Pis Model 4B with Broadcom BCM2711, Quad core Cortex-A72 64-bit SoC 1.5GHz, 8GB RAM, and 128GB Storage, which can be deployed easily at large-scale. Raspberry Pis are connected to the orchestrator by joining the K3s cluster as worker nodes. A TP-Link 16-Port Desktop Gigabit Ethernet Switch is used to connect the Raspberry Pis to the access network via Cat 6 Ethernet cables. This setup procedure is the same for Desktop machines, Raspberry Pis, and Linux servers. The steps mentioned under orchestrator are to be executed on the orchestrator and the steps mentioned under worker nodes are to be executed in the worker nodes environment otherwise explicitly stated. For this procedure all the cluster nodes should have a static IP address referred to as orchestrator_ip and worker_ip.

4.1   Orchesterator setup

To set up a K3s cluster of orchestrator (master) and worker nodes execute the commands below in the terminal of your orchestrator node. Based on your machine architecture the installation can be different. Refer to the K3s documentation at https://docs.k3s.io/installation/configuration for more detailed information. Execute below command to create a K3s cluster on the orchestrator node:
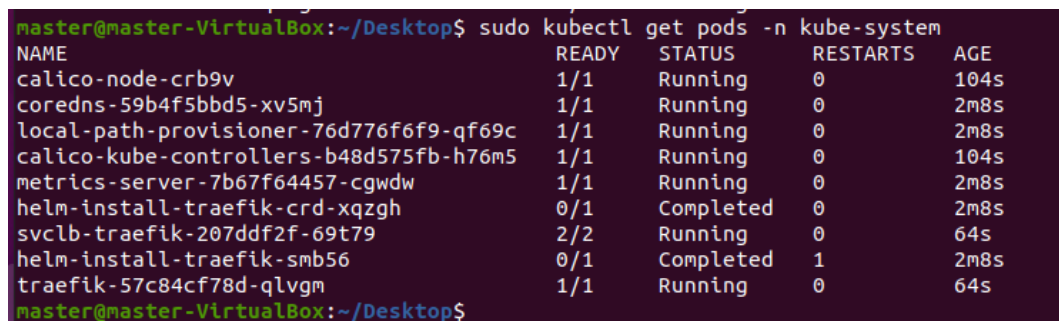
```
curl -sfL https://get.k3s.io | INSTALL_K3S_EXEC=
"--flannel-backend=none --disable-network-policy
--cluster-cidr=192.168.0.0/16
--node-ip=<orchestrator_ip>" sh —
```

, where cluster-cidr is the range of IP addresses Kubernetes assigns to each node and orchestrator_ip is the static IP address of orchestrator. We use Calico to provide networking for pods and services. Calico creates a flat layer-3 network and assigns a fully routable IP address to every pod. To configure the network manager execute:

```
Sudo kubectl apply -f
https://raw.githubusercontent.com/projectcalico/
calico/v3.26.0/manifests/calico.yaml
```

Run next command and wait until all the listed Kubernetes system pods are in "running/completed" status and have "1/1" or "2/2" in READY column, as shown in Figure 2:

```
sudo kubectl get pods -n kube-system
```


Figure 2: Kubernetes system pods running.

Make sure the K3s service is active and the orchestrator node is ready by running the following commands:

```
sudo systemctl status k3s.service
sudo kubectl get nodes -o wide
```

SMOTEC calls Kubernetes APIs without sudo permission. For running the Testbed experiments you need to temporarily or permanently remove the sudo permission using the options below.

- To remove the sudo permissions temporarily execute the following command:

  ```
  sudo chmod 644 /etc/rancher/k3s/k3s.yaml
  ```

- To remove the sudo permissions permanently, automate the chmod command at boot-time by putting it in your bashrc or bash_profile. Bashrc file is a hidden file inside your home directory. Execute the following command to open the file:

  ```
  vi ~/.bashrc
  ```

  Then add the chmod command to the end of this file, save it and restart your machine. The next time you log into the machine, you can run kubectl commands without sudo permission.

Run the following command and copy the displayed k3s token for use in worker nodes. You can use scp command to send the token to your worker nodes.

```
sudo nano /var/lib/rancher/k3s/server/node-token
```

4.2 Worker nodes setup

To connect an edge (worker) node to the orchestrator, execute the command below on the worker node to install K3s agent on it. The following commands must be run on every worker (edge) node you want to join your edge infrastructure.

```
curl -sfL https://get.k3s.io | INSTALL_K3S_EXEC=
"--node-ip=<worker_ip>"
K3S_URL=https://orchestrator_ip:6443
K3S_TOKEN="TOKEN" sh -
```

Where, worker_ip is the static IP address of the worker node the command is running on, orchestrator_ip is the IP address of the orchestrator and Token is the token value stored at /var/lib/rancher/k3s/server/node-token on orchestrator. Check if the K3s agent is active and running on the worker node:

```
sudo systemctl status K3s-agent
```

After setting up K3s on your edge nodes, execute the following command on the orchestrator to check if the worker nodes are joined the cluster.

```
kubectl get nodes -o wide
```

ALL the following commands in this section are run on the orchestrator machine. K3s has a flag called "−node-label" in order to add labels to its cluster nodes. Now that all the nodes are ready we label them so that they can be uniquely identified by SMOTEC for allocating resources to its services and pods. SMOTEC uses the label/value pair with the label "nn" to deploy a container image on an edge node. Choose your nodes one by one, and add a label to them by running the following command:

```
kubectl label nodes <node-name> nn=<value>
```

, where the value is a string that uniquely identifies a node in the cluster, and node-name is the name of the node which can be provided using the command below.

```
kubectl get nodes -o wide
```

Finally, verify if the labels are applied:

```
kubectl get nodes --show-labels
```

For more information on Kubernetes labels refer to Kubernetes reference page at https://kubernetes.io/docs/tasks/configure-pod-container/assign-pods-nodes/.

4.3 SMOTEC traffic monitoring experimentation

Download the Testbed module and output directory available on SMOTEC GitHub repository at `https://github.com/DISC-Systems-Lab/SMOTEC`, and save them in directory /home/orchestrator and /root/Documents, respectively. A TestbedConfig.json file located in the Testbed/Conf directory defines a SMOTEC traffic monitoring experiment using the parameters listed in Table 2. The default configuration values are for a 3-node edge infrastructure including two edge nodes and one orchestrator node, and 10 mobile nodes. You may change the values based on the requirements of your experiment.

Edit the configuration file according to the K3s labels you added to your cluster nodes: (i) "Orchestrator" parameter indicates the orchestrator node label, (ii) "node labels" indicates the labels of worker nodes, (iii) "ExperimentTime" identifies the experiment time in second with the default value of 600 seconds. Change the value of those parameters based on your settings and the requirements of your experiment. If you wish to adapt the new configuration, run the experiment by navigating to the Testbed directory /home/ochestrator/Testbed in orchestrator terminal.

To run the experiment you can execute the below command and wait until the experiment finishes and the message "End of experiment!" is shown on the terminal.

```
java -jar Testbed.jar <path to the TestbedConfig.json>
```

You can also append & to the run command and continue using the terminal as follows. Be aware that you will not see text output to the terminal, such as error messages as the terminal output is now directed to a file (out.txt).

```
java -jar Testbed.jar <path to the TestbedConfig.json >out.txt &
```

4.4 SMOTEC Monitoring Configuration and Output

This section describes the methods to make sure SMOTEC does the job it has been designed to do. SMOTEC uses its own Kubernetes namespace "smotec" for running its containers in an isolated environment within the K3s cluster. To have an overview of the status of services and pods running in smotec namespace execute the following command in the orchestrator terminal during the experiment life-time:

```
kubectl get pods -n smotec -o wide
```

As a result a list of running pods, their status and the edge node they are running on will be shown on the terminal. To see the terminal output of a pod, choose your pod from the list of pods and execute logs command with the name of that pod as follows:

```
$kubectl logs <PodName> -n smotec
```

After running an experiment in SMOTEC, the output results will be available in root/Documents/output directory. It contains:

- The output of service distributor module is available in "epos-out" directory.

- CPU and Memory utilization of edge nodes are available in ".csv" format.

- A copy of service placement plans selected by EPOS for every edge node is available in "S-plans".

- Service deployments and releasing of resources performed by SMOTEC and K3s are listed in the "deprel" directory, with the corresponding deployment files in the "deployment" directory.

- Datasets of service placement plans generated by every edge node is in "Datasets" directory.

We recommend users to config Grafana/Prometheus monitoring tools on the cluster machines to have a comprehensive visualization of edge resources performance metrics such as CPU, Memory, Disk, and network utilization during the experimentation with SMOTEC. For this purpose, users need to install Grafana on the orchestrator (or on the machines they want to monitor). In addition, Prometheus and node exporter should be installed on the orchestrator to visualize the metrics on Grafana. You can follow the steps in the tutorial available at `https://grafana.com/docs/grafana/latest/getting-started/get-started-grafana-prometheus/` to install Prometheus, Grafana and node exporter for exporting the system metrics. Make sure you use an appropriate release for your system architecture from `https://github.com/prometheus/prometheus/releases`.

5   Source Code and Docker Images

This section provides a description of SMOTEC source code and its corresponding Docker images available on GitHub at URL `https://github.com/DISC-Systems-Lab/SMOTEC` and Docker Hub at URL `https://hub.docker.com/repository/docker/zeinabne/smotec/general`, respectively. SMOTEC is a distributed edge computing system that consists of five collaborating modules including Testbed, Edge Agent (Docker image: edgeagent), Mobile Agent (Docker image: vehicleagent), Service Distributor (Docker image: servicdistributor), and Service (Docker image: traficservice).

The testbed provides services to the Mobile Agents in the form of networked Docker containers hosted on edge nodes and managed by the orchestrator (K3s master node). The SMOTEC components are outlined in the following subsections.

5.1   Testbed

Users interact with this module to define a smart mobility experimentation and run mobility service containers from pre-created Docker container images on an edge infrastructure. The SMOTEC configuration file TestbedConfig.json available in directory Testbed/Conf specifies the test environment and test application using the parameters listed in Teble 2. The Testbed life cycle includes two stages of preparation and execution, as shown in Figures 4 and 3. During the preparation stage, the Testbed performs as follows:

- Creates a namespace named "smotec" to provide an isolated group of resources for SMOTEC within the cluster.

- Creates a configmap object named "testbedconfig" from the configuration file for sharing the configuration parameters among all the pods running in the smotec namespace.

- Creates an edge infrastructure via calling K3s APIs and deploying one edgeagent image on every edge (K3s worker) node via the method create_infrastructure.

- Deploys one servicdistributor image and multiple vehicleagent images on the orchestrator (K3s master) node by calling create_vehicleagents method.

During the execution step, the realized pods from SMOTEC images start talking to each other using ZeroMQ messaging library. Vehicle agents send their service requests to the edge agents in close proximity. The edge agents cooperate with each other and decide on the hosts of the requested services through the service distributor. The Testbed program monitors the edge agents and service distributor and based on their resource allocation decisions calls K3s API for deploying/releasing traficservice images on the edge nodes for serving vehile agents.

When the experiment time is over, the program releases all resources allocated to the smotec namespace and generates output files for investigating how the resources are allocated during the experiment.
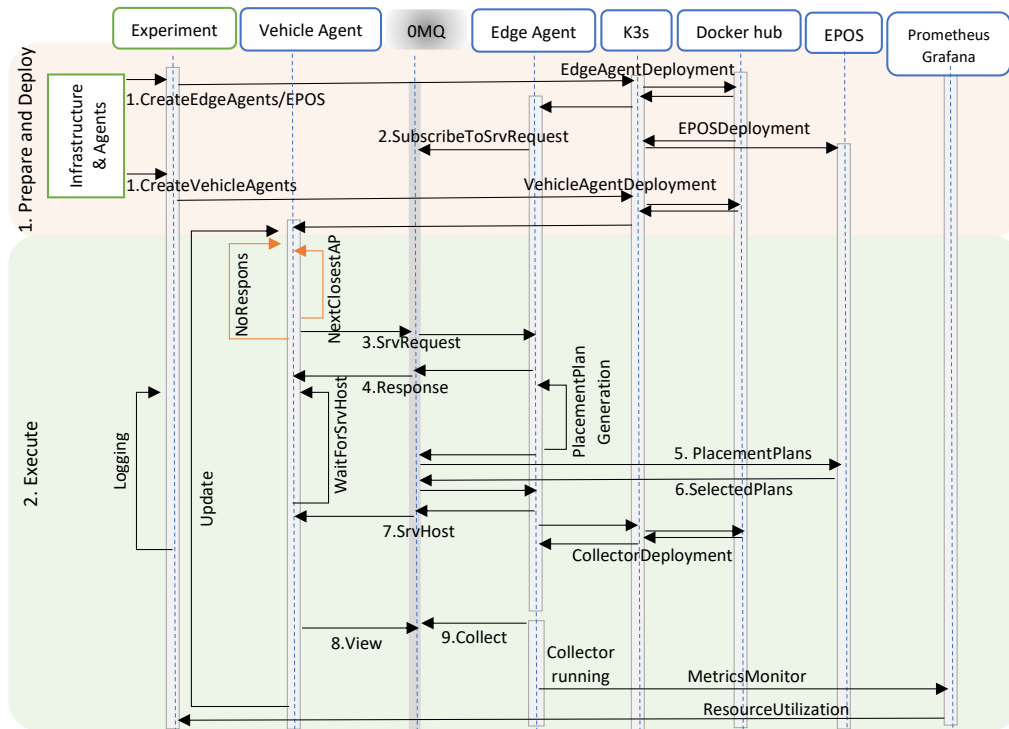
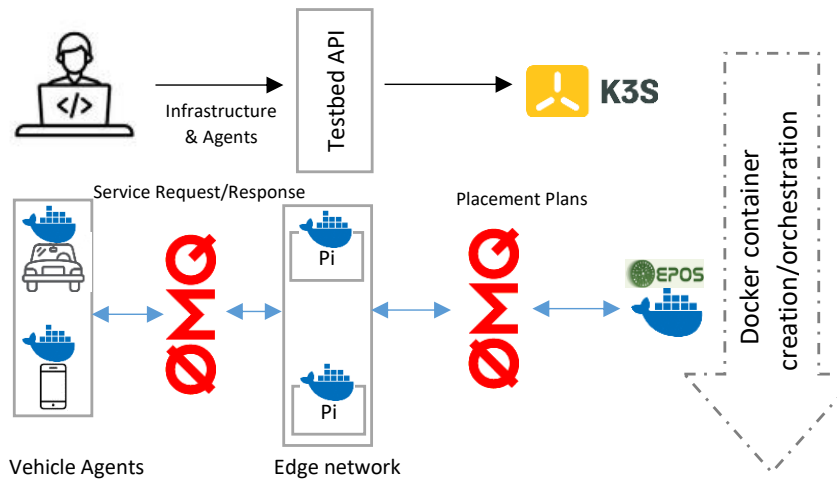Figure 3: Traffic monitoring application workflow

Figure 4: Real-time traffic monitoring service architecture corresponding to Figure 3

5.2 Edge Agent

This program runs on edge nodes and integrates them with the SMOTEC testbed. Once deployed, the agent manages the communication and computation tasks of its host node as an edge node. Some classes of this agent that are responsible for the main functionalities of this agent are outlined below.

- A "Server" class listens to the requests from mobile clients (vehicle agents) and responds appropriately. The request can be of different types including connection (service), handover, and down request.

- A "ResourceMgm" class manages all the requests received from mobile clients and decide on the allocations and dealocations of edge resources to the service requests. Regarding the service requests, "SrvMapper" class generates a certain number of "requests to resources" mappings named service placement plan with respect to the requests resource demands of processing power and memory.

- A "PlanDistributor" class sends the placement plans to the SMOTEC service distributor for selecting the optimized plan among them with respect to the system-wide balancing of edge resources utilization.

Upon receiving a response from service distributor which contains one selected plan for every edge agent, the agent informs its mobile clients of the name of their service host from which they can receive their traffic monitoring service. At the same time, the agent generates Kubernetes deployment files for service placement procedure directed by Testbed module and K3s.

Apart from the service requests, this agent handles the handover and down requests, and migration of services. In addition, the edge agent keeps a record of the number of mobile agents connected to it (moving vehicles around) and propagates this information to the traffic monitoring services hosted on itself using a "TrafficMonitorPublisher" module which implements a Callable interface.

5.3 Mobile Agent

It is an agent that installs on a mobile device in the form of a docker image, vehicleagent, and enables it to communicate with SMOTEC (send/receive messages) and receive smart mobility (e.g., traffic monitoring) services. This agent is in charge of sensing the location of its mobile host and creating/removing communication links with edge nodes in its close proximity.

Table 3: Format of mobility profile for a mobile device in SMOTEC.

TimeStamp	Angle	X-coordinate	Y-coordinate	Speed m/s
1	11.27	13957.68	15221.41	11.54

SMOTEC leverages SUMO traffic simulator [3] as its mobility service provider. A real-world map of Munich city is imported to the SUMO simulator to accurately simulate realistic vehicle movements. The developed "Mobility" API in Mobile Agent enables it to implement different mobility models. The Mobility module parses the mobility database of SUMO stored in a csv format. As illustrated in Table 3, each csv file represents the mobility profile of a vehicle as a sequence of traversed points during simulation. Each point includes agent's x and y coordinates on the map, its speed in meters per second, the direction in radiant, and the time in which this data is collected.

With the passage of time, the location of the mobile agent is updated and it finds the closest edge devices to itself via "Distance" class. The "Client" class which is in charge of creating/removing communication links with the edge network, sends connection request to the closest edge agents until it receives a response for its request. The request defines the resource demands for the mobile agent service including processing power, memory, and storage. Upon receiving a response from the edge network containing the address of its service host, the "View" module which implements a callable interface starts communicating with the service host and receiving traffic updates.
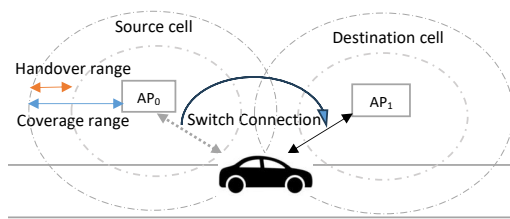

Figure 5: Handover procedure during vehicle movement.

With the movement of the mobile agent, when the agent is in a certain distance from its connected edge node (handover range), its data transmission should be switched to the next closest edge node, shown in Figure 5. In this case, the "Client" sends a connection request to the next closest edge nodes and a disconnection request to its current connected edge node. After receiving acknowledgements from the next edge agent the connection is handed over and the agent data is transferred from

[3]SUMO. Avaliable at https://sumo.dlr.de/docs/index.html (Accessed 17 July 2023).

one edge node to another without losing connectivity to the network.

5.3.1 How to update mobility profiles

This section explains how to modify the default mobility profiles of mobile agents, test area, and number of vehicle agents based on the requirements of your experiment. For this purpose, you must update the profiles in Mobile Agent source code, dockerize it, and finally update TestbedConfig.json file according to the changes.

We have used real world traffic scenario for 10 vehicles moving in Munich city as the default SMOTEC experimentation. In default configuration of SMOTEC we consider an area of Munich city downtown identified by four GPS coordinates as the corners of our rectangular test area: [48.1534, 11.5525], [48.1417, 11.5533], [48.1413, 11.5810], [48.1541, 11.5812] and two edge nodes located at [11.5527,48.1534] and [11.5805,48.1415]. These locations are converted to the Cartesian coordinates using SUMO simulator and then given to the testbed via TestbedConfig.json file (min_x, max_x, min_y, max_y).

The mobility profile of mobile agents are extracted from FCD output file of SUMO[4]. For every vehicle, the mobility profile is a sequence of mobility characteristics listed in Table 3. After generating your mobility profiles according to the format mentioned in Table 3, you can assigne them to the mobile (vehicle) agents as follows. Download the Mobile Agent source code from SMOTEC GitHub repository. Go to the mobility_dataset directory in the src directory, inside it create a subdirectory with your intended name and save the csv mobility profiles there. After that, package the source code as a jar file named "MobileAgent" and include the external libraries in the jar file.

Now that the new mobility profiles are added to the mobile agent package, you must generate a docker image from the new jar file and push it to your image repository to make it available to the testbed. In case you use a Docker Hub repository as your image repository, the following commands are applicable to create a Docker image, tag, and push it to Docker Hub. For more detailed information refer to Docker reference page at `https://docs.docker.com/engine/reference/commandline/build/`.

Navigate to the root directory of Mobile Agent, there is a Docker file named "Dockerfile" without any extension in this directory with the below contents. The listed commands are called on the command line to assemble an image from the MobileAgent.jar. The five values listed in CMD command are the input arguments which will be used when the vehicleagent image is realized. These values are optional as SMOTEC uses a JSON configuration file and

---

[4]SUMO FCDOutput. Available at `https://sumo.dlr.de/docs/Simulation/Output/FCDOutput.html` (Accessed 19 July 2023).

Kubernetes deployment files to pass the required parameters to its images at run-time.

```
FROM openjdk:17
COPY ./MobileAgent.jar /tmp
COPY ./src/conf/ /tmp/src/conf/
COPY ./src/Mobility_Dataset/ /tmp/src/Mobility_Dataset/
WORKDIR /tmp
ENTRYPOINT ["java","-jar","MobileAgent.jar"]
CMD ["10001","200","100","40","subdirectory"]
```

Run the following docker commands to generate a Docker image from the jar file using the Docker file:

```
$docker build -t <image> .
$docker run -t -i —name <containername> <image>
$docker commit <containername> <exportedimagename>
$docker tag <exportedimagename> <your_repository>:vehicleagent
$docker push <your_repository>:vehicleagent
```

You can replace the parameters between "<>" symbols with any string. However, you must replace <your_repository> with the path of your image repository.

Then, alter the following parameters in configuration file of Testbed source code available at Testbed/Conf/TestbedConfig.json according to the changes you made:

- Replace the name of current mobility directory (default value is "1") with the new subdirectory name you created for your mobility profiles in parameter "AgentMobilProfile".

- Correct the name of the mobile agent image using the parameter "MobileAgentImage", if you used a name other than "vehicleagent" for your Docker image.

- Update the values of "NumMobileAgents", "AgentId", "CPUResourceDemand", "MemResourceDemand", and "StoResourceDemand" based on the number of mobile agents and their resource demands in your experiment.

- Update the value of "ContainerImagesPath" with the path to your Docker repository where you pushed the new image of mobile agent to.

- If your Docker image repository is private and requires authentication to enter, you must create a secret and specify it in the configuration file. To do this refer to the section 5.5.1.

If you consider to use a new location for your test area you must modify the infrastructure part of the TestbeConfig.json file as follows:

- Replace the default coordinates of SMOTEC test area with the coordinates of your area using "min_x", "max_x", "min_y", "max_y".

- Edit the location of the edge nodes in your test area using "Xpoints" and "Ypoints".

- Specify the number of edge nodes located in the new test area by "NumEdgeNodes". Edit the parameter "NodeLabel", "EdgeId", "Xpoints", "Ypoints", "CpuCap", "MemCap", "StorageCap" according to the characteristics of the new edge nodes.

5.4   Service: Collector

SMOTEC currently offers a traffic monitoring service named "Collector" as its smart mobility service. This service, available on Docker Hub as "traficservice", is deployed on the edge nodes located in different parts of Munich city based on service placement strategy of SMOTEC and receives traffic updates from its host node and publishes the updates to its subscribed users which are the vehicle agents connected to it.

5.5   Service Distributor: EPOS

Service distributor presents the proposed resource management scheme of SMOTEC. It is the service distributor that makes decision on computation offloading and the placement of users service container images that are awaiting resource allocation. SMOTEC uses this service distributor program as a wrapper for its placement policy, EPOS, which can be replaced by other placement policies. EPOS [5] is a decentralized multi-agent system for multi-objective combinatorial optimization. It balances the input workload across the edge network, while minimizing deadline violations, service deployment cost and services that do not meet hosting requirements.

The "Consumer" class in service distributor is continuously listening to receiving service placement plans from edge agents. After receiving the plans, EPOS is called to perform collective decision-making among edge agents that autonomously generate a set of service placement plans from which they make a choice such that their combination satisfies network-wide (e.g., minimizing over-utilized edge nodes) and individual objectives (e.g.,

---

[5]EPOS. Available at `https://github.com/epournaras/EPOS`, (Accessed 17 July 2023).

minimizing service execution cost). Finally, the edge agents get informed about the EPOS placement decisions via ZeroMQ messages generated by "Producer" class.

Configuration parameters for EPOS can be changed using SMOTEC Testbed-Config.json file and epos.properties file of Service distributor module available in its Conf directory. EPOS can be replaced with any other coordinator in the form of a Docker image without requiring further reprogramming. For more information about EPOS refer to EPOS Documentation at `https://epos-net.org/software_doc/documentation/experimantal-setup.html`.

5.5.1 How to use your private image repository

This section describes how to use your own Docker image repository when you replace current SMOTEC modules with new images or alter their functionalities. For this purpose there are a number of common steps you must follow for applying your changes to every module:

1. Download the source code of the SMOTEC modules from SMOTEC GitHub repository.

2. Investigate the source codes, make your desired changes.

3. Create Docker images from the modified modules using Docker and available Docker files in each module.

4. Push the Docker images to your repository.

5. Alter the SMOTEC configuration file based on the changes you made.

SMOTEC currently uses the publicly available Docker Hub repository at `https://hub.docker.com/repository/docker/zeinabne/smotec` to store and import its Docker images. Users can use their own images and image repository to test according to the requirements of their experiment. The alterations required to use the custom repository are explained below. To use your own private repository first create a secret to authorize access to your repository in K3s orchestrator. For this login to your repository in orchestrator terminal, in case it is Docker Hub run:

```
docker login -U username -p password
```

As a result you will have a config.json file in /root/.docker directory. Execute the following command to create a secret file containing your credentials using the json file:

```
kubectl create secret generic <SecretName>
--from-file=.dockerconfigjson=/root/.docker/config.json
--type=kubernetes.io/dockerconfigjson
```

, where SecretName is a string identifying the name of the secret. After Secret creation, edit TestbedConfig.json file accordingly by updating "ContainerImagesPath" with the path to your image repository and "Secret" with the created secret name. If your repository is public, you don't need to create a secret and just edit the "ContainerImagesPath" parameter with the path of your repository. For more information on how to create a secret you can check the following link: https://kubernetes.io/docs/concepts/configuration/secret/.

6   Troubleshooting

This section discusses the possible problems a user can face while setting up the testbed and how to overcome these problems.

6.1   K3s installation challenges

If your K3s services do not start properly after installation, remove them completely with all their associated packages using the following commands and try the installation steps again. For k3s server:

```
systemctl stop k3s.service
/usr/local/bin/k3s-uninstall.sh
rm -rf /var/lib/rancher/
rm -rf /etc/rancher/
```

For k3s agent:

```
/usr/local/bin/k3s-agent-uninstall.sh
rm -rf /var/lib/rancher/
rm -rf /etc/rancher/
```

When you set up the K3s in a non virtualized environment the following simple commands mentioned in Kubernetes website can be used for the installation on master and worker nodes, respectively:

```
curl -sfL https://get.k3s.io | sh -

curl -sfL https://get.k3s.io | K3S_URL=https://orchestrator_ip:6443
K3S_TOKEN="TOKEN" sh -
```

Please note that when you install K3s on a virtual machine and use the installation commands written above, often the K3s connection settings between Master and Workers are set to NAT interfaces. This creates a communication problem in virtualized environments, as the same IP address is usually assigned to those interfaces in different virtual machines. In this case, after running containers that are located on different nodes and need to communicate with each other, a communication problem arises. On the other hand, K3s will be installed with its default network manager when installed using the commands listed above. For a virtual environment, we recommend disabling the default and using Calico Network Manager for better communication between virtual machines. To solve the mentioned problems, we recommend using the commands provided in the sections 4.1 and 4.2 to install Kubernetes in all environments where you work.

6.2   Debug running pods

To make sure that your experiment runs properly based on your configuration use the following methods. SMOTEC uses its own created namespace named "smotec" to run its pods and services. When the Testbed module is running, user can see the specification of each pod using describe command as follows:

```
$kubectl describe <PodName> -n smotec
```

, where the PodName can be one of the pods listed as the output of following command:

```
kubectl get pods -n smotec
```

To see the terminal output of each pod execute logs command as follows:

```
$kubectl logs <PodName> -n smotec
```

The "logs" allow you to know the performance of your applications, whether they are failing or healthy, and are particularly useful for debugging purposes. To have a shell to a running pod in an interactive mode use the following command:

```
kubectl exec --stdin --tty <PodName> -- /bin/bash
```

The "exec" command enables you to get inside a running container by opening its shell. The shell provides a command-line interface for running commands and interacting with the container's environment, similar to running commands on your own computer's command line.