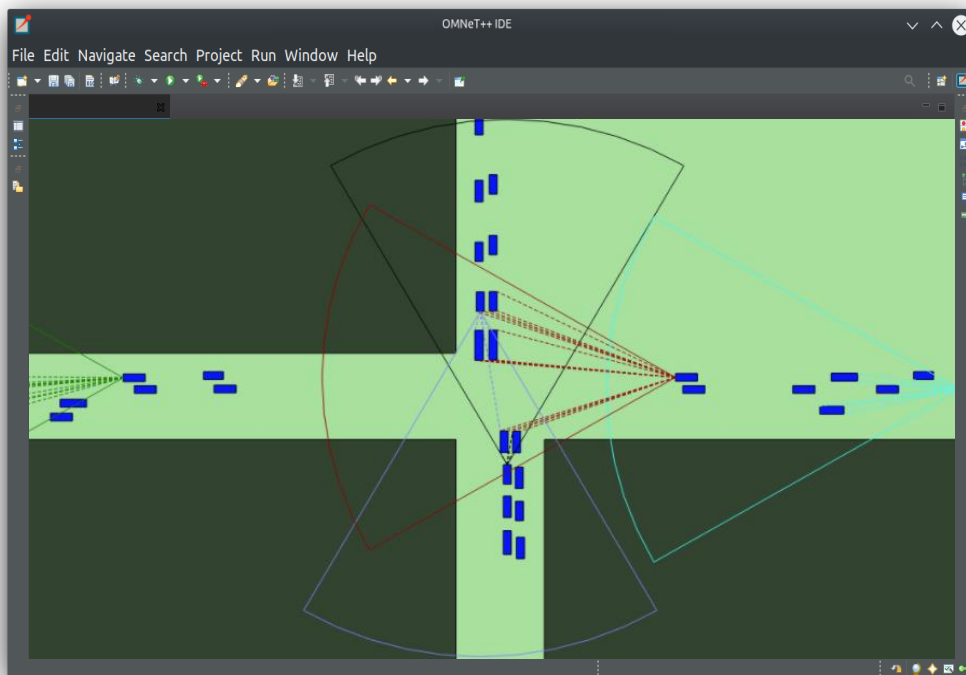


F2MD Technical Documentation

Maxime Georges – July 2021



This document aims to describe all the functionalities of the F2MD framework, and how they work. This document will first offer a brief overview of the code architecture and the features it offers, and then will study in details all its components (CAM and CPM).

We highly recommend that you first take a look at the document *"Artery: Large Scale Simulation Environment for ITS Applications"* from Springer Link first to make you familiar with Artery, as this Framework is based on it.

www.irt-systemx.fr



Installation

Requirements:

- Linux 18.04 or higher
- [OMNeT++](#) 5.5.1 or later
- [Artery Framework](#)
- [Visual Studio Code](#) is highly recommended for code editing

More information about Artery installation can be found on the official Artery [website](#).

Installation steps:

Before adding F2MD to artery, we advise you to check your installation first, by launching a scenario provided in the original installation of the Artery framework.

Once you have a clean Artery installation, simply add the F2MD files to the Artery installation folder.

Before launching a scenario, you need to re-build the code. In VSCode, simply right click on the CMakeList file at the root of Artery, and select "Clean re-build all code".

You can also do it manually, as you did to install and make Artery run a scenario.

Launch a scenario

You are now ready to launch your first F2MD scenario!

To do so, move on to the build folder [cd build] and select your scenario:

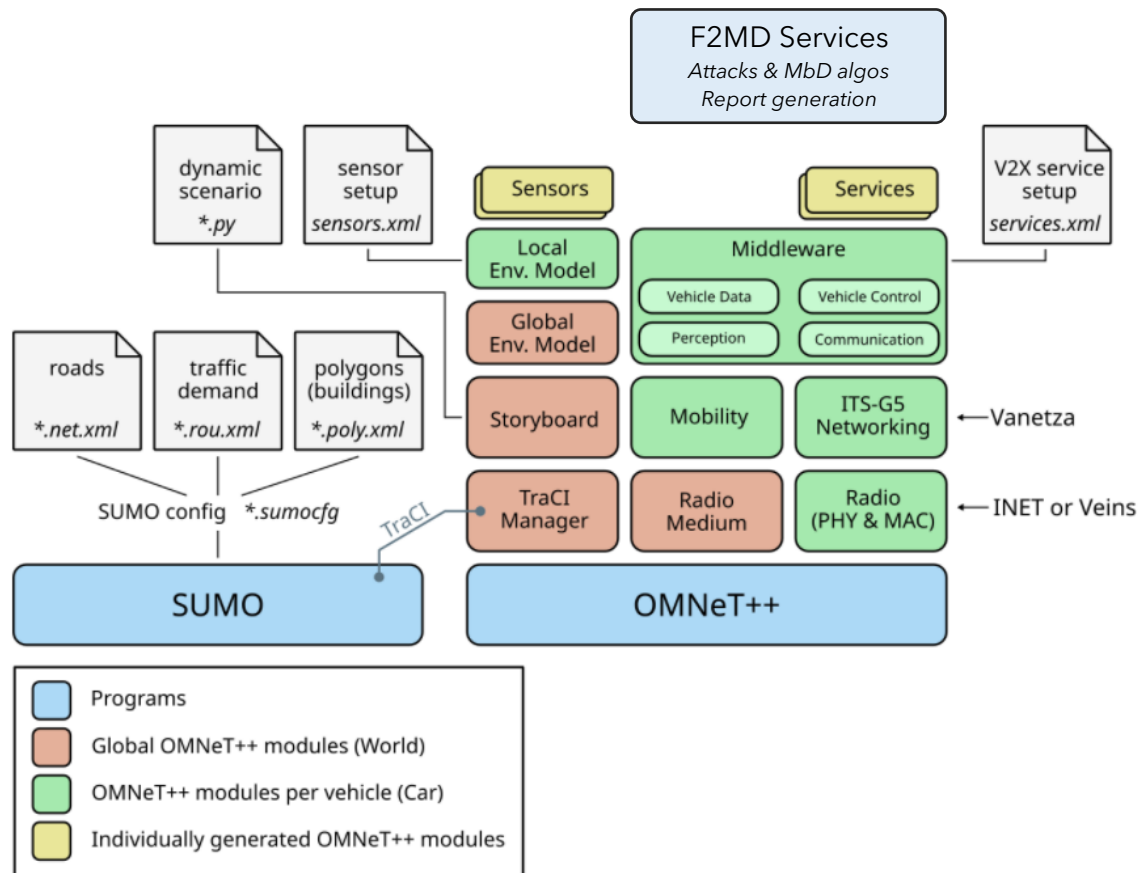
- Make run_F2MD_Highway_CAM
- Make run_F2MD_Highway_CPM
- Make run_F2MD_IRTS

A scenario for CPM sending and receiving but without F2MD module is also available:

- Make run_Highway_CPM

Features

F2MD was developed as a custom service of Artery. As such it uses the following architecture:



As shown in the figure above, we use without any modifications the original architecture of Artery. All the F2MD features are added through a new service named F2MD Service, that can be assigned to any vehicle of the simulation through the *services.xml* file.

Therefore, we highly recommend that you make yourself familiar with all the artery's features

Artery features

Artery is a highly customisable V2X simulation Framework, based on OMNET ++.

You can learn more of its architecture on their [website/architecture](https://www.arteryproject.org/website/architecture).

If you want to know more about the features it offers, we also recommend their [feature guide](https://www.arteryproject.org/feature-guide).

Last but not least, we provide with this documentation a very complete document that explain in details how artery works.

F2MD Features

On top of Artery, F2MD Services allow vehicles to send and receive Cooperative Awareness Messages, and Cooperative Perception messages.

It is an evolution of a previous work, based on Veins. You can find the code of F2MD on Veins [here](#). Please note that the previous version only supports BSM message type.

As a reminder, a CAM is a message sent periodically by connected vehicles in a V2X network, to inform the other vehicles about its own current state (position, speed, heading and so on).

On the other hand, a CPM is designed to contain perception data, acquired by a vehicle through its sensors. These messages are sent periodically as well.

For now, F2MD for CAM actually use the BSM format which is similar, but less complete. In the future, F2MD is planned to use the official CAM format, as defined on the Artery's ASN1 files. In this document we will use the term CAM, even though at the time this is written, F2MD still works with BSMs.

On top of these basic functionalities, the reason why F2MD was developed is to implement misbehaviour detection. As such it also includes the following features:

F2MD for CAM:

- Basic Plausibility Checks on Received Beacons (mdChecks)
- Node Level Plausibility Investigation (mdApplications)
- Real Time Detection Status Output (mdStats)
- Support for Multiple Reporting Mechanisms (mdReport)
- Support for Global Reports Collection and Investigation (HTTP to the Python Server: misbehaviour-authority-server)
- Basic Pseudonym Change Policies (mdPCPolicies)
- Local and Global Misbehaviour Attacks Implementation (mdAttacks)
- Launch Attacks in Real Time (HTTP to the Python Server: attack-server)

F2MD for CPM:

- Basic Plausibility Checks on Received CPM (mdCPMChecks)
- Local Misbehaviour Attacks Implementation (mdCPMAttacks)

Code Architecture

Overall Architecture

Here is a brief code overview. F2MD folders are highlighted.

+ ansible	Ansible playbooks and roles for deployment
+ cmake	Some additional CMake macros
+ docs	Artery documentation
- extern	Third-party components
+ inet	INET Framework
+ vanetza	ITS-G5 network stack
...	
- scenarios	Simulation scenarios
+ artery	Example Artery scenario for demonstration
+ Highway_CAM	Basic F2MD scenario on Highway, with CAM service
+ Highway_CPM	Basic F2MD scenario on Highway, with CPM service
+ F2MD_IRTS	Complex F2MD scenario in a city, with CAM & CPM service
- src	
- artery	
- application	V2X application layer, facilities, services
+ F2MD	F2MD app layer implementation
CPMService.cc/.h	Standalone CPM Service implementation
...	
+ envmod	Environment model with sensors
- inet	Adaptation of INET Framework for V2X
+ gemv2	GEMV ² path loss model (requiring INET)
+ networking	Integration of Vanetza as OMNeT++ modules
+ nic	Generic code to adapt various radio models
+ traci	Integration of SUMO vehicles
...	
- traci	OMNeT++ endpoint for interaction with SUMO
+ sumo	TraCI C++ client API from SUMO project

F2MD Architecture

The files added during this internship are highlighted.

- F2MD	F2MD app layer implementation
- mdFiles	Additional F2MD files
+ mdApplications	Node Level Plausibility Investigation (CAM only)
+ mdAttacks	Local (CPM & CAM) and global (CAM) Misbehaviour attacks
+ mdBase	Various files for basic F2MD functionalities
+ mdChecks	Basic Plausibility Checks on received CAM and CPM
+ mdEnumTypes	Enumeration of all F2MD types (Attacks, checks, ...)
+ mdPCPolicies	Basic Pseudonym Change Policy (CAM only)
+ mdReport	Support for Multiple Reporting Mechanisms (CAM only)
+ mdStats	Real Time Detection Status output (CAM only)
+ mdSupport	Back-End F2MD files

Source Files description

- **Scenario Folder: Artery/Scenario/Scenario_Name/**

Omnetpp.ini: Description of all the simulation parameters. In this file you can personalise the simulation, from basic parameters such as simulation time, to precise F2MD settings such as the number of attackers. These parameters will be transferred to the F2MD_[CAM_or_CPM]Parameters file during the initialisation.

Services.xml: You can choose here the services used by the vehicles in the simulation. By default, F2MD offers 2 services (CAM Service & CPM Service).

Sensors.xml: You can choose here the sensors used by the vehicles in the simulation.

[...].sumo.cfg: SUMO configuration file. Must contain a path to the *net* and *rou* files.

[...].net.xml: Description file of the road network sent to SUMO.

[...].rou.xml: You can choose here the number of vehicle in the simulation, and their attributes (when they appear, at which speed, in which lane, etc...)

- **F2MD Src Folder: Artery/src/artery/application/F2MD**

BSM.msg: As previously explained the initial version of F2MD on veins used a personalised BSM format instead of the official ASN1 CAM format provided by Artery. To make F2MD work on artery without too many changes, we re-defined a BSM message type (BSM.msg). In the future versions, F2MD should move onto the artery CAM format, that follows the ETSI standards.

F2MD_[CAM or CPM]Service.ned: ned file that will store temporarily the omnetpp.ini parameters, before they are saved in the Parameters file at the initialisation. We store the parameters in a separate file (F2MD_[CAM or CPM]Parameters) so that they can be used by any module of the simulation.

F2MD_[CAM or CPM]Service.cc: Main file of the CAM/CPM Service. Contains the methods to send (trigger), receive (indicate) and test reception of CAMs and CPMs.

Concerning the CAMs, for now, this service does not create a CAM on its own, but ask the F2MD Facility to do it - *F2MDFacility.createBSM()* - with or without an attack.

Concerning the CPMs, this service does create the CPM on its own. As such, this service could be used without any F2MD features, just to enable vehicles to send and receive CPM. At the reception of a new message, these services send it to the F2MD Facility for further testing, and display the relevant fields in the console of the simulation. There is also a standalone CPM service with no F2MD features in the main application folder.

F2MD_[CAM or CPM]Facility.cc: Main files of the F2MD Framework. These files contain all the front-end methods that implement the F2MD functionalities, such as adding the attacks, running the checks, and triggering the pseudonym changes (only in the CAM version)

Most of these operations are done using others methods that are implemented in the mdFiles folder.

F2MD CPM

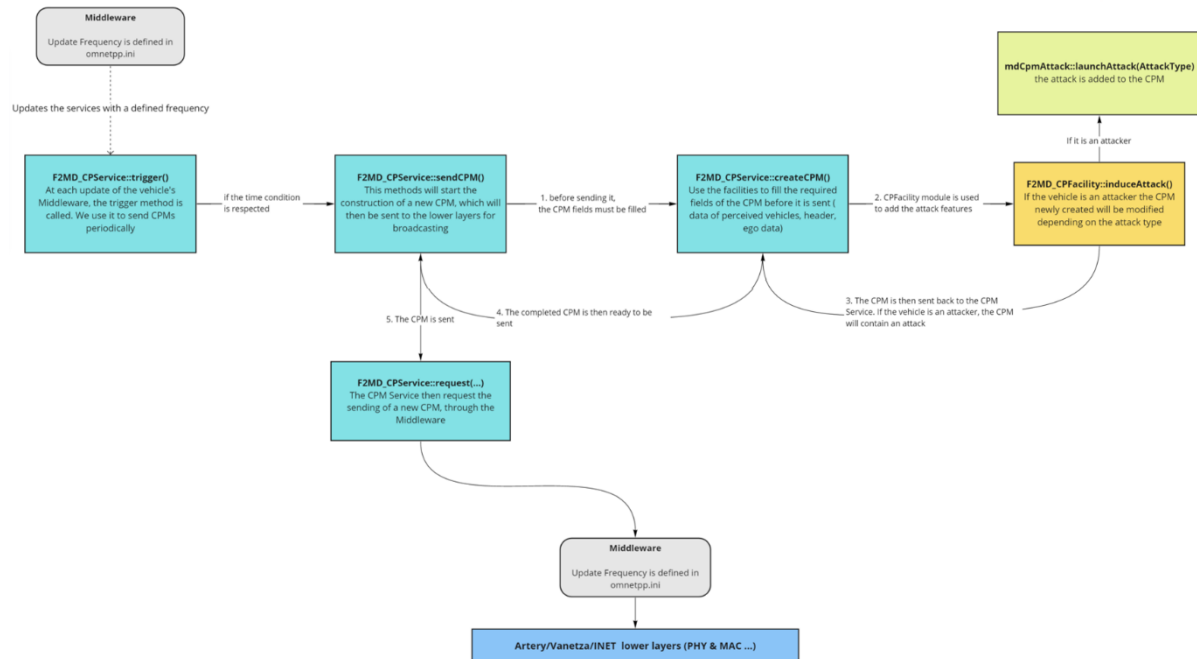
Temporal overview

F2MD for CPM was developed as a service of Artery, that allows vehicle to send and receive CPM, but also to perform attacks and security checks on these messages.

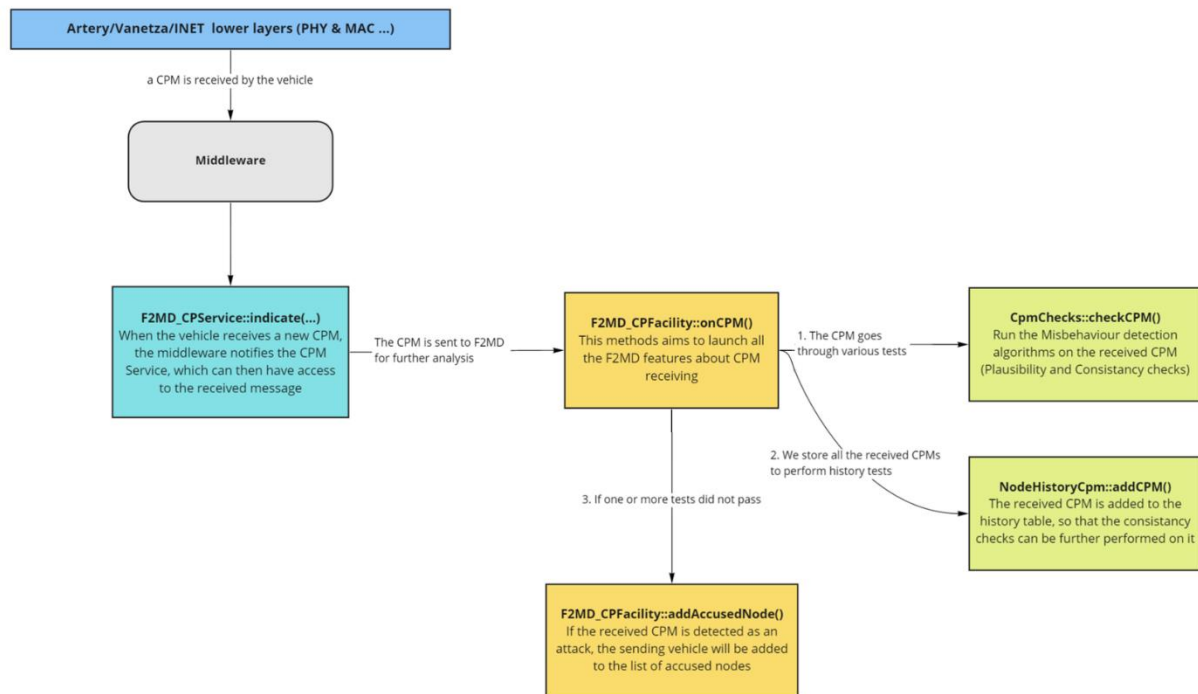
It is firstly composed of a basic CPM Service module, that only handles the sending and the receiving of CPMs. This service then relies on a F2MD Processing Facility, that enables the vehicle to perform malicious attacks and misbehaviour detection.

To make you familiar with how this framework actually works, we propose 2 temporal diagrams, down below. These diagrams are also accessible on [miro](#).

Legend



Temporal diagram for CPM emission, with a potential attack



Temporal diagram for CPM reception, with Misbehaviour detection

These diagrams give an overview of F2MD for CPMs. On each box, you can identify a title, that contains the name of the function used and the source file where it is located, and a subtitle that explains the main purpose of this function.

For more complex features, like misbehaviour detection or attacks implementation, additional files (located in the mdFiles folder) are used. They are explained in the next section of this document

Additional files – mdAttacks/mdCpmAttacks

F2MD for CPM provides 18 new attacks, mainly based on the 8 following ones:



In the omnetpp.ini file, you can choose between several options:

MixCpLocalAttack = True:

- *RandomCpLocalMix = True*: each attacker will use an attack type chosen randomly among all the attack types based on CPM, with potential redundancy.
- *RandomCpLocalMix = False*: each attacker will use a different attack type.

MixCpLocalAttack = False:

- *CP_Local_Attack_type = attackID*: All the attackers will use the attack corresponding to attackID.

During the initialisation of the F2MD Processing Facility, each vehicle that is an attacker will be given an AttackType depending on these parameters.

Additional files – mdChecks/CpmChecks

In order to perform misbehaviour detection, and to detect the previously described attacks, so MbD algorithms have been implemented:

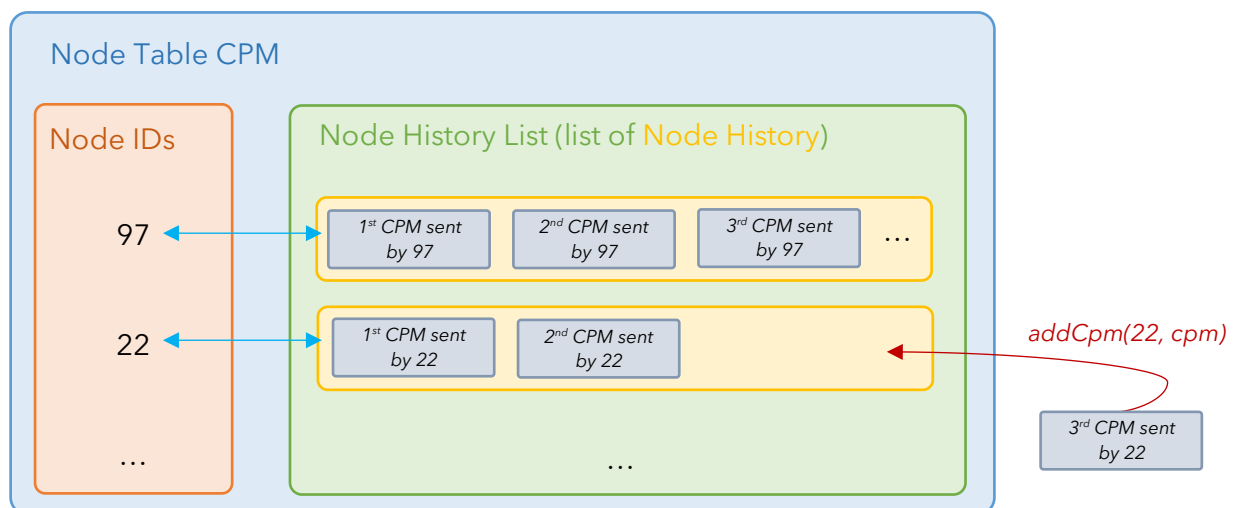
- *Plausibility Checks*, that only checks the plausibility of the data received through the CPM. If the distance or the speed perceived is out of any plausible bounds, the check will fail.
- *Consistency Checks*, that use history to check the consistency of data received through several consecutive CPM from the same neighbor. In our implementation we use two timesteps. If the distance or speed deltas are not consistent, the check will not pass. These checks rely on *NodeTables* for storing CPMs, that are explained in the next section.

More detailed information about these checks can be found in the internship presentation.

Additional files – mdBase/NodeTable & NodeHistory

To store CPMs received for further consistency and predictive tests, we use a *Node Table*.

This node table contains a NodeID list that is linked to a Node History List, which saves the CPMs, as explained below.



To make sure that we do not use unnecessary memory, Node Histories are deleted if they are not updated after a chosen period of time, defined in the `omnetpp.ini` file of the scenario. By default, this value is set to 15 seconds.

The same structure is used in F2MD for CAM.

Data logging

The original F2MD Framework contains a complex and real time logging solution which is explained on the official documentation, and is still included in F2MD for CAM in this version.

In the CPM part, this is not yet implemented. We still offer a logging solution that can export both scalars and vectors.

- To save a unique scalar, the procedure is very simple. You can simply use the function *recordScalar* provided by omnet:

```
F2MD_CPService.cc
[ ... ]

recordScalar("Id", myId);
recordScalar("attackType", attackType);

[ ... ]
```

- To save a vector, things are more complex. Each vector that you want to record must be declared in the ned file. In this example, we export 3 vectors of type double.

```
F2MD_CPService.ned.as
@signal[prediction](type=double);
@signal[percvdObj](type=double);
@signal[senderID](type=double);

@statistic[prediction](source=prediction;record=vector; interpolationmode=none);
@statistic[percvdObj](source=percvdObj;record=vector; interpolationmode=none);
@statistic[senderID](source=senderID;record=vector; interpolationmode=none);
```

Then, the signals must be declared at the beginning of the .cc file.

```
F2MD_CPService.cc
[ ... ]

static const simsignal_t prediction = cComponent::registerSignal("prediction");
static const simsignal_t senderID = cComponent::registerSignal("senderID");
static const simsignal_t percvdObj = cComponent::registerSignal("percvdObj");

[ ... ]
```

Finally, each time you want to add a new value to the recorded vector, you want to use the *emit* function:


```
F2MD_CPService.cc
[ ... ]


emit(prediction, prediction_result);
emit(senderID, mySenderId);
emit(percvdObj, percvdObjcount);


[ ... ]
```

This will emit a signal (either prediction, senderID or percvdObj in our case) containing the value to add, that will be recorded by the statistic declared in the ned file, and then be stored in the final .vec file.

All the vectors and scalars recorded are stored in .sca and .vec files located in *scenarios/scenario_name/results*

 General-#0.sca

 General-#0.vci

 General-#0.vec

For better analysis, we recommend to open these files with omnetpp, that has a module to open and analyse this file. You can next export the chosen data to a csv file for further analysis with a python script for example.

F2MD CAM

F2MD for CAM on artery is the porting of Veins F2MD to Artery. Veins F2MD was originally developed as an application layer of veins

In the way it works, CAM F2MD is very similar to the CPM version, with some additional methods, due to missing features on the CPM version.

The main differences are:

- Instead of using the template service of Artery in the src/application folder (for example Caservice for CAM), we integrate the F2MD Veins implementation of CAM to artery in a specific folder called F2MD. In this folder can be found the F2MD_CAMFacility which not only integrates the kinematic information into the CAM message before its transmission but also perform all the message processing at the reception and includes the attacks at the transmission (and potentially kinematic data imprecision).
- We store and update directly in F2MDCAM_Facility all the kinematic data about the vehicle. As such, when we need such data, as for example when we fill a new CAM, we gather it directly in the attributes of F2MD_CAMFacility (curPostion, curSpeed, etc...), instead of getting it through the Middleware, which would be the optimal way. Again, this was done to make the porting faster. We update them at each emission or reception of a CAM, through updateVehicleState()
- We use the CAM format defined in the BSM.msg file, instead of the official CAM format defined in the ASN1 file.

Additional files - mdAttacks & mdChecks

These parts of the code are pretty similar to what you can find in the CPM part, with a lot more possibilities.

Overall, there are 19 various local attacks, and 1 global attack (MA Stress).

As for the attacks, more Misbehaviour Detection checks are implemented for CAMs, including Kalman Filtering.

You can find the list of these attacks and checks in the mdFiles/mdEnumTypes.

Additional files - mdApplications

This is another difference between F2MD CAM and F2MD CPM.

In the CPM part, if one of the checks did not pass, the vehicle we received the CAM from will be considered as an attacker.

In the CAM part however, all the implemented checks return a factor. The more this factor is closed from 0, the less likely the vehicle is to be an attacker.

After having performed all the checks on the received CAM, a list of factors is obtained. We then use an application to determine, using these factors, if the vehicle must be considered as an attacker or not. The mdApplication folder contains several application types. The application type can be chosen for a given scenario in the omnetpp.ini file, from the following list:

- *ThresholdApp* (by default): if one of the factors is below a threshold ($\sim 0,3$), the vehicle is considered as an attacker.
- *MachineLearningApp*: A deep learning algorithm is used to analyse the factors and determine if the vehicle is an attacker.
- *AggregationApp*: This application is based on the node history. The checks result of certain messages are aggregated with the last n results. A node is reported if the aggregated results fall below a certain threshold.
- *BehavioralApp*
- *CooperativeApp*
- *ExperiApp*

Additional files – mdPCPolicies

Another feature available only for the CAMs is the Pseudonym changes.

Each vehicle in the simulation is given a pseudonym, which is changed. The decision whether to change or not is taken according to one of the following methods:

- *PeriodicalPCP*: Pseudonyms are changed with a constant period of time.
- *DisposablePCP*: Pseudonyms are changed after a chosen number of iterations.
- *DistanceBasedPCP*: Pseudonyms are changed after a chosen distance travelled.
- *Car2carPCP*: Pseudonyms are changed depending on the distance between vehicles.

Additional files – mdReport

With F2MD for CPM, when a node is detected, the only reaction is to change its color in the simulation, to visualize it in real time.

In the CAM part though, things are a bit more complex. When a node is detected as an attacker, a report is built before being sent to a Misbehavior authority. These reports can be chosen to range from a basic report to a report with evidence included to help the MA in its decision.

A report can originally be composed of 3 containers:

- *Header container*, with basic information
- *Source container*, containing the result of the checks
- *Evidence container*, containing the relevant evidence

Based on that, we define several types of reports, including:

- *Base report*: includes only the header container and the source container
- *Beacon report*: includes a base report and the reported beacon in the evidence container
- *Evidence report*: includes a more complete evidence container depending on the type of checks that failed. If a consistency check failed, we include both inconsistent CAMs as an evidence.

These reports are sent to the MA server through HTTP and can be visualized in real time.

We advise you to take a look at the thesis paper about the F2MD framework for more details.

Additional files – mdStats

These files are used to export various statistics about the ongoing simulation

Format conversion

The F2MD framework was originally developed on VEINS. In this work, we created an Artery version of F2MD, without changing any backend file. Therefore, we had to deal with several format incompatibility between VEINS and Artery.

In order to use the already developed F2MD checks and attacks, we had to convert the data in artery format we get through the vehicle API, to the veins format that is used in the original F2MD framework.

Some additional changes are required. Angles must indeed be transformed in radians, and an offset of $-\frac{\pi}{2}$ must be added, as the angle origin is not the same. Speed and Acceleration are also projected on the X and Y axis, in the VeinsCoord format.

The first line explain how we get the data through the Artery vehicle's API, and the second one shows the operations realised to convert it into the Veins format.

We use the **veinsCoord(x,y,z)** to build veins format coordinates.

Here is a list of the conversions realised, in the F2MD_CAMFacility file:

- 1. *Position conversion*: here we simply use the veinsCoord to store the 3 components of the position vector.

$$position_{artery} = vehicle_{api}.getPosition(id)$$

$$position_{veins} = veinsCoord(pos.x, pos.y, pos.z)$$

- 2. *Heading conversion*: here we also use the constructor, but we also convert the angles in radian. The veins format to store angles is very different from the Artery one. First, an offset of $-\frac{\pi}{2}$ must be added. Then, the veins format does not store the actual angle, but 2 factors representing the angles ($\cos(\theta)$ and $-\sin(\theta)$):

$$heading_{artery} = vehicle_{api}.getAngle(id) * \frac{\pi}{180}$$

$$heading_{veins} = veinsCoord\left(\cos\left(angle_{artery} - \frac{\pi}{2}\right), -\sin\left(angle_{artery} - \frac{\pi}{2}\right)\right)$$

- 3. *Speed conversion*: here we simply take the 3 axis components of the speed. As the simulation area is leveled, the Z component of speed is null.

$$speed_{artery} = vehicle_{api}.getSpeed(id)$$

$$speed_{veins} = veinsCoord(speed_{artery} * \cos\left(angle_{artery} - \frac{\pi}{2}\right), speed_{artery} * \sin\left(angle_{artery} - \frac{\pi}{2}\right), 0)$$

- 4. *Acceleration conversion*: exact same conversion as the speed.

$$accel_{artery} = vehicle_{api}.getAccel(id)$$

$$accel_{veins} = veinsCoord(accel_{artery} * \cos\left(angle_{artery} - \frac{\pi}{2}\right), accel_{artery} * \sin\left(angle_{artery} - \frac{\pi}{2}\right), 0)$$

- 5. *GPS Coordinates conversion*: exact same conversion as the position

$$GPSCoord_{artery} = vehicle_{api}.convertGeo(position_{artery})$$

$$GPSCoord_{veins} = veinsCoord(GPSCoord_{artery}.longitude, GPSCoord_{artery}.latitude, 0)$$