**Multi Modal Intelligent Traffic Signal System（MMITSS）**

**Field Deployment – Installation Manual**

Version 2.1

Jan. 2024

CART Lab, Purdue University

**Table of Contents**

1. Systems Requirements

2. Deployment – Docker Containers

3. Configuration

4. Setup and Monitoring

**Overview**

This documentation is created for the deployment for MMITSS in the city of Ann Arbor for the Smart Intersection Project funded by FHWA. This document includes the deployment guidance from pre-developed docker images to a docker container.

**Revision History**

|  |  |
| --- | --- |
| **Revision No. (Date)** | **Description** |
| 1.0  (05/20/2023) | 1. MMITSS Introduction with system structures. 2. Deployment steps, configuration examples. |
| 2.0  (02/01/2024) | 1. System validation strategy |
| 2.1  (10/01/2024) | Update new released docker image |

# System Requirements

Some required preparations have to be done before implementing MMITSS in the field.

1. Both MMITSS roadside (MRP) and vehicle-side (VSP) applications are run in an Ubuntu based operation system, the version is 18.04 and 20.04 LTS. The roadside unit (MRP) is running on an x86-based device while the vehicle-side unit (VSP) is running on an ARM-based device. (i.e., a laptop for MRP and a Raspberry Pi 4 for VSP respectively)
2. To deploy MMTISS applications in container, docker and supervisor are required to be installed first. (URL: <https://docs.docker.com/engine/install/ubuntu/>)
3. Docker images need to be extracted from DockerHub and loaded to the field device. For Ann Arbor testing, a pre-tested docker image has been uploaded publicly. The following shows the detailed sources for these two images.
   * For roadside applications: cartlabpurdue/mmitss-mrp-x86:AA1.0

<https://hub.docker.com/r/cartlabpurdue/mmitss-mrp-x86>

$ *docker pull cartlabpurdue/mmitss-mrp-x86:AA1.0*

* + For vehicle-side applications: cartlabpurdue/mmitss-mrp-x86:AA1.1

<https://hub.docker.com/r/cartlabpurdue/mmitss-vsp-x86>

$ *docker pull cartlabpurdue/mmitss-vsp-x86:AA1.1*

# Configurations

This section introduces the details of modifying configuration files for the deployment of MMTISS.

**File 1**: *mmitss-phase3-master-config. json*

*mmitss-phase3-master-config. json* is the major configuration file for MRP and VSP including IP addresses, UDP ports, NTCIP configurations, and intersection information.

1. For the MRP container, “*HostIp”,* “*SourceDsrcDeviceIp”, “IntersectionName”, “MapPayload”, “IntersectionID”*, *signal controller* “*IpAddress”, “NtcipPort”,* and “*NtcipBackupTime\_sec”, “Vendor”, “TimingPlanMib”*, “*IntersectionReferencePoint*” , “*Priority Parameter*” are required.
   * “*HostIP*”: ipv4 IP addresses for current edge device
   * “*SourceDsrcDeviceIP*”: ipv4 IP addresses for communication device, for MRP container, it should be RSU IP and for VSP, it should be OBU IP.
   * “*IntersectionName*”: name for the current intersection in MRP.
   * “*MapPayload*”: The map payload can be obtained by creating an intersection map using USDOT map tool (<https://webapp.connectedvcs.com/isd/>).
   * “*IntersectionID*”: must be consistent with the ID from map payload.
   * “*IpAddress”*: ipv4 IP address for current signal controller.
   * “*NtcipPort*”: NTCIP communications port, must be consistent with controller settings
   * “*NtcipBackupTime\_sec*”: must be consistent with controller settings.
   * “*Vendor*”: Vendor Name of controller, **for Ann Arbor deployment with SIEMENS controller, set it as “Standardmib”**
   * “*TimingPlanMib”*: path for additional NTCIP mib file. **for Ann Arbor deployment with SIEMENS controller, it is not used.**
   * *“ProrityParameter”*: set the weights of priority for different vehicle types and coordination for traffic operations.
2. For the VSP container, *“HostIp”,* “*SourceDsrcDeviceIp”,* “*VehicleType”* are required. The vehicle type has to be one of following strings:
   * “Transit”
   * “Truck”
   * “EmergencyVehicle”

**File 2**: *mmitss-coordination-plan.json*

The mmitss-coordination-plan.json file contains the coordination plan parameter and split data to run priority-based coordination within MMITSS. An example of mmitss-coordination-plan.json file as follows:

{

"IntersectionName": "xxx",

"CoordinationParameters": [

{

"CoordinationPlanName": "AM-Plan",

"CoordinationPatternNo": 1,

"SplitPatternNo": 1,

"CycleLength": 90,

"Offset": 0,

"CoordinationStartTime\_Hour": 6,

"CoordinationStartTime\_Minute": 30,

"CoordinationEndTime\_Hour": 9,

"CoordinationEndTime\_Minute": 30,

"CoordinationSplit": 20.0,

"CoordinatedPhase1": 2,

"CoordinatedPhase2": 6,

"SplitPatternData": {

"PhaseNumber": [

1,

2,

3,

4,

5,

6,

7,

8

],

"Split": [

15,

39,

12,

24,

15,

39,

16,

20

]

}

},

{

"CoordinationPlanName":"PM-Plan",

"CoordinationPatternNo": 2,

"SplitPatternNo": 2,

"CycleLength": 90,

"Offset": 10,

"CoordinationStartTime\_Hour": 15,

"CoordinationStartTime\_Minute": 30,

"CoordinationEndTime\_Hour": 19,

"CoordinationEndTime\_Minute": 0,

"CoordinationSplit": 20.0,

"CoordinatedPhase1": 2,

"CoordinatedPhase2": 6,

"SplitPatternData": {

"PhaseNumber": [

1,

2,

3,

4,

5,

6,

7,

8

],

"Split": [

19,

35,

12,

24,

19,

35,

16,

20

]

}

}

]

}

**File 3**: *mmitss-bus-stop-location.json*

The mmitss-bus-stop-location.json file is required only for the transit vehicles. Transit vehicles dwell in the bus stop for boarding and taking off passengers. MMITSS software component (priority-request-generator) is designed to send priority requests after passing the bus stop. The mmitss-bus-stop-location.json file contains the information of the bus stop location for each transit vehicle (depending on the travel route). An example of mmitss-bus-stop-location.json file is as follows:

{

"NoOfBusStop": 5,

"BusStopInformation": [

{

"IntersectionName": "xx-yy",

"IntersectionID": xy,

"TravelDirection": "EastBound",

"ApproachNo": 3,

"Latitude\_DecimalDegree": 33.142863,

"Longitude\_DecimalDegree": -110.134406,

"Elevation\_Meter": 739

},

{

"IntersectionName": "yy-xx",

"IntersectionID": yx,

"TravelDirection": "EastBound",

"ApproachNo": 5,

"Latitude\_DecimalDegree": 32.250825,

"Longitude\_DecimalDegree": -112.416047,

"Elevation\_Meter": 960

},

{

"IntersectionName": "yy-zz",

"IntersectionID": yz,

"TravelDirection": "WestBound",

"ApproachNo": 1,

"Latitude\_DecimalDegree": 30.289396,

"Longitude\_DecimalDegree": -112.112475,

"Elevation\_Meter": 687

},

{

"IntersectionName": "zz-yy",

"IntersectionID": zy,

"TravelDirection": "WestBound",

"ApproachNo": 1,

"Latitude\_DecimalDegree": 34.346593,

"Longitude\_DecimalDegree": -108.183794,

"Elevation\_Meter": 546

},

{

"IntersectionName": "xx-zz",

"IntersectionID": xz,

"TravelDirection": "WestBound",

"ApproachNo": 6,

"Latitude\_DecimalDegree": 35.9643024,

"Longitude\_DecimalDegree": -114.934988,

"Elevation\_Meter": 839

}

]

}

**File 4**: *supervisord.conf*

This file is used for managing the applications in MMITSS container. A template file is attached in Github repo in the directory: *AA\_Deploy\_2023/MRP\_config/nojournal/bin* ([*https://github.com/Dylan-Wyl10/SIP-MMITSS.gi*](https://github.com/Dylan-Wyl10/SIP-MMITSS.gi)). For the deployment purposes, there is no need to modify this file.

# Field Deployment

Required files for field deployment include configuration files for MRP and VSP, as well as the launch scripts. Purdue Team has published these required files in Git-Hub, and they could be pulled from the command below. For the Ann Arbor deployment, please access the directory: *$HOME/AA\_Deploy\_2023*:

$ *git clone https://github.com/Dylan-Wyl10/SIP-MMITSS.git*

After finishing preparing all configuration files, follow the following steps:

**Step1**: Create configuration files/folder.

It is required to create a configuration folder for MMITSS to access configuration files and save the logs. The configuration folder needs to follow a given structure in order to successfully run the applications. Both MRP and VSP follow the same structure but including different files. Create folder named “log”, which must be placed in the same directory as in the *mmitss-phase3-master-config.json* configuration file. To log the data, specify “Logging”: “True” in the mmitss-phase3-master-config.json file otherwise MMITSS will specify it as “False”. A detailed folder structure is listed below:

-**ROOT**

-**nojournal**

**-bin**

**-log**

**-** mmitss-phase3-master-config.json (MRP,VSP)

**-** mmitss-bus-stop-location.json (VSP)

**-** mmitss-data-external-clients.json (MRP)

- mmitss-coordination-plan.json (MRP)

- supervisord.conf (MRP, VSP)

Noted that the bold text means the folder directory. The *mmitss-phase3-master-config.json* configuration file is the main configuration file. The other json files mainly contain the required inputs for specific MMITSS applications. The supervisord.conf is used for managing the applications running in docker. For Ann Arbor deployment, a configuration template will be provided by Purdue Team in GitHub Repo (<https://github.com/Dylan-Wyl10/SIP-MMITSS/tree/main/AA_Deploy_2023> )

**Step2**: Launch MMITSS container

After copying the configuration files properly in the edge device, execute the launch scripts to run MMITSS applications. The scripts are in path: */AA\_Deploy\_2023/launch\_scripts.* For a new device that firstly deploys MMITSS applications, it is required to firstly execute *setup-deployment-environment.sh* to set up the environment variables. Then execute *launch-container.sh* to build MMITSS containers.

A detailed example for running *launch-container.sh* script is shown below:

1. Run the launch-container.sh script to run the docker container. The VSP container can be start by executing the following command:

$ *launch-container.sh*

A user interface will appear, and the following information have to be provided. The bold text needs to be figured out based on deployment details.

Full absolute path of MMITSS configuration directory: **<path-to-mmitss>/mrp\_config**

Name of container image on the Dockerhub: **mmitssuarizona/mmitss-vsp-arm:1.1**

Name of container: **vsp\_container**

Specify timezone string: **America/New\_York**

1. To monitor the containers, execute the following command:

$ *docker container exec –it <container name> /bin/bash*

1. To stop all the containers, execute the following command:

$ *docker container stop <container name>*

1. To start container the execute the following command:

$ *docker container start <container name>*

# Settings for other devices

* + - 1. **Communication devices**

Communication devices, OBU and RSU, are needed in the field applications for MMITSS to exchange information between VSP and MRP. An IMF script needs to be executed for the deployment. Noted that MMITSS has defined a port to receiving the forwarded messages from communication devices (*MessageReceiver: 10002*). So, the real-time forwarding message needs to configure its forward address as: <*MRP\_IP>: 10002.* On the other hand, MMITSS applications utilize the Immediate Forward (IMF) function to send customized V2X messages, the communication devices are supposed to listen from MMITSS through port 1516.

* + - 1. **Signal controller**

MMITSS receives the SPaT mib from the signal controller. The receiving port is predefined as 6053. The controller should be configured to send SPaT mib to: <MRP IP>: 6053

# System Validation

When the MMITSS applications are running, the system status can be monitored through the following methods.

1. **Infrastructure side (MRP)**
   1. **System Check**

When the container is up, use the following script to check if all the processes are running.

$ *sudo lsof -i -n -o +c0*

A screen shot is shown below that indicates the required applications running on MRP. Noted that there are in total of 10 processes (except for supervisord) with occupied port numbers. The ten processes are:

|  |  |
| --- | --- |
| **Process Name** | **Ports** |
| M\_wirelessMsgDecoder | 10002 |
| M\_MsgEncoder | 10003 |
| M\_ProrityRequestSolver | 20003 |
| M\_ProrityRequestSolverToTCIInterface | 20010 |
| M\_ProrityRequestServer | 20002 |
| M\_TrajectoryAvare | 20001 |
| M\_V2XDataCollector | <mrp\_ip>: 30001 |
| M\_MapSpateBroadcaster | <mrp\_ip>: 6053 |
| M\_TrafficControllerInterface | <mrp\_ip>: 20005 |
| M\_TrafficControllerInterfaceToSNMP | <mrp\_ip>: 20021 |

电脑萤幕画面

描述已自动生成

* 1. **Communication Port Check**

The MRP is designed to receive the SPaT mib message from the controller from port 6053 and broadcast SPaT and MAP messages (in J2735 format) to the RSU. To check if the workflow is functional, users could firstly check these two ports by script $ *tcpdump -i <net-adapter-id> port <port-number>*. The SPaT mib is a 245-byte-length message while the J2735 format messages are encoded with a message head (0x13 for SPaT and 0x12 for MAP), as shown below.

电脑屏幕截图

描述已自动生成

SPaT head = 0013

Map head = 0012

J2735 message type, head, etc.

Command

Length of spat mib message.

* 1. **Check the process log**

Within the MMITSS configuration directory, three log files are maintained: s*nmpEngineLog, prsolverLog*, and *prsLog*. The *snmpEngineLog* file is instrumental in verifying NTCIP-based communication between MRP applications and signal controllers. Successful NTCIP connections are confirmed by the presence of "Success" entries within this log, as illustrated in the following screenshot. The OIDs in snmp log indicates different parameters from the controllers. The *prsolverLog* and *prsLog* files, on the other hand, document the status of priority request processing. These logs are automatically generated at regular intervals once the system is operational.

图形用户界面, 文本, 应用程序

描述已自动生成

The target device IP should be the same as the controller NTCIP settings



Check if ping test are successful



Check if the mib components are successfully connected with valid value

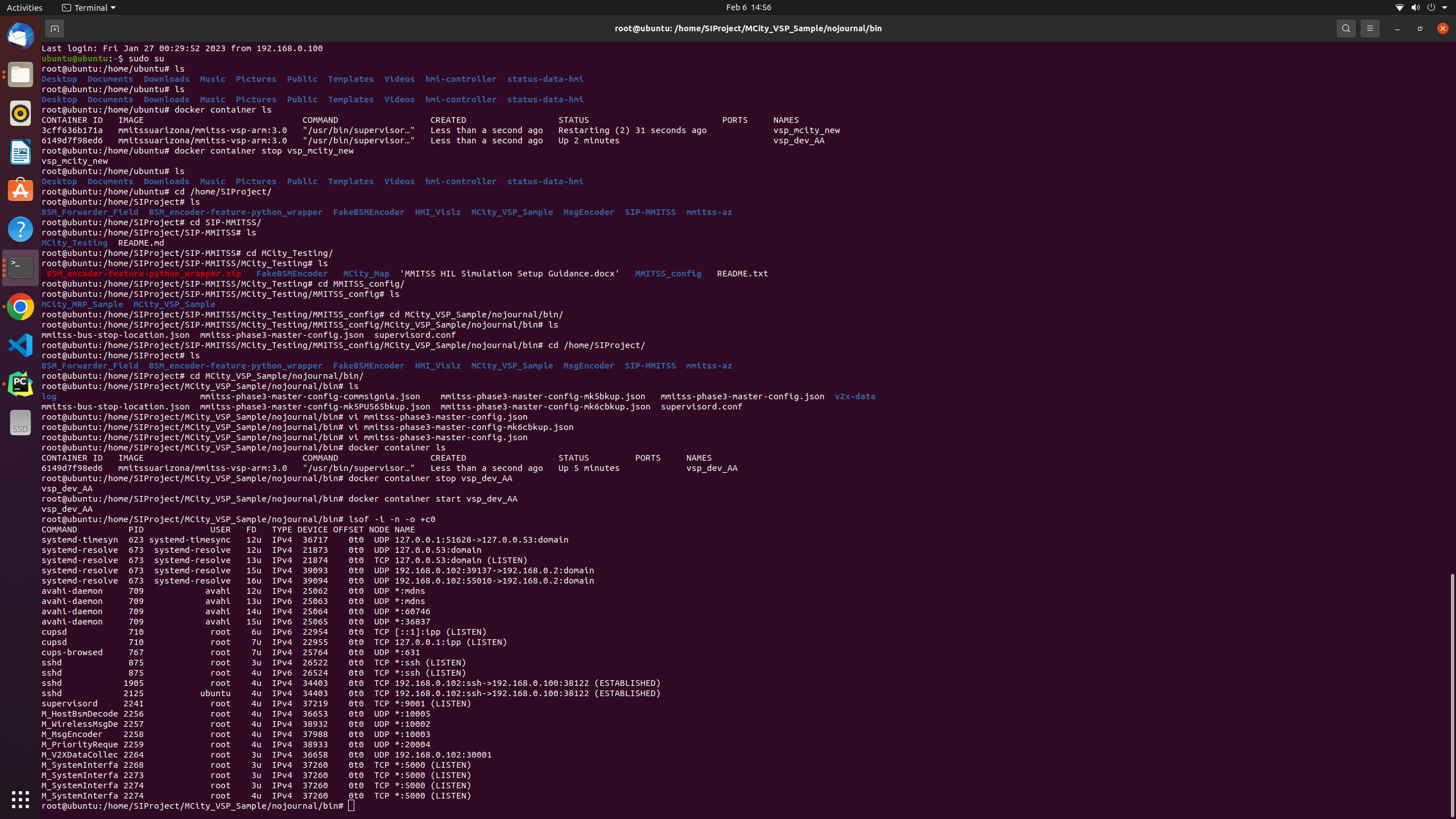


1. **Vehicle side (VSP)**

**Process Check**

The same process check strategy can be used to check if vehicle side applications are brought up. As the same command $ *sudo lsof -i -n -o +c0* is executed, the screen shot is shown below indicates the processes about vehicle side applications. A table shown below contains six applications that need to be checked.

|  |  |
| --- | --- |
| **Process Name** | **Ports** |
| M\_HostBsmDecoder | 10005 |
| M\_WirelessMsgDecoder | 10002 |
| M\_MsgEncoder | 10003 |
| M\_ProrityRequestGenerator | 20004 |
| M\_V2XDataCollector | <vsp\_ip>: 30001 |
| M\_SystemInterface | TCP:5000 |



**(b) Communication Port Check**

Given the infrastructure side applications are raised, the VSP container is expected to receive the MAP and SPaT messages from port 10002, which are forwarded from the OBU to validate the V2X radio communications. If the vehicle approaches to the intersection, the VSP generates SRMs and send them to port <OBU\_id>:1516 for broadcasting. Once the SRM is generated and sent successfully, it proves the MMITSS vehicle side applications are running correctly. The RSU should also receive the SRM at the same time and forward it to <MRP\_ip:10002>. By listening the port <OBU\_ip:1516> and <MRP\_ip:10002>, users are able to validate the communication and evaluate the end-to-end latency as the screen shot below.

VSP-OBU side

电脑屏幕截图

描述已自动生成电脑屏幕截图

描述已自动生成

Check Payload, message type,

Real-time forward from RSU, Check message (same with OBU side), forwarding port (<mrp\_ip>:10002)

RSU-MRP side

**(c) HMI Interface**

For vehicle side, there is a third-party tool that could visualize the data flow through an user interface. Users can run this interface on another laptop that is connected within the same subnet. The source code is provided in the repo (<https://github.com/Dylan-Wyl10/SIP-MMITSS/tree/main/HMI_interface> ). The folder includes two components: *hmi-controller*, *status-data-hmi* and both of them need to be brought up simultaneously.

A configuration file (*hmi-controller/mmitss-phase3-hmi-config.json*) contains the parameter settings of the HMI as shown below:

{

"HostIp": "192.168.0.100",

"PortNumber": {

"HMIController": 20109

}

}

The “*HostIP*” should be the ip address for the third-party laptop that is used for this interface, the “*PortNumber-HMIController*” are supposed to be consistent with the value “*HMIController*” in VSP configuration (in the mmitss-phase3-master-config.json configuration file).

The figure below illustrates the HMI (Human-Machine Interface) dashboard, which updates in real-time with the vehicle's movements and priority status. In the upper left quadrant of the HMI, SPaT is displayed, including the current signal status, timing parameters, and elapsed time. The lower left quadrant presents the Basic Safety Message (BSM) data received from the vehicle. On the upper right, the interface indicates the the priority request status of the vehicle and other nearby priority requests and their status (through the Signal Status Messages (SSM)). The bottom right interface shows the received and decoded maps. If the vehicle is within the range of multiple intersections, then multiple maps will be displayed. However, only one map should be “active”.

Priority request status

电脑萤幕的截图

描述已自动生成

Map message

BSM Message

SPaT Message

Priority Request table