

**Multi Modal Intelligent Traffic Signal System**

**Simulation Deployment – User Manual**

Revision 0.0

(Initial Release)

July 10, 2020

**Table of Contents**

1. Purpose of Document 3

2. Requirements: 3

3. Simulation Model 3

3.1. Configuring Connected Vehicle 3

3.2. Signal Controller 4

3.2.1. Configuring Econolite ASC/3 Signal Controller 4

3.2.2. Configuring MaxTime Signal Controller 6

4. VISSIM Driver Model DLL 7

5. Simulation-Tools 8

5.1. Message-Distributor 9

5.2. Priority-Request-Generator-Server 10

6. Deployment – Docker Containers 13

**List of Figures:**

Figure 1: Vissim simulation model of multiple intersections 3

Figure 2:Defining Connected Vehicle Type 4

Figure 3: Prepare to setup the parameters 5

Figure 4: Define the parameters 5

Figure 5: Starting MaxTime signal Controller 6

Figure 6: Defining the IP address in MaxTime Signal Controller 6

Figure 7: Flow diagram of Priority-Request-Generator-Server 12

Figure 8: Snapshot of supervisord.conf files for (a) intersection container

(b) simulation-tools container 17

# Purpose of Document

This document is an instruction guide for deploying Multi-Modal Intelligent Traffic Signal System (MMITSS) applications in the simulation platform. The document contains the detailed configuration and usage instructions for deploying the MMITSS software components in the docker container.

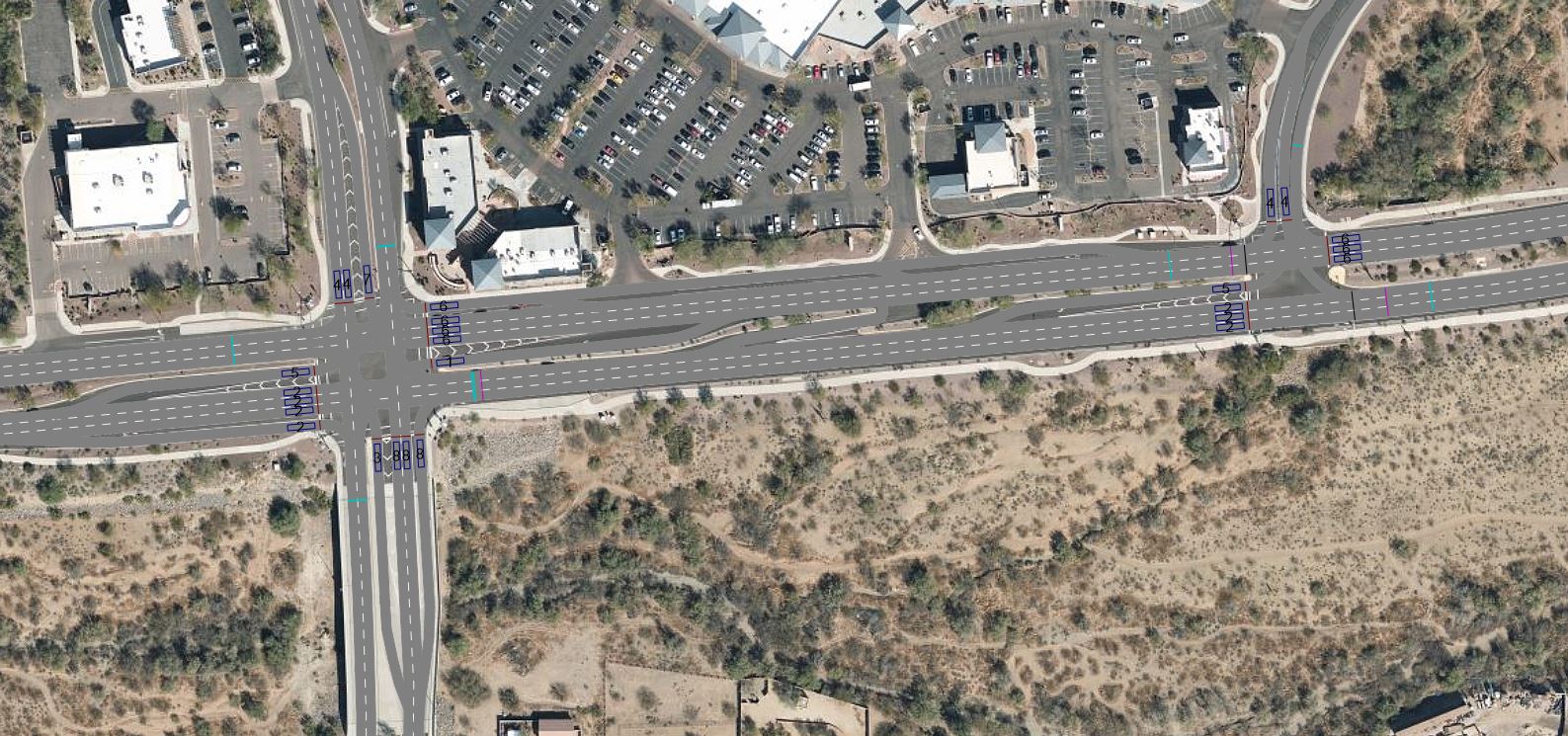
# Requirements:

Following devices, software and libraries are required to be installed to deploy MMITSS applications in the simulation environment.

1. VISSIM 10 or VISSIM 2020 installed in a windows computer
2. A Linux box (preferably Ubuntu 18.04) installed with docker, docker-compose, supervisor

# Simulation Model

PTV Vissim (a microscopic simulation software) is used to simulate all modes of traffic and analyses the performance of MMITSS software components. Vissim can be utilized to create realistic and accurate in every detail to test different traffic scenarios. A sample Vissim model of DaisyMountain-GavilanPeak and DaisyMountain-DedicationTrail intersections at Anthem, Arizona is shown in figure1.

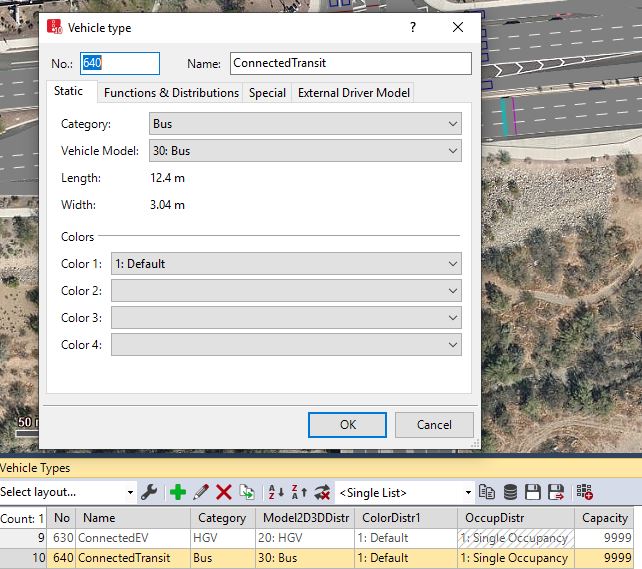
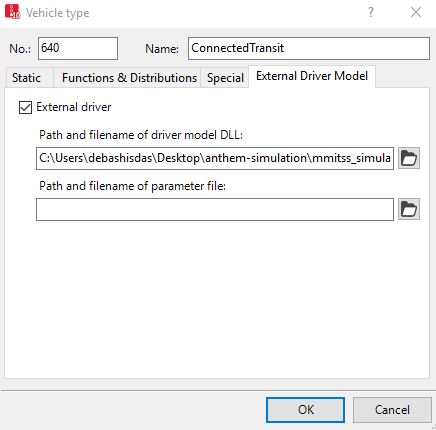


**Figure 1:** Vissim simulation model of multiple intersections

# 3.1. Configuring Connected Vehicle:

A new vehicle type (Connected vehicle) can be created using external driver model dll. In section 4, the method of setting up driver model dll is discussed.

1. To create a connected vehicle say Transit, a new vehicle type required to be defined (figure2).
2. After that, the directory of the diver model dll file has to define under the External Driver Model as shown in the figure 2.
3. Finally this new vehicle type is required to assign under Base Data->Vehicle Classes.

**Figure 2:** Defining Connected Vehicle Type

**3.2. Signal Controller:**

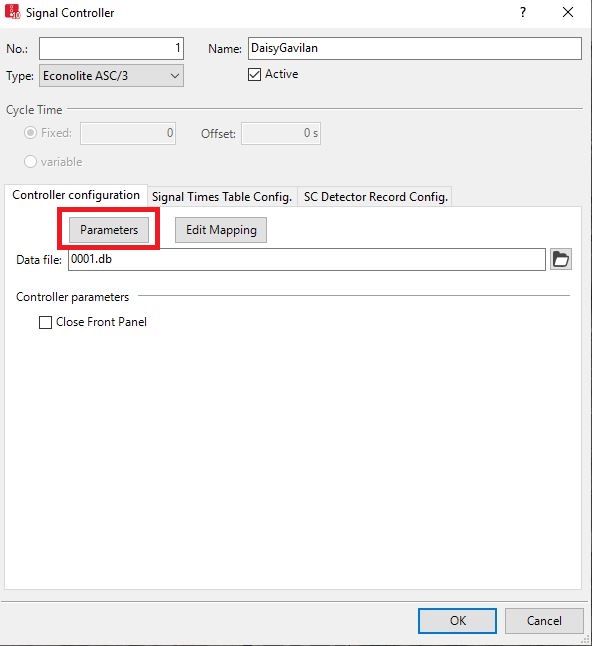
To establish communication between the MMITSS software components signal controller, it is required to configure the signal controller. Different agencies use different type of signal controllers. In Tucson and Anthem Econolite ASC/3 signal controller is used in the field, whereas in Portland MaxTime signal controller is used to control the traffic flow at each intersection.

**3.2.1. Configuring Econolite ASC/3 signal controller:**

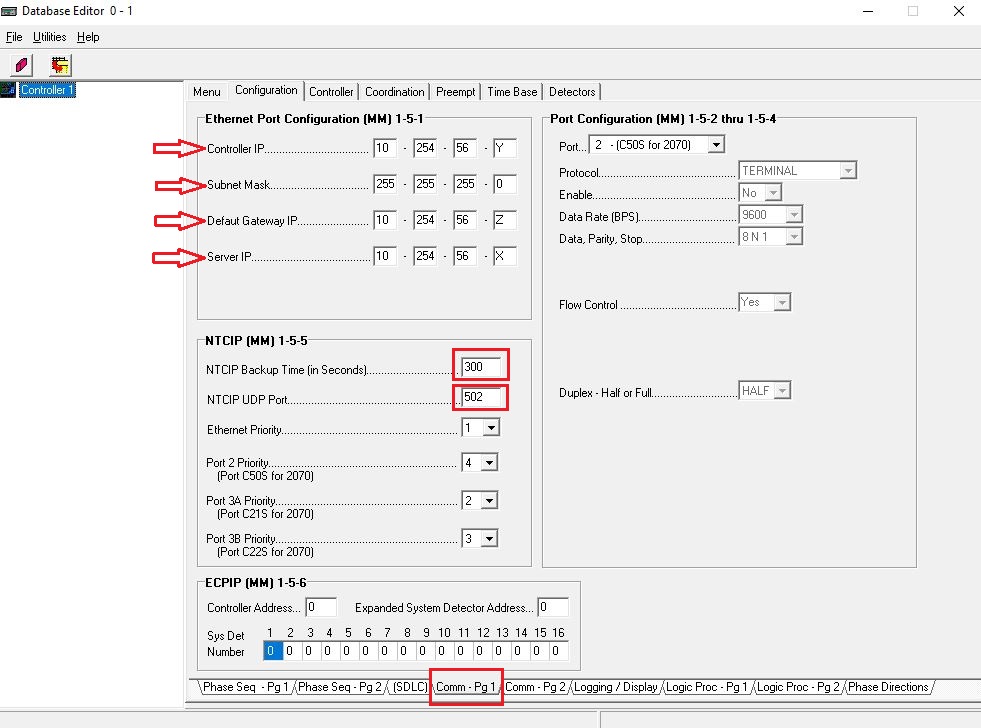
Following steps are required to configure Econolite ASC/3 signal controller:

(a) At first define the IP address of the computer on which Vissim software is installed. Mrp container and Vissim computer must be in the same subnet. For example- if the intersection container host IP address is: 10.254.56.xx then Vissim computer IP address will be 10.254.56.yy.

(b) Next step is to define the controller IP, subnet mask, default gateway IP, server IP, NTCIP backup time and NTCIP UDP Port in the Vissim ASC/3 contoller. To do so, from the Vissim go to each signal controller and click “Parameters” option (figure3). Then go to the Comm-Pg1 option (MM 1-4) under configuration and define the parameters (figure4).



**Figure 3:** Prepare to setup the parameters

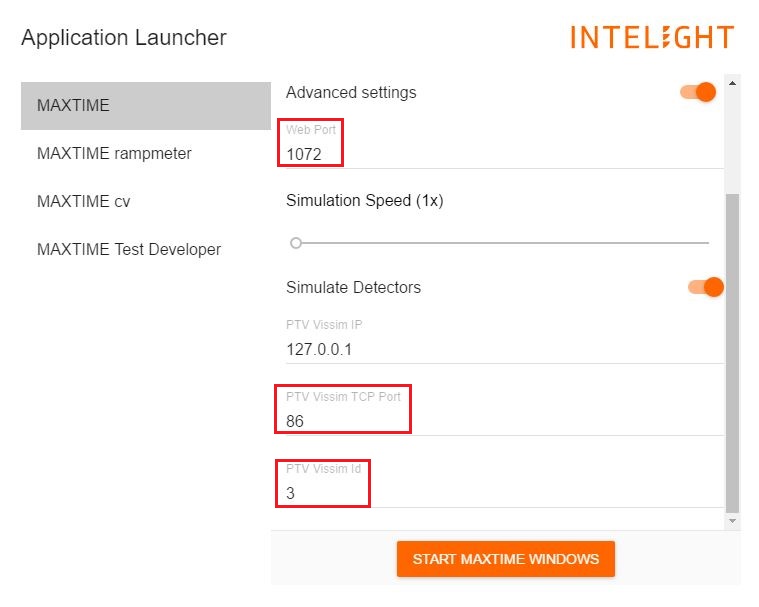


**Figure 4:** Define the parameters

**3.2.2. Configuring MaxTime signal controller:**

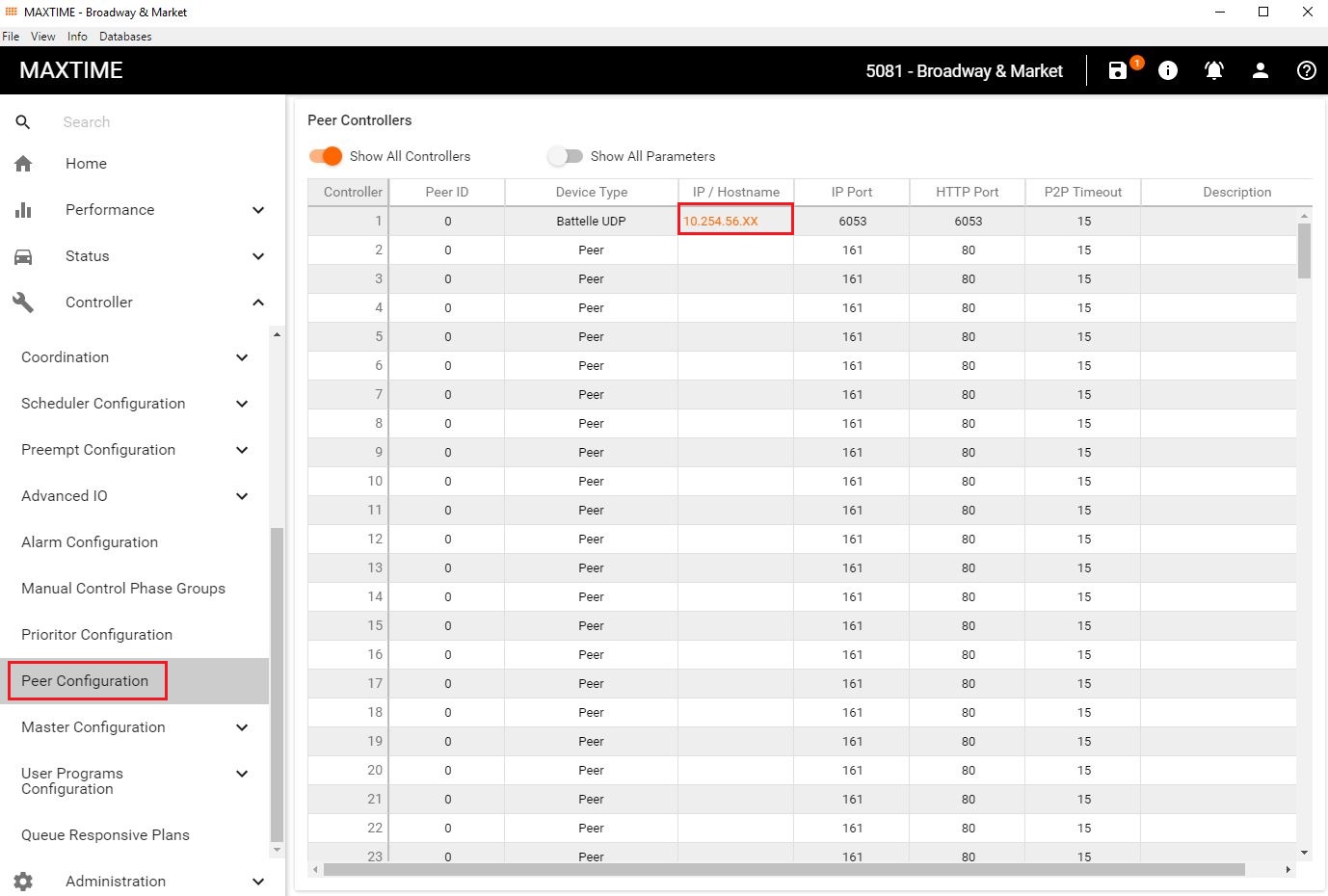
Following steps are required to configure MaxTime Signal controller:

(a) Start MaxTime signal controller by defining unique Web Port, PTV Vissim TCP port, PTV Vissim id for each intersection (figure 5).



**Figure 5:** Starting MaxTime signal Controller

(b) Then controller IP address has to define from peer configuration (figure 6).



**Figure 6:** Defining the IP address in MaxTime Signal Controller

# VISSIM Driver Model DLL

Built upon the original sample driver-model provided with VISSIM distribution, a custom driver-model has been developed to simulate connected vehicles that interact with MMITSS applications. Four variants of this driver-model are built which are the following:

1. M\_DriverModelPassenger.dll (type = “passenger”)
2. M\_DriverModelEmergency.dll (type = “emergency”)
3. M\_DriverModelTransit.dll (type = “transit”)
4. M\_DriverModelTruck.dll (type = “truck”)

At every time instant in the VISSIM simulation, vehicles using any of above driver-models send a UDP packet containing a Basic Safety Message (BSM) like JSON formatted information related to the vehicle’s identification, type, size, current location and motion to a network node defined in the configuration file. An example of JSON string developed by the driver model is as follows:

{

"MsgType": BSM,

"BasicVehicle": {

"heading\_Degree": 359.1,

"position": {

"elevation\_Meter": 587,

"latitude\_DecimalDegree": 33.843455,

"longitude\_DecimalDegree": -112.135159

},

"secMark\_Second": 2600,

"size": {

"length\_cm": 400,

"width\_cm": 300

},

"speed\_MeterPerSecond": 12.5,

"temporaryID": 5822735,

"type": "transit"

},

}

Functionally all four driver models are identical except that the value in the *type* field in the JSON string is different for each driver-model. These driver-models are tested in VISSIM 10 and VISSIM 2020.

To use any of these driver-models, a directory named “mmitss\_simulation” needs to be placed in the same directory where the “\*.inpx” file of the simulation exists. This directory is provided in the MMITSS simulation distribution package. The “mmitss\_simulation” directory contains a configuration file named “mmitss\_driver\_model\_config.json” and a subdirectory named “driver\_models” where all four prebuilt driver-models are stored. All four driver-models use the same configuration file and it must be made sure that the relative location of the configuration file from the “\*.inpx” file is not altered. The configuration file allows to configure the network node where the UDP packets of JSON strings need to be sent. In addition, it is required to input the GPS coordinates of the origin (0,0) of the simulation model in the configuration file. This information is required to correctly transform the vehicle’s current location from the VISSIM’s local coordinate system to the GPS coordinate system. An example of the content of the “mmitss\_driver\_model\_config.json” is as follows:

{

"msg\_distributor\_ip": "127.0.0.1",

"msg\_distributor\_port": 5000,

"vissim\_origin\_position": {

"elevation\_Meter": 540.11,

"latitude" : {

"Degree" : 33.0,

"Minute" : 50.0,

"Second" : 35.4

},

"longitude" : {

"Degree" : -112.0,

"Minute" : -8.0,

"Second" : -6.1

}

}

}

The IP address and UDP port of the receiver network node can be specified in the “msg-distributor-ip" and “msg-distributor-port" fields respectively. To interface with other MMITSS applications, this network-node must be the *message-distributor* component of MMITSS simulation-tools.

# Simulation-Tools

To interface simulation models with other MMITSS core components, two additional components are developed: *message-distributor* and *priority-request-generator-server*. The *message-distributor* component is responsible for receiving and distributing messages from VISSIM simulation model and other MMITSS components to the *priority-request-generator-server* component or to appropriate intersection containers, whereas the *priority-request-generator-server* component is responsible for formulating the Signal Request Messages (SRMs) based on the vehicle information received from the *message-distributor* component.

## 5.1. Message-Distributor

As the name suggests, the *message-distributor* component is responsible for receiving messages from VISSIM simulation model or other MMITSS components and distributing them to applicable configured clients. The *message-distributor* also allows to configure and simulate the wireless range (in Meter) of each intersection in the corridor along with a probability of dropping message packets. The clients (IP address and UDP port) for supported messages can be configured in the configuration file of the *message-distributor*. An example of the configuration file (mmitss-message-distributor-config.json) is as follows:

{

"package\_drop\_probability": 0,

"raw\_bsm\_logging": true,

"intersections": [

{

"name": "DaisyMountain\_GavilanPeak",

"ip\_address": "xxx.xxx.xxx.xxx",

"bsm\_client\_port": 5001,

"srm\_client\_port": 20002,

"dsrc\_range\_Meter": 500,

"position": {

"latitude\_DecimalDegree": 33.842932,

"longitude\_DecimalDegree": -112.135186,

"elevation\_Meter": 539

}

},

{

"name": "DaisyMountain\_DedicationTrail",

"ip\_address": "xxx.xxx.xxx.yyy",

"bsm\_client\_port": 5001,

"srm\_client\_port": 20002,

"dsrc\_range\_Meter": 500,

"position": {

"latitude\_DecimalDegree": 33.843239,

"longitude\_DecimalDegree": -112.131541,

"elevation\_Meter": 539

}

},

],

"bsm\_clients":

{

"transit":[

{

"ip\_address": "xxx.xxx.xxx.zzz",

"port":20022

}

],

"truck":[

{

"ip\_address": "xxx.xxx.xxx.zzz",

"port":20022

}

],

"emergency":[

{

"ip\_address": "xxx.xxx.xxx.zzz",

"port":20022

}

],

"passenger":[

]

},

"map\_clients":

[

{

"ip\_address": "xxx.xxx.xxx.zzz",

"port":20022

}

],

"ssm\_clients":

[

{

"ip\_address": "xxx.xxx.xxx.zzz",

"port":20022

}

]

}

The *message-distributor* receives and distributes following messages in the described way:

1. *BSM-like information from VISSIM simulation model:*

If the *vehicle type* in the received message is either “emergency”, “transit”, or “truck”, the message is forwarded to the priority-request-generator-server component (or more clients if required). In the configuration file above, the IP address xxx.xxx.xxx.zzz corresponds to the docker container (or physical machine) that hosts the *priority-request-generator-server.* In addition, regardless of the *vehicle type*, if the location described in the received message is within the wireless range (Meter) of any of the configured intersections, the message is distributed to that intersection’s BSM receiving client (defined by intersection\_ip and bsm\_client\_port).

1. *SRMs from priority-request-generator-server*

The *priority-request-generator-server* sends JSON strings of formulated SRMs to the *message-distributor.* If the location of vehicle (as per the received message) is within the wireless range of any (Meter) of any of configured intersections, the message is distributed to that intersection’s SRM receiving client (defined by intersection\_ip and srm\_client\_port). For MMITSS priority applications, the SRM receiving client is the *priority-request-server*.

1. *SSMs from priority-request-server of each intersection*

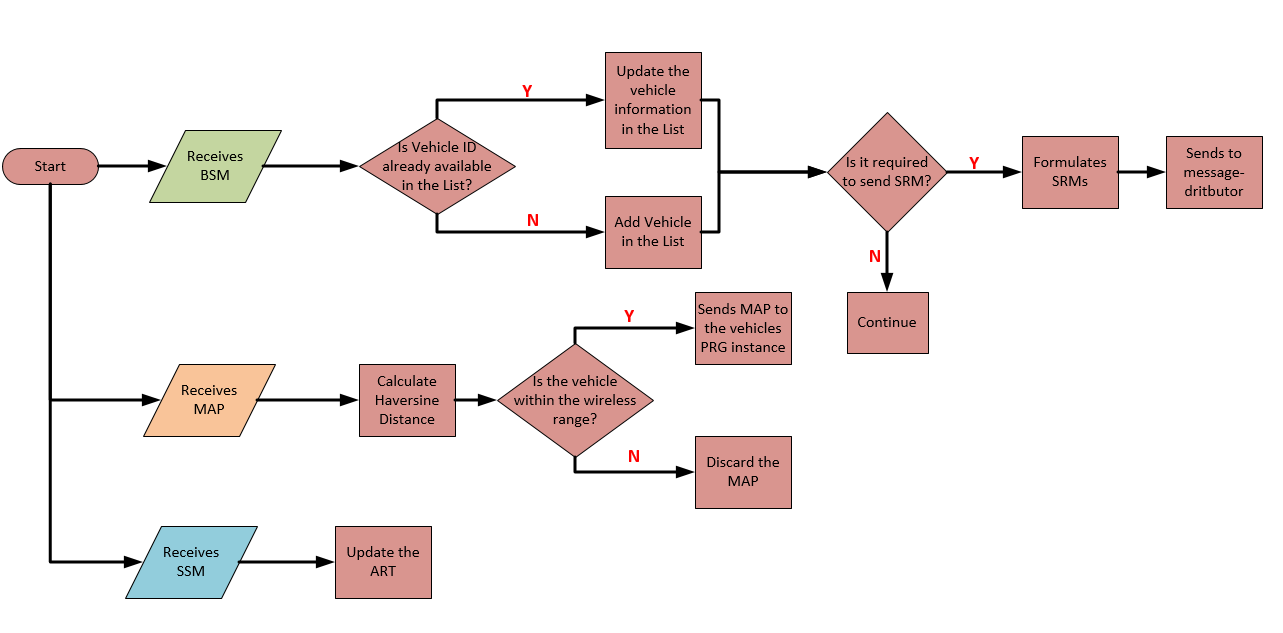
The *priority-request-server* component of each intersectionthat receives SRMs, generateand broadcast SSMs as an acknowledgement of the receipt of SRM if it is eligible for the priority service. The *priority-request-server* of each intersection also forwards such SSMs to the *message-distributor,* which then forwards these SSMs to configured SSM receiving clients. For MMITSS priority applications, one SSM receiving client is the *priority-request-generator-server* component.

1. *MAPs from map-spat-broadcasters of each intersection*

The *map-spat-broadcaster* component of each intersection broadcasts MAP message at a frequency of 1 Hz. It also forwards the MAP messages to the *message-distributor*, which then forwards these MAPs to configured MAP receiving clients. For MMITSS priority applications, one MAP receiving client is the *priority-request-generator-server* component.

**5.2. Priority-Request-Generator-Server**

*Priority-request-generator-server* is responsible managing list of priority eligible vehicles. It receives BSMs from the *message-distributor*. If the vehicle ID is not available in the list, it creates an instance of *priority-request-generator* for that vehicle ID. On the other hand, if the vehicle ID is already listed, it updates the vehicle information in the list. *Priority-request-generator-server* also receives MAP messages from the *message-distributor.* It calculates the haversine distance between the map reference point and all vehicles current gps points. If the distance is within wireless range, it forwards the MAP messages to the vehicle. Each instance of *priority-request-generator* can generate SRM JSON string, based on the MAP and GPS points. *The priority-request-generator-server* sends those SRMs JSON strings to the *message-distributor*. It can also receive SSMs JSON strings form the *message-distributor* and forwards it to the vehicles *priority-request-generator* instances to maintain the Active Request Table (ART).



**Figure 7:** Flow diagram of Priority-Request-Generator-Server

# Deployment – Docker Containers

To deploy MMITSS in the simulation environment, following steps can be followed:

**Step1:** Create configuration file

It is required to create mmitss-phase3-master-config.json, mmitss-data-external-clients.json, configuration files for each intersection and mmitss-phase3-master-config.json, mmitss-message-distributor-config.json configuration files for simulation tools (message-distributor, priority-request-generator-server). The configuration files contain the IP addresses, UDP ports etc. which are required to establish communication between the MMITSS software. Configuration files for each intersection need to be placed in the <intersectionName>/nojournal/bin directory and for simulation tools configuration files required to be placed in simulation-tools/nojournal/bin directory. An example of mmitss-phase3-master-config.json is follows:

{

"HostIp": "xxx.xxx.xxx.xxx",

"SourceDsrcDeviceIp": "xxx.xxx.xxx.yyy",

"IntersectionName": "xxx",

"MapPayload":001283fe38083020315abe2149d0eecf1800a0000271c4fcbd028280",

"IntersectionID" : XXXX,

"RegionalID" : 0,

"DataCollectorIP": "xxx.xxx.xxx.xyx",

"HMIControllerIP": "xxx.xxx.xxx.yxx",

"MessageDistributorIP": " xxx.xxx.xxx.zzz ",

"PriorityRequestGeneratorServerIP": "xxx.xxx.xxx.zzz",

"VehicleType" : 6,

"Logging" : "True",

"SRMTimedOutTime" : 10.0,

"PortNumber":

{

"MessageTransceiver":

{

"MessageSender": 10003,

"MessageReceiver": 10002,

"MessageEncoder": 10003,

"MessageDecoder": 10002

},

"MessageDistributor": 5000,

"RsmDecoder": 10006,

"OBUBSMReceiver": 10005,

"HostBsmDecoder": 10005,

"TrajectoryAware": 20001,

"PriorityRequestServer": 20002,

"PrioritySolver": 20003,

"PriorityRequestGenerator": 20004,

"TrafficControllerInterface": 20005,

"TrafficControllerCurrPhaseListener": 20006,

"TrafficControllerTimingPlanSender": 20007,

"PerformanceObserver": 20008,

"HMIController": 20009,

"PrioritySolverToTCIInterface": 20010,

"SignalCoordination": 20011,

"MapSPaTBroadcaster": 6053,

"DsrcImmediateForwarder": 1516,

"PriorityRequestServer\_SendSSM": 50003,

"DataCollector": 30006,

"SnmpEngine": 20020,

"SnmpEngineInterface": 20021,

"PriorityRequestGeneratorServer": 20022

},

"psid":

{

"map": "E0000017",

"spat": "8002",

"rsm": "8003",

"srm": "E0000019",

"ssm": "E0000020",

"bsm": "20"

},

"msgId":

{

"map": "0012",

"spat": "0013",

"rsm": "0021",

"srm\_lower": "001d",

"srm\_upper": "001D",

"ssm\_lower": "001e",

"ssm\_upper": "001E",

"bsm": "0014"

},

"SignalController":

{

"IpAddress": " xxx.xxx.xxx.yyy",

"NtcipPort": 502,

"TimingPlanUpdateInterval\_sec": 60,

"NtcipBackupTime\_sec": 300,

"Vendor": "Econolite",

"TimingPlanMib": "/nojournal/bin/EconoliteMib.py",

"InactiveVehPhases":[],

"InactivePedPhases":[],

"SplitPhases":

{

"1": 6,

"3": 8,

"5": 2,

"7": 4

},

"PermissiveEnabled":

{

"1": true,

"3": true,

"5": true,

"7": true

}

}

}

1. For each intersection, *“HostIp”, “IntersectionName”, “MapPayload”, “IntersectionID”*, *signal controller* *“IpAddress”, “NtcipPort”,* and “*NtcipBackupTime\_sec”, “Vendor”, “TimingPlanMib”* are required to be specified in the mmitss-phase3-master-config.json file. The map payload can be obtained by creating an intersection map using USDOT map tool (<https://webapp.connectedvcs.com/isd/>).
2. For simulation tools, *“MessageDistributorIP”*, *“PriorityRequestGeneratorServerIP”* are required to be specified in the mmitss-phase3-master-config.jsonfile.
3. Create a log folder in the nojournal/bin directory for each intersection and simulation tools. To log the data, specify *“Logging”: “True”* in the mmitss-phase3-master-config.json file.
4. The structure of mmitss-message-distributor-config.json is discussed in the section 5.1.

**Step2:** Define the docker files

It is required to specify two dockerfiles: 1. For MMITSS roadside software component- *“Dockerfiles”* and 2. For simulation tools- *“Dockerfile\_simulation-tools”*. The dockerfiles are required to be placed in the mmitss/bin directory.

An example of the *Dockerfile* is as follows:

#---------------------------------------------------------------------------#

# Dockerfile to build an x86 platform image for an intersection #

#---------------------------------------------------------------------------#

FROM mmitssuarizona/mmitss-base-x86:1.3

COPY TrafficControllerInterface/x86/M\_TrafficControllerInterface /mmitss

COPY SnmpEngine/x86/M\_SnmpEngine /mmitss

COPY PriorityRequestSolver/x86/M\_PriorityRequestSolver /mmitss

COPY PriorityRequestServer/x86/M\_PriorityRequestServer /mmitss

COPY MapSpatBroadcaster/x86/M\_MapSpatBroadcaster /mmitss

COPY supervisord.conf /mmitss

CMD ["/usr/bin/supervisord"]

An example of the *Dockerfile\_simulation-tools* is as follows:

#---------------------------------------------------------------------------#

# Dockerfile to build an x86 platform image for an intersection #

#---------------------------------------------------------------------------#

FROM mmitssuarizona/mmitss-base-x86:1.3

COPY PriorityRequestGeneratorServer/x86/M\_PriorityRequestGeneratorServer /mmitss

COPY MessageDistributor/x86/M\_MessageDistributor /mmitss

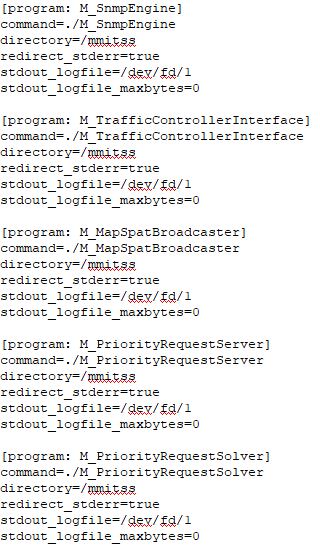
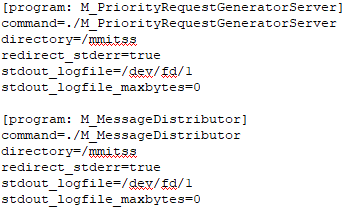
RUN mkdir -p /mmitss/map

COPY supervisord\_simulation-tools.conf /mmitss/supervisord.conf

CMD ["/usr/bin/supervisord"]

**Step 3:** Define supervisrod.conf and supervisor\_simulation-tool.conf file

To monitor the applications in the container, it is required to define two supervisord.conf files. For MMITSS roadside software components *supervisrod.conf* and for simulation *tools supervisrod-simulation-tools.conf* are required. They must be placed in the same level directory structure of Dockerfile (mmitss/bin).

(a) (b)

**Figure 8:** Snapshot of supervisord.conf files for (a) intersection container (b) simulation-tools container

**Step 4:** Define docker-compose.yml file

To build and run all the software components in the docker container, docker-compose.yml file is required to be specified. The yml file must be placed in the same level directory structure of Dockerfile (mmitss/bin).

1. For each *“intersection”, “container name”, “source”, and “ipv4\_address”* are required to be specified. Container name can be intersection name, source must be the directory of configuration files and ipv4 address has to match the Host IP address which is defined in the mmitss-phase3-master-config.json file for that intersection.
2. For simulations tools *“dockerfile”* name has to be defined along with the *“container name”, “source”, “ipv4\_address”*. Container name can be simulation-tools, source must be the directory of configuration files and ipv4 address has to match the *message-distributor* IP address which is defined in the mmitss-phase3-master-config.json file.
3. The ethernet interface of the computer and subnet has to be defined under the network. The name of the ethernet interface which will be used to communicate with the containers can be obtained by executing following command in a terminal on the computer:

ifconfig

An example of the docker-compose.yml file for two intersections and simulation tools is as follows:

version: "3.8"

services:

daisy-gavilan:

build:

context: ./

container\_name: daisy-gavilan

volumes:

- type: bind

source: $MMITSS\_ROOT/mmitss/bin/corridors/Anthem/Daisy-Gavilan/nojournal

target: /nojournal

networks:

mmitss\_vlan:

ipv4\_address: xxx.xxx.xxx.xxx

daisy-dedication:

build:

context: ./

container\_name: daisy-dedication

volumes:

- type: bind

source: $MMITSS\_ROOT/mmitss/bin/corridors/Anthem/Daisy-Dedication/nojournal

target: /nojournal

networks:

mmitss\_vlan:

ipv4\_address: xxx.xxx.xxx.yyy

simulation-tools:

build:

context: ./

dockerfile: Dockerfile\_simulation-tools

container\_name: simulation-tools

volumes:

- type: bind

source: $MMITSS\_ROOT/mmitss/bin/corridors/simulation-tools/nojournal

target: /nojournal

networks:

mmitss\_vlan:

ipv4\_address: xxx.xxx.xxx.zzz

networks:

mmitss\_vlan:

driver: macvlan

driver\_opts:

parent: eno2

ipam:

config:

- subnet: xxx.xxx.xxx.0/24

**Step 5:** Build and run docker container

1. Set the MMITSS path in the .bashrc file by executing the following command

Export /MMITSS\_ROOT=<mmitss directory>

For example if mmitss is cloned on home/user directory then the command will be:

Export /MMITSS\_ROOT=/home/user

1. Run the VISSIM simulation model
2. Open a terminal in Linux box and directed to mmitss/scripts directory
3. Run following script to build all the software components:

mmitss\_docker\_make\_all\_for\_x86.sh

1. To build and run the docker containers go to mmitss/bin directory and execute the following command:

docker-compose up --build

1. To monitor the containers execute the following command:

docker-compose exec <container\_name> /bin/bash

It will direct inside the container. Following command can be executed to monitor the applications using supervisor.

supervisordctl

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

docker-compose stop

To stop a specific container, execute the following command

docker-compose stop <container\_name>

1. To start container the execute the following command:

docker-compose start

To start a specific container, execute the following command

docker-compose start <container\_name>

1. To down the whole macvlan network of docker containers, execute the following command

docker-compose down

1. To up the whole macvlan network of docker containers, execute the following command

docker-compose up