NAP-Vanetza - Introduction and Containerized Usage Guide

Gitlab: https://code.nap.av.it.pt/mobility-networks/vanetza

Vanetza is an open-source C++ implementation of the ETSI C-ITS protocol suite designed to operate on ITS-G5 channels in a Vehicular Ad Hoc Network (VANET) using IEEE 802.11p (WAVE).

NAP-Vanetza extends the base Vanetza project to integrate MQTT and JSON capabilities, as well as additional types of ETSI C-ITS messages.

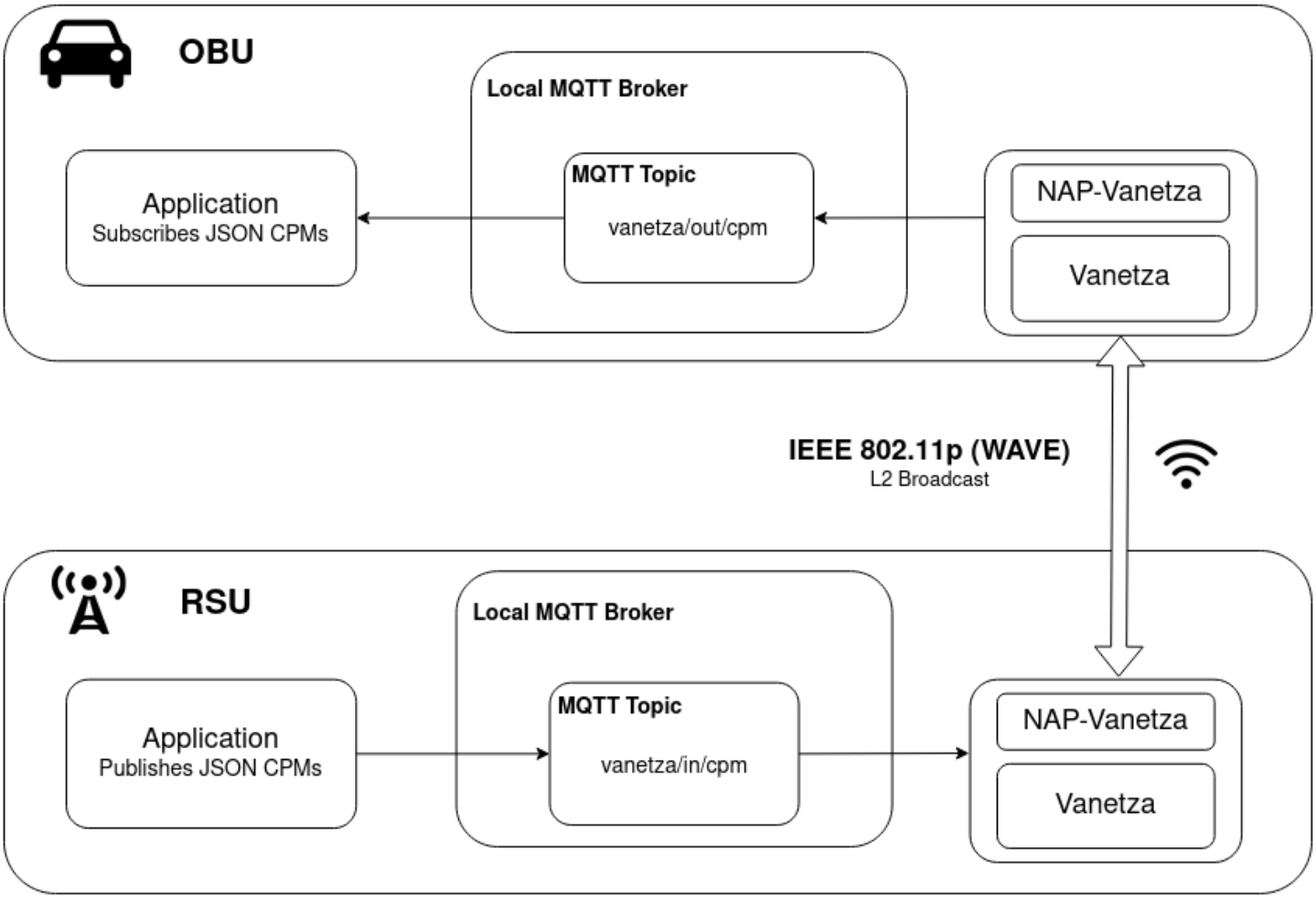
The following message types are supported:

- Cooperative Awareness Messages (CAM)
- Decentralized Environmental Notification Message (DENM)
- Collective Perception Message (CPM)
- Vulnerable Road User Awareness Message (VAM)
- Signal Phase And Timing Extended Message (SPATEM)
- MAP (topology) Extended Message (MAPEM)
- GeoNetworking Beacons

Put simply, NAP-Vanetza's purpose is to manage the encoding, decoding, sending, and receiving of ETSI C-ITS messages, thus abstracting those layers from VANET application developers.

Applications that need to send ETSI C-ITS messages interact with the service by building a JSON representation of the given message and publishing it in a specific MQTT topic, which Vanetza subscribes to. Likewise, applications that need to receive incoming messages do so by subscribing to the respective MQTT topics, that Vanetza publishes JSON to.

The following diagram examplifies a common usage pattern:



Note: In the case of CAMs, NAP-Vanetza also has a pre-defined "hard-coded" CAM message which is periodically sent at a configurable frequency and with updated GPS values, without the need for an external application to publish JSON CAMs (which is also allowed). This behaviour can be disabled.

Building (Docker)

1. Install docker and docker-compose

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
sudo bash -c 'echo "deb [arch=$(dpkg --print-architecture)] https://download.dock
sudo apt update
sudo apt install docker-ce docker-compose
```

2. Clone NAP-Vanetza's repository and navigate to the project's root directory

3. Build the docker image

```
docker build -t vanetza:test .
```

2. Create the Docker network that the Vanetza containers will use to exchange ETSI C-ITS messages

```
docker network create vanetzalan0 --subnet 192.168.98.0/24
```

Usage (Docker)

To start the Vanetza containers run:

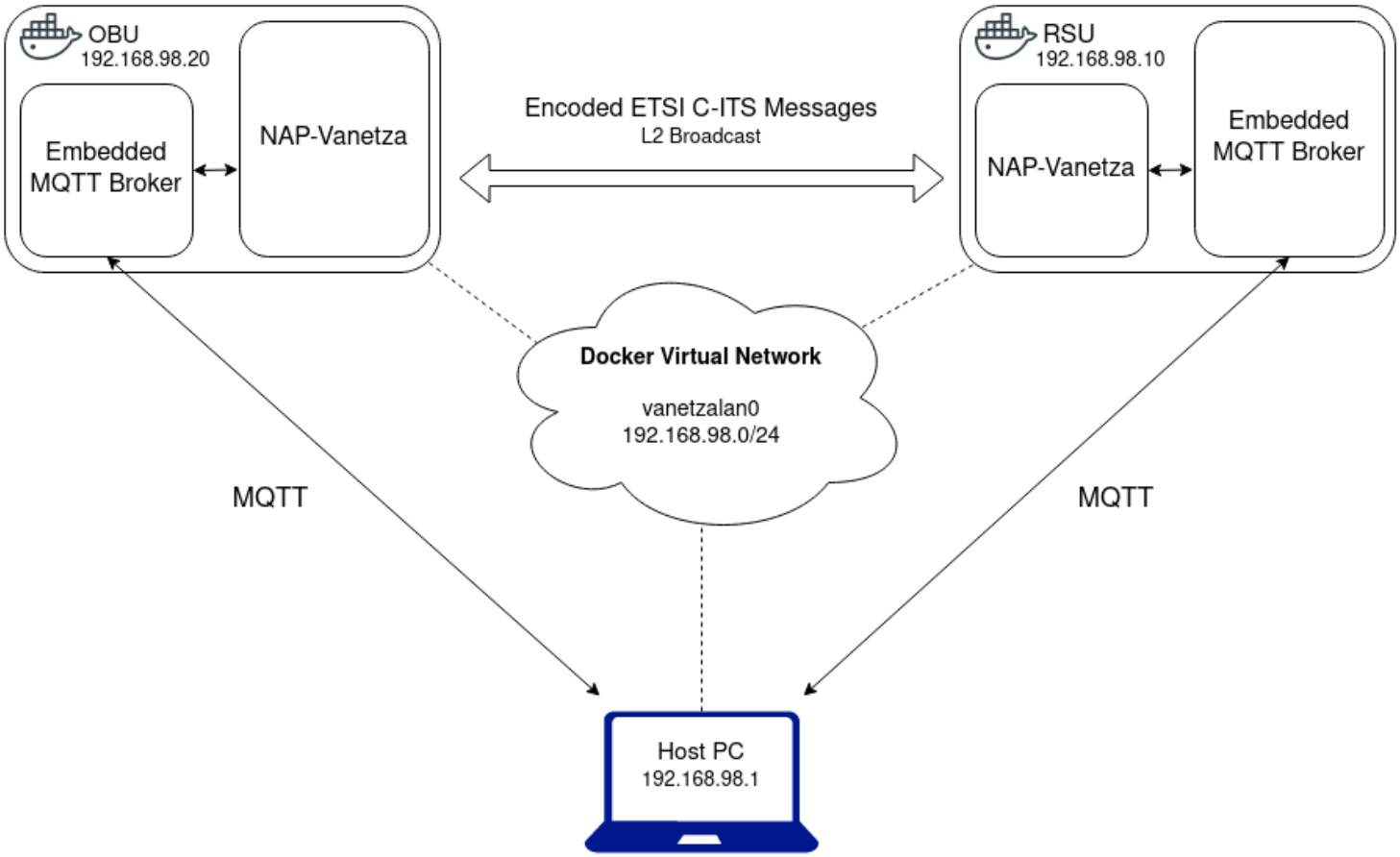
```
docker-compose up
```

To stop use CTRL-C.

By default, docker-compose creates two containers, emulating an RSU and OBU pair. This can, however, be freely extended just by duplicating the existing container specifications in **docker-compose.yml** (while making sure to set a unique IP address for each one).

Refer to the configuration section for more details on how to configure each container.

Each container includes an embeded MQTT broker in order to fully simulate the common environment described in the first section. This feature can be disabled if required.



Running in the background

To run the containers in the background:

```
docker-compose up -d
```

To stop:

```
docker-compose down
```

To consult the logs of a particular container:

```
docker-compose logs obu
```

Wireshark

Wireshark can be used to sniff and analyse the ETSI C-ITS messages exchanged between the different containers on the virtual network interface created by Docker

To find out the correct interface name run:

```
ifconfig
```

And find the interface whose IP address is **192.168.98.1**. Its name will usually start with 'br-'

To filter out miscellaneous packets and protocols and show only ETSI C-ITS messages, use the following filter:

```
eth.type == 0x8947
```

Note: CPMs are only supported/interpreted by Wireshark since version 3.6, which is not yet available to install on Linux OSs.

To circumvent this, compile Wireshark 3.7 from source by running:

```
wget -q -O - https://gist.githubusercontent.com/RodrigoRosmaninho/be85885b1cd415a
```

The script will print out further instructions on how to run the compiled binary.

MQTT

For quick tests on the command line, install the mosquitto-clients package

```
sudo apt install mosquitto-clients
```

To subscribe to MQTT topics and see every new message on the command line use:

```
# All topics
mosquitto_sub -h 192.168.98.10 -t "#" -v

# Subset of topics
mosquitto_sub -h 192.168.98.10 -t "vanetza/out/#" -v

# Specific topic
mosquitto_sub -h 192.168.98.10 -t "vanetza/out/cam" -v
```

To publish an MQTT message to a specific topic use:

```
mosquitto_pub -h 192.168.98.10 -t "vanetza/in/cam" -m "{\"accEngaged\":true,\"acc
```

MQTT can also be easily integrated into your application's code by using third-party libraries such as paho-mqtt, available for multiple programming languages.

Constructing the JSON messages according to ETSI specifications

In order to make the encoding and decoding process possible, the JSON messages received and sent by Vanetza through MQTT are required to follow the strict format specified in each message type's respective ETSI specification document.

You may consult those documents in the following links:

- Common to all messages - ETSI TS 102 894-2 V1.2.1 - Annex A & B
- CAM - ETSI EN 302 637-2 V1.4.1 - Annex A & B
- DENM - ETSI EN 302 637-3 V1.2.1 - Annex A & B
- CPM - ETSI TR 103 562 V2.1.1 - Annex A & B
- VAM - ETSI TS 103 300-3 V2.1.1 - Annex A & B
- SPATEM - ETSI TS 103 301 V1.1.1 - Annex A (and Common Data Types)
- MAPEM - ETSI TS 103 301 V1.1.1 - Annex B (and Common Data Types)

NAP-Vanetza includes some examples of valid JSON messages in the examples folder.

Note that the ITS PDU Header must not be included in messages published to Vanetza, as it's automatically filled at encode time. It is, however, present in messages published by Vanetza, so as to give developers access to the StationID field.

Simpler Formats

In the case of CAMs and VAMs, NAP maintains a seperate, simpler, format, which Vanetza also accepts (see examples folder).

As such, CAMs and VAMs use four MQTT topics each, ie:

- vanetza/in/cam - input CAMs using the simple format
- vanetza/in/cam_full - input CAMs using the full ETSI format
- vanetza/out/cam - output CAMs using the simple format

- vanetza/out/cam_full - output CAMs using the full ETSI format

The topic names are fully configurable.

Note: The simpler format for VAMs is not yet fully implemented. For the time being, both vam and vam_full topics behave as vam_full.

Error Messages

If your application publishes an invalid JSON ETSI C-ITS message, the following errors will appear in the respective Vanetza container's logs:

- JSON is malformed and can't be parsed. Verify that the message follows JSON's schema rules.

```
-- Vanetza JSON Decoding Error --
Check that the message format follows JSON spec
<Exception Info>
```

- JSON is valid but can't be parsed as the specified ETSI C-ITS message. Verify that all required fields are present and that the messages follow the respective ETSI format correctly.

```
-- Vanetza ETSI Decoding Error --
Check that the message format follows ETSI spec
<Exception Info>
```

- One or more values does not fit the type or constraints specified in ETSI documents listed above.

```
-- Vanetza UPER Encoding Error --
Check that the message format follows ETSI spec
<Exception Info>
```

- An unexpected error occurred. Please report it.

```
-- Unexpected Error --
Vanetza couldn't decode the JSON message. No other info available
```

```
or

-- Unexpected Error --
Vanetza couldn't send the requested message but did not throw a runtime error on
No other info available
```

Configuration

NAP-Vanetza has a set of configurable attributes with the goal of allowing for fine-tuning its operation.

These attributes are generally set in the **config.ini** file located in **tools/socktap/config.ini**

However, this becomes impractical for docker-compose deployments which include several Vanetza containers with heterogeneous configuration needs. These situations would require a separate config file for each container mounted as a volume.

To solve this, NAP-Vanetza also accepts configuration via environment variables that can be set in the environment section of **docker-compose.yml**. Any values set via environment variables have priority over the ones found in config.ini, thus making it possible to use config.ini for common configurations and environment variables for values that are unique to each container (i.e: Station ID, MAC Address, etc)

Note: The vanetza docker image must be rebuilt after every change to the config.ini file.

```
docker build -t vanetza:test .
```

Environment variable changes, however, only require a restart of the running containers.

```
docker-compose restart
```

The following table summarizes the available configuration options:

.ini file key	Environment var key	Description
general.interface	VANETZA_INTERFACE	Network interface where the ETSI

		messages are exchanged
general.mqtt_broker	VANETZA_MQTT_BROKER	MQTT Broker's IP address or DNS name
general.mqtt_port	VANETZA_MQTT_PORT	MQTT Broker's Port
general.prometheus_port	VANETZA_PROMETHEUS_PORT	Port on which Vanetza exposes metrics
general.rssi_port	VANETZA_RSSI_PORT	Port on which Vanetza communicates with the RSSI_Discovery service
general.ignore_own_messages	VANETZA_IGNORE_OWN_MESSAGES	Don't capture or decode messages originating from the station itself
general.ignore_rsu_messages	VANETZA_IGNORE_RSU_MESSAGES	Ignore messages from RSUs - Usually set on RSUs
general.to_dds_port	VANETZA_TO_DDS_PORT	Port on which Vanetza sends JSON to be published in DDS topics
		Port on which

general.from_dds_port	VANETZA_FROM_DDS_PORT	Vanetza receives JSON from DDS topics
station.id	VANETZA_STATION_ID	ETSI Station ID field
station.type	VANETZA_STATION_TYPE	ETSI Station Type field
station.mac_address	VANETZA_MAC_ADDRESS	Virtual Mac Address used as the source on L2 ethernet headers
station.beacons_enabled	VANETZA_BEACONS_ENABLED	Send GeoNetworking Beacons every 3 seconds
station.use_hardcoded_gps	VANETZA_USE_HARDCODED_GPS	Use hardcoded gps coordinates instead of information provided by a GPS module
station.latitude	VANETZA_LATITUDE	Hardcoded GPS latitude - Usually set on static RSUs
station.longitude	VANETZA_LONGITUDE	Hardcoded GPS longitude - Usually set on static RSUs
station.length	VANETZA_LENGTH	Vehicle lenght

		in meters
station.width	VANETZA_WIDTH	Vehicle width in meters
cam.full_topic_in	VANETZA_CAM_FULL_TOPIC_IN	MQTT/DDS topic from which Vanetza receives JSON CAMs in the full ETSI spec format
cam.full_topic_out	VANETZA_CAM_FULL_TOPIC_OUT	MQTT/DDS topic to which Vanetza sends JSON CAMs in the full ETSI spec format
vam.full_topic_in	VANETZA_VAM_FULL_TOPIC_IN	MQTT/DDS topic from which Vanetza receives JSON VAMs in the full ETSI spec format
vam.full_topic_out	VANETZA_VAM_FULL_TOPIC_OUT	MQTT/DDS topic to which Vanetza sends JSON VAMs in the full ETSI spec format
---	START_EMBEDDED_MOSQUITTO	Start an MQTT server inside the container to simulate a full OBU or RSU

		with a local MQTT broker

Each supported type of message (CAM, DENM, CPM, VAM, SPATEM, MAPEM) has its own set of configurations, which are specified in the following table (using CAMs as an example):

.ini file key	Environment var key	Description	Default
cam.enabled	VANETZA_CAM_ENABLED	Enable the CAM module	true
cam.mqtt_enabled	VANETZA_CAM_MQTT_ENABLED	Enable publishing and subscribing to MQTT topics	true
cam.dds_enabled	VANETZA_CAM_DDS_ENABLED	Enable publishing and subscribing to DDS topics	true
cam.periodicity	VANETZA_CAM_PERIODICITY	Periodicity with which to send the default CAM, in milliseconds	1000
cam.topic_in	VANETZA_CAM_TOPIC_IN	MQTT/DDS topic from which Vanetza receives JSON CAMs to encode and send	vanetza/in/cam
cam.topic_out	VANETZA_CAM_TOPIC_OUT	MQTT/DDS topic to which Vanetza sends JSON CAMs that were received and	vanetza/out/ca

		decoded	
cam.udp_out_addr	VANETZA_CAM_UDP_OUT_ADDR	Address of the UDP server to which Vanetza sends decoded JSON CAMs, in order to minimize communication latency - Used in NAP's Connection Manager v1	127.0.0.1
cam.udp_out_port	VANETZA_CAM_UDP_OUT_PORT	Port of the UDP server to which Vanetza sends decoded JSON CAMs, in order to minimize communication latency - Used in NAP's Connection Manager v1	5051

Project's State and Missing Fields

The NAP-Vanetza project is still under active development and is frequently updated to introduce new features and correct issues. Please report any problems you encounter.

The following field types are not yet supported by Vanetza. As such, they will be absent from JSON messages generated by Vanetza and ignored in messages received by Vanetza.

These fields are generally optional and relatively unimportant. They will be added later on a case-by-case basis, should they become necessary.

- PhoneNumber
- OpeningDaysHours
- MessageFrame
- DescriptiveName
- RegionalExtension
- Iso3833VehicleType
- REG-EXT-ID-AND-TYPE.&id
- REG-EXT-ID-AND-TYPE.&Type
- MESSAGE-ID-AND-TYPE.&id
- MESSAGE-ID-AND-TYPE.&Type
- PreemptPriorityList
- WMInumber
- VDS
- TemporaryID

Advanced Usage

DDS

In order to minimize communication latency between Vanetza and any critical producer/consumer applications, NAP-Vanetza also supports publishing and subscribing to OMG Data Distribution Service (DDS) topics. This is accomplished using an external Golang DDS module that is included in the Vanetza container.

To use it, simply activate the <message_type>.dds_enabled configuration flags. NAP-Vanetza will use the same topic names configured for MQTT. In fact, both technologies may be used simultaneously, if required.

The **tools/dds_service** folder includes simple python examples for producer and subscriber applications.

You may also use NAP's dds-mqtt-adapter project, which subscribes to every Vanetza DDS topic and relays any published messages to an MQTT broker. This may be useful for:

- Allowing developers easy access to the exchanged messages, for debugging or monitoring purposes

- Allowing less critical services to effectively subscribe to a DDS topic via MQTT, if NAP-Vanetza's MQTT functionality is disabled for performance reasons.

Prometheus Metrics

When running, Vanetza continuously computes a set of metrics regarding its current status, message statistics, and latency information. These are exposed using the Prometheus format, at the port specified in the configuration file.

```
observed_packets_count_total{direction="tx",message="spatem"}
observed_packets_count_total{direction="rx",message="spatem"}
observed_packets_count_total{direction="tx",message="vam"}
observed_packets_count_total{direction="rx",message="vam"}
observed_packets_count_total{direction="rx",message="mapem"}
observed_packets_count_total{direction="tx",message="cpm"}
observed_packets_count_total{direction="tx",message="mapem"}
observed_packets_count_total{direction="tx",message="denm"}
observed_packets_count_total{direction="rx",message="denm"}
observed_packets_count_total{direction="tx",message="cam"}
observed_packets_count_total{direction="rx",message="cpm"}
observed_packets_count_total{direction="rx",message="cam"}
# ---
observed_packets_latency_total{direction="tx",message="spatem"}
observed_packets_latency_total{direction="rx",message="spatem"}
observed_packets_latency_total{direction="tx",message="vam"}
observed_packets_latency_total{direction="rx",message="vam"}
observed_packets_latency_total{direction="rx",message="mapem"}
observed_packets_latency_total{direction="tx",message="cpm"}
observed_packets_latency_total{direction="tx",message="mapem"}
observed_packets_latency_total{direction="tx",message="denm"}
observed_packets_latency_total{direction="rx",message="denm"}
observed_packets_latency_total{direction="tx",message="cam"}
observed_packets_latency_total{direction="rx",message="cpm"}
observed_packets_latency_total{direction="rx",message="cam"}
```

These are easily extensible.

Help

Questions and Bug Reports: @rrosmaninho / r.rosmaninho@av.it.pt