Internship report

Simulation of Trust Management in VANET

Yujun JIN

Company supervisors : Ye-Qiong SONG, Runbo SU

School supervisor : Vincent CHEVRIER

August 27, 2022

Contents

Abstract—This report presents a summary of the activities I was involved during an internship at Loria from March 1 to August 30, 2022. I worked in the Simbiot group, and working on cyber-physical systems. We proposed a new Trust Management model to address the new security requirements of Vehicular Ad-hoc Networks (VANET) and seeking its applicability in the real world. In this work, my goal is to build a new trust model that maintains the reliability of VANETs while detecting malicious intrusions or misbehavior from vehicles. We utilize two types of VANET information, namely CAM and DENM, while I build a simulator based on the Veins framework to validate our proposed trust model. internship was a perfect opportunity to apply the programming and networking knowledge I learned in my previous courses. In addition, I was able to develop some additional soft skills such as communication, teamwork, and flexibility. Finally, I learned how research institutions and laboratories operate, and how to search and edit academic articles.

Résumé—Ce rapport présente un résumé des activités auxquelles j'ai participé lors d'un stage au Loria du 1er mars au 30 août 2022. J'ai travaillé dans le groupe Simbiot et travaillé sur les systèmes cyber-physiques. Nous avons proposé un nouveau modèle de gestion de la confiance pour répondre aux nouvelles exigences de sécurité des VANETs et cherchant son applicabilité. Dans ce travail, mon objectif est de construire un nouveau modèle de confiance qui maintient la fiabilité des VANETs tout en détectant les intrusions malveillantes ou les mauvais comportements des véhicules. Nous utilisons deux types d'informations VANET, à savoir CAM et DENM, tandis que je construis un simulateur basé sur le cadre Veins pour valider notre modèle proposé. Ce stage a été une occasion parfaite d'appliquer les connaissances en programmation et en réseau que j'ai acquises dans mes cours précédents. En outre, j'ai pu développer d'autres compétences générales telles que la communication, le travail d'équipe et la flexibilité. Enfin, j'ai appris comment fonctionnent les institutions et les laboratoires de recherche, et comment rechercher et éditer des articles académiques.

Acknowledgements

The internship opportunity I had with Loria was a great chance for learning and experiencing of research life. Therefore, I consider myself a very lucky individual as I was provided with an opportunity to be a part of it. I am also grateful for having a chance to meet so many wonderful colleagues who led me through this internship period.

I would like to express my gratitude to my supervisors of this internship, Mr. Ye-Qiong SONG, and Mr. Runbo SU. I would like to thank them for their support, guidance, help, and advice.

I thank my dearest parents who have always been there for me. Their unconditional support and encouragement have been a great help.

I perceive this opportunity as a big milestone in my career development. I will strive to use gained skills and knowledge in the best possible way, and I will continue to work on their improvement, in order to attain my desired career objectives.

1 Introduction

1.1 Loria

Loria is the French acronym for the “Lorraine Research Laboratory in Computer Science and its Applications” and is a research unit (UMR 7503), common to CNRS, the University of Lorraine, and INRIA. This unit was officially created in 1997.

Loria’s missions mainly deal with fundamental and applied research in computer sciences. This internship is conducted in the Networks, Systems and Services department SIMBIOT group. The goal of the team is the design and validation of “smart” Cyber-Physical systems. The members of SIMBIOT concentrate their work on the adaptability properties of cyber-physical systems, in terms of calculations and communications, in order to increase their autonomy capacity.

1.2 Introduction of the project

The past two decades have seen the rapid development of Intelligent Transport Systems (ITS) worldwide. The subclass of ITS called Cooperative Intelligent Transportation System (C-ITS) is a cyber-physical system characterized by a large deployment of networked devices equipped with both sensors and actuators [1], that allows vehicles to communicate with each other (V2V) and also with the infrastructure components (V2I), which is commonly referred to as a Vehicular Adhoc Network (VANET) or V2X. However, this cyber-physical system brings new security requirements that challenge traditional embedded systems, where individual nodes interact with the real world in strongly constrained environments. In recent years, trust management (TM) is proposed as a feasible solution to meet this challenge, and it has drawn increasing attention from researchers in academia and industry companies. However, most of the current related research on TM in VANET stays on the theoretical and algorithmic steps. Only a few models are simulated with the road traffic simulation. Unfortunately, not much more traffic details are given in those simulations, which is the key to the model's applicability in the real world.

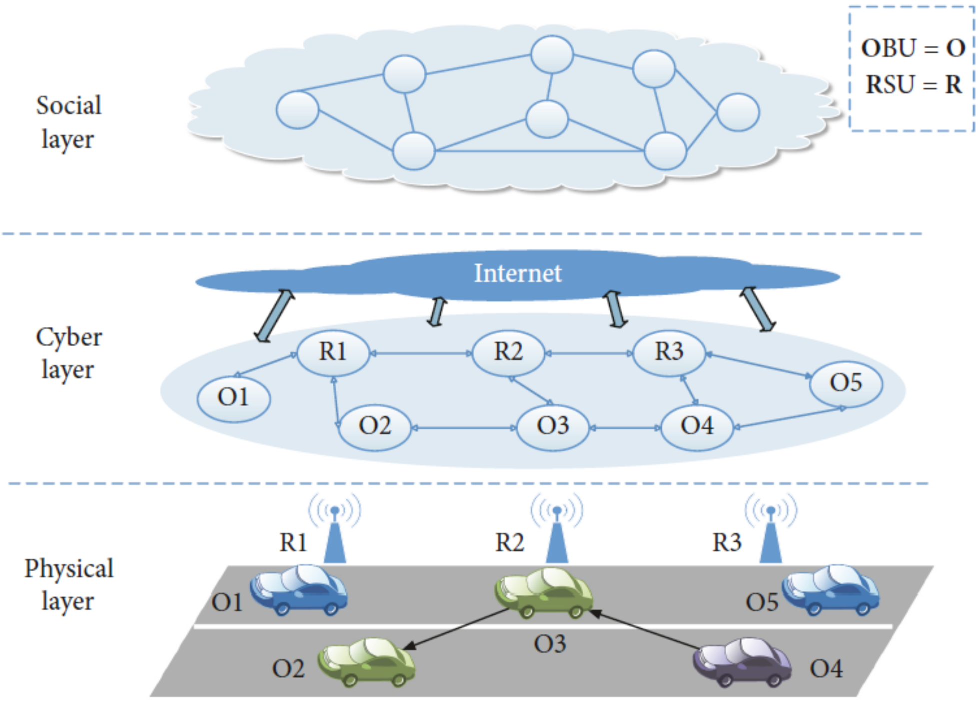

Figure 1.1: Architecture of VANET

In this work, we aim to implement a new trust model to maintain the reliability of VANET and simultaneously detect malicious intrusions or misbehavior from vehicles. We utilize two types of VANET messages standardized by the European Telecommunications Standards Institute (ETSI) for vehicular network communication, which are Cooperative Awareness Message (CAM) and Decentralised Event Notification Message (DENM). These two types of messages are designed to exchange relevant data cooperatively: each vehicle broadcasts CAM periodically containing its speed, position, type, acceleration, etc., to inform about the traffic situation, and DENM will be generated when a specific event is detected. At the same time, we built a simulator that extends the Veins framework [2], an open-source framework for running vehicular network simulations, To validate our proposed trust model.

2
State of the art

2.1 VANET and related security issues

2.1.1 VANET

Vehicular Ad-hoc Networks (VANETs) are created by applying the principles of Mobile Ad-hoc Networks (MANETs) – the spontaneous creation of a wireless network of mobile devices – to the domain of vehicles. VANETs were first mentioned and introduced [3] in 2001 under "car-to-car ad-hoc mobile communication and networking" applications, where networks can be formed and information can be relayed among cars. It was shown that vehicle-to-vehicle and vehicle-to-roadside communications architectures will co-exist in VANETs to provide road safety, navigation, and other roadside services. VANETs are a key part of the intelligent transportation systems (ITS) framework. Sometimes, VANETs are referred as Intelligent Transportation Networks.[3] They are understood as having evolved into a broader "Internet of vehicles". [4] which itself is expected to ultimately evolve into an "Internet of autonomous vehicles".

While, in the early 2000s, VANETs were seen as a mere one-to-one application of MANET principles, they have since then developed into a field of research in their own right. By 2015, the term VANET became mostly synonymous with the more generic term inter-vehicle communication (IVC), although the focus remains on the aspect of spontaneous networking, much less on the use of infrastructures like Road Side Units (RSUs) or cellular networks.

VANETs support a wide range of applications – from simple one-hop information dissemination of, e.g., cooperative awareness messages (CAMs) to multi-hop dissemination of messages over vast distances. Most of the concerns of interest to MANETs are of interest in VANETs, but the details differ. Rather than moving at random, vehicles tend to move in an organized fashion. The interactions with roadside equipment can likewise be characterized fairly accurately. And finally, most vehicles are restricted in their range of motion, for example by being constrained to follow a paved highway.

VANETs can use any wireless networking technology as their basis. The most prominent are short-range radio technologies are IEEE 802 11p and DSRC. In addition, cellular technologies or LTE and 5G can be used for VANETs.

Prior to the implementation of VANETs on the roads, realistic computer simulations of VANETs using a combination of Urban Mobility simulation and Network simulation are necessary. Typically open source simulator like SUMO(which handles road traffic simulation) is combined with a network simulator like Veins, TETCOS NetSim, or NS-2 to study the performance of VANETs.

2.1.2 Security issues in VANET

Security is an important issue for VANET, especially for some security-sensitive applications related to the safety of life or property of traffic participants. To secure an ad-hoc network, we need to consider the following attributes as criteria to measure security which include availability, confidentiality, integrity, and authentication.

Availability

The availability deals with network services for all nodes comprises of bandwidth and connectivity. In order to encounter the availability issues, prevention and detection technique using a group signature scheme has been introduced. The scheme is focusing on the availability of exchanging messages between vehicles and RSUs. When the attack causes network unavailability, the proposed technique still survives due to interconnection using public and private keys between RSUs and vehicles.

Authentication

Authentication is the verification of the identity between vehicles and RSUs and the validation of the integrity of the information exchange. Additionally, it ensures that all vehicles are the right vehicle to communicate within the network. Public or private keys with CA are proposed to establish connections between vehicles and RSUs. On the other hand, a password is used to access the RSUs is also an authentication method.

Integrity

Data integrity is the assurance that the data received by nodes, RSUs and AS is the same as what has been generated during the exchanges of the message. In order to protect the integrity of the message, digital signature which is integrated with password access is used.

Confidentiality

Confidentiality ensures that classified information in the network can never disclose to unidentified entities. It also prevents unauthorized access to confidential information such as name, plate number and location. The most popular technique, pseudonyms are used to preserved privacy in vehicular networks. Each vehicle node will have multiple key pairs with encryption. Messages are encrypted or signed using different pseudo and these pseudo has not linked to the vehicle node but relevant authority has access to it. Vehicle need to obtain new pseudo from RSUs before the earlier pseudo expires.

2.1.3 CAM and DENM

V2X systems are supported by wireless networks for the exchange of information between vehicles (V2V) and roadside infrastructure (V2I). The V2X system enables a wide range of beneficial use cases. Road safety and traffic efficiency use cases are appealing as they hold potential for meeting European Union [i.1] societal objectives. Interoperability is an important aspect to be ensured by the V2X system at different OSI layers. At the facilities layer in particular, basic common functionalities are defined in order to ensure the correct system functioning and to satisfy the interoperability requirement. Respecting common functionalities allows correct and efficient information exchange between nodes participating in V2X networks. This requirement is met by identifying a set of basic functional components at the facilities layer.

ETSI EN 302 637-2 specifies the Cooperative Awareness Basic Service, which provides by means of periodic sending of status data a cooperative awareness to neighboring nodes. Quality requirements are also proposed for this mandatory facility in order to provide reliable component performance for application development. According to this standard, CAM is a multi-casting one-hop and one-way message standard, and the CAM-based communication is without request, reply, or forwarding. It also means that transmission failure cannot be detected.

ETSI EN 302 637-3 defines the decentralized environmental notification (DEN) basic service that supports the Road Hazard Warning (RHW) application. The DEN basic service is an application support facility provided by the facilities layer. It constructs manages and processes the Decentralized Environmental Notification Message (DENM). The construction of a DENM is triggered by an ITS station application. A DENM contains information related to a road hazard or abnormal traffic conditions, such as its type and its position. The DEN basic service delivers the

DENM as payload to the ITS networking & transport layer for the message dissemination. Typically for an ITS application, a DENM is disseminated to ITS stations that are located in a geographic area through direct vehicle-to-vehicle or vehicle-to-infrastructure communications. At the receiving side, the DEN basic service of an receiving ITS station processes the received DENM and provides the DENM content to an ITS station application. This ITS station application may present the information to the driver if information of the road hazard or traffic condition is assessed to be relevant to the driver. The driver is then able to take appropriate actions to react to the situation accordingly.

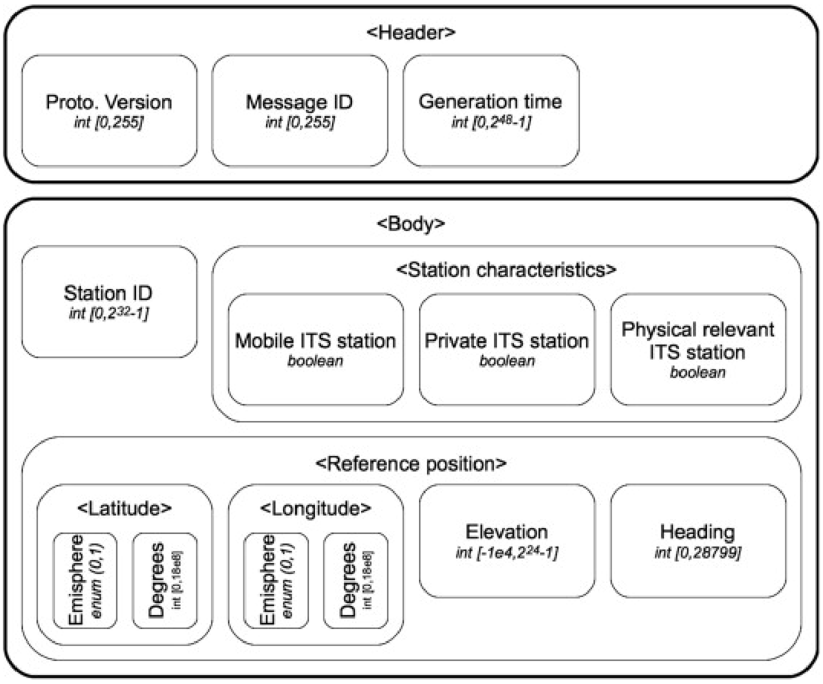

Figure 2.1: CAM structure

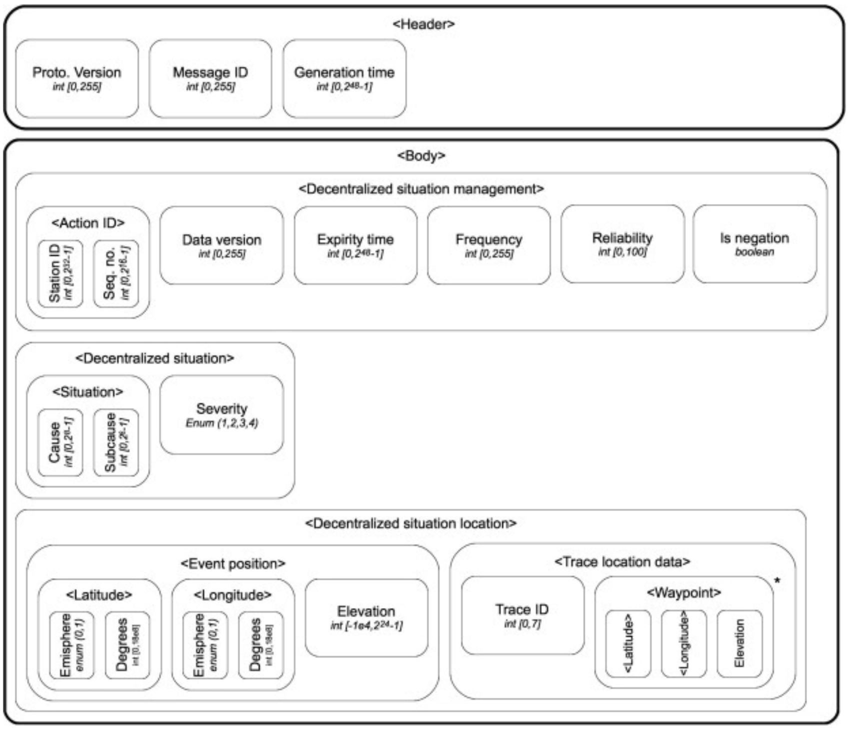

Figure 2.2: DENM structure

CAM and DENM have both confidence information, which may be considered the accuracy of the data. Meanwhile, these parameters can also be linked to the trust values as a reference for the algorithm.

2.2 Trust management (TM) and TM in VANET

2.2.1 Overview of TM

There has been a lot of research and development in the field of computational trust in the past decade. Trust seems to essentially be a means for people to deal with uncertainty about the future and their interaction partners [5]. As the definition implies, it is affected by the pre-existing experience. In VANET, we focus more on evaluating the quality of information that is sent by nodes, to cope with reports from malicious nodes which may compromise the network. For example, consider a node who reports the roads on his path as congested, hoping that other nodes would avoid using these roads, thus clearing the way. Therefore, the goal of incorporating trust is to allow each node in a VANET to detect dishonest nodes and malicious data sent by these dishonest nodes, and to give incentives for these nodes to behave honestly and discourage self-interested behavior [6].

2.2.2 Applying TM in VANET

Another key challenge in modeling trust in a VANET environment is that a VANET is a decentralized, open system i.e. there is no centralized infrastructure and peers may join and leave the network any time respectively. If a peer is interacting with a vehicle now, it is not guaranteed to interact with the same vehicle in the future. Also, information about road condition is rapidly changing in VANET environments, this also brings out an important challenge that the information received from VANETs needs to be evaluated in a particular context. The two key context elements in VANETs are location and time. Information which is closer in time and location of an event is of more relevance [6].

Van der Heijden et al. [7] provided a comprehensive survey of misbehavior detection mechanisms based on attacker behavior and information analysis. It provides a certain reference for the follow-up research of behavioral trust. Kerrache et al. [8] proposed a recommendation-based trust model called T-VNets. They discussed separately vehicle-to-vehicle trust and RSU(Road-Side Unit)-to-vehicle trust. In their model, RSUs can have a quasi-global view of vehicles. This mechanism gives RSUs higher authority but also causes more damage when an attacker forges an RSU to access VANET. Furthermore, they proposed an algorithm dealing with DENM communication, but the DENM utilization is missing in the simulation part. Santa et al. [9] conducted an experimental evaluation of CAM and DENM information services in a test bed deployed at the campus of the University of Murcia. This work provides a performance analysis of CAM and DENM messaging services under a real driving scenario without involving security schemes such as TM. Liu et al. [10] presented a novel Lightweight Self-Organized Trust (LSOT) model containing certificate-based trust and recommendation-based trust evaluations. The metrics of certificate-based trust include number weight, time decay weight, and context weight. However, the details of the road traffic are not given in the part of simulations. Avleen et al. [11] designed a fuzzy-based trust model for reliable packet forwarding in VANET. Relaying nodes in their proposed model are selected by each vehicle based on the calculated trust using a fuzzy inference engine. This leads to a topic worthy of study: a fuzzy-based mechanism needs to build secure optimal decisions from the prior knowledge of real-time traffic information, but it is unclear what kind of prior knowledge is helpful for distinguishing good or bad driving behavior in VANET. Cheng et al. [12] proposed a new approach of trust assessment based on three-valued subjective logic, but this approach is only used to determine the message propagation path without dealing with trust evaluation.

From the above analysis, the current trust models implemented in VAENT are subject to the limitation that no model is combined with specific road traffic simulation. To overcome this limitation, we design a novel trust model specifically designed for CAM and DENM applications and build a simulator based on the Veins framework and SUMO traffic simulation suite.

3 Project progress

3.1 TM model

In our work, we consider two types of trust: **Direct Trust**(T_d) and **Indirect Trust**(T_i). The former consists of communication trust and behavior trust to evaluate neighboring nodes, and the latter is on the basis of direct trust computed from neighboring nodes to evaluate nodes out of the evaluator node's communication zone. Due to the scalability and distributed architecture in VANET, we only simulate the indirect trust in the case of DENM communication, more precisely, when cooperative communication between vehicles is necessary to transfer DENM messages.

3.1.1 Direct trust

The direct trust in our work is mainly of three types: **Communication Trust**(T_c) describes the quality of communication, such as Quality of Service (QoS); **Behavior Trust**(T_b) monitors and assesses vehicle's mobility including speed, direction, acceleration, etc; And **Social Level Trust**(T_s) evaluates the VANET in a social perspective [13], e.g., the connectivity counting the number of neighboring reachable vehicles, the vehicle's type if belonging to the same production batch, etc.

Communication trust

Communication trust can be affected by numerous factors in QoS: the communication success rate, the freshness of the message, etc. As defined in the CAM standard, each vehicle receives passively CAM messages from others in a single hop. Moreover, the CAM message may be generated in an unstable manner due to the high-dynamic nature of VANET and the complex road traffic situation. Based on the above discussion, we take only time decay and the number of communications into consideration for the communication trust assessment.

(1)Time decay measuring the freshness of the message p_1

By convention, the higher freshness of the message from the evaluated node, the more trustful it is. To achieve this, the exponential time decay model can be employed to weigh the CAM information in terms of the timestamp of the message received. The weight for n^{th} CAM $w[n]$ is:

$$w[n] = \rho^{t-t_n} \tag{3.1}$$

where $\rho \in]0,1[$ is the decay factor, it reflects the importance of the history, i.e. $\rho = 0.5$ indicates that the trust in the CAM drops by half every second, t is the current time and t_n is the timestamp of n^{th} CAM.

Assuming that the transmission frequency is one second, the discrete weighted sum of the decay function in time is equal to the convolution with $u[n]$ and its value converges to $\frac{1}{1-\rho}$. So as the first part of communication trust, p_1 should be:

$$p_1 = (1-\rho)\sum_{n=1}^{N} \rho^{t-t_n} \tag{3.2}$$

(2)Number of communication p_2

Imagine an attack scenario where an attacker tries to re-communicate with a new identity to regain the trust value. In this scenario, the direct trust value of the new attacker identity will rise less quickly than a known member that is reconnecting. Therefore we consider the number of communications as a factor in direct trust because apparently, a reconnected known member has more communication records than a newcomer. So as the number of communications increases, the value of the p_2 needs to increase monotonically and converge to one but not grow too fast when the number is small. Thus we define p_2 as:

$$p_2 = \rho^{\frac{\lambda}{n}}, \quad \lambda \in R_+ \tag{3.3}$$

λ can be considered as scaling factor on the horizontal axis, it characterize how many communications will bring p_2 to a higher value, i.e. $\rho = 0.5, \lambda = 5$ indicates that the reliability of this part is 50% in the fifth communication.

(3)Comprehensive formula

In general, we default that both the time decay and the number of communications contribute the same to communication trust, so the communication trust can be given by the following comprehensive formula:

$$T_c = (p_1 * p_2)^{\frac{1}{2}} \tag{3.4}$$

Behavior trust

There are many ways to abstract driving behavior and quantify behavior trust: statistical methods e.g. similarity with car group, Bayesian distributions, modeling driver behavior based on driving data, fuzzy logic, etc. In our work, behavior trust values are generated from vehicle distances.

There are two distances considered: safe distance(d_s) and communication distance(d_c). Usually, the communication distance is far greater than the safe distance. The behavior trust value initiates with 0 and varies from 0 to 1. In one period, if the source position is in the trust zone, i.e. between safe distance and communication distance, then its score grows by incrementing $+\delta$ until 1, otherwise, the score reduces a small amount $-\delta$ until 0. The principle applied by this model is that when a car gets too close for a while, it must no longer be trusted.

The two-second rule is a rule of thumb that tells a driver the minimum distance needed to reduce the risk of collision under ideal driving conditions. The allotted two seconds is a safety buffer, to allow the following driver time to respond [14]. The rule is quite convenient to express using the speed in m/s: $d_s = 2v$.

Thus, the size of the trust zone is dynamic, and also plays a role in filtering vehicles of similar speed. Figure 3.1 describes an application scenario for an evaluator and its annular trust zone between safe distance and communication distance.

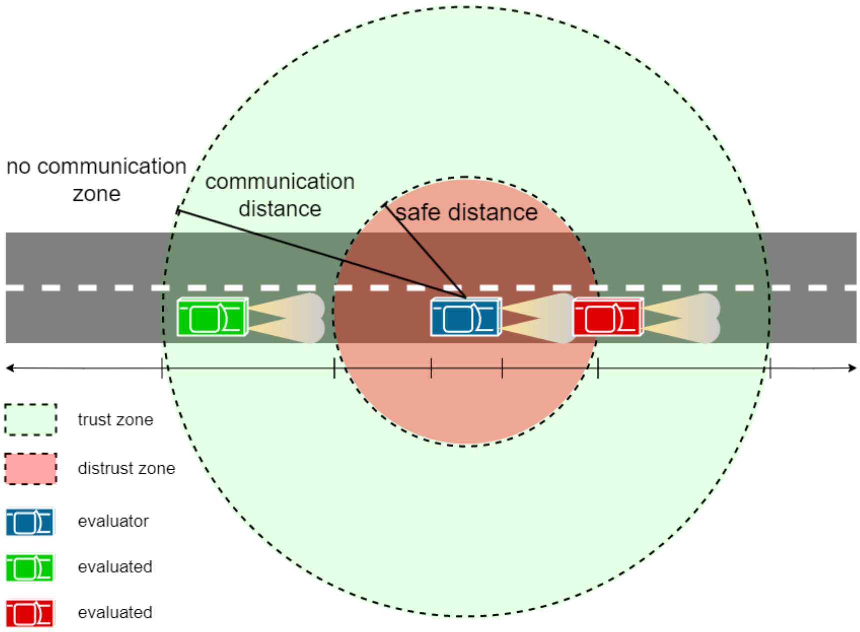

Figure 3.1: Behavior trust model based on car distances

Comprehensive direct trust

The calculation method of comprehensive trust value is also an interesting topic and there are various techniques that can be applied. Some commonly used techniques include weighted average, fuzzy logic, etc.

As introduced in section 2, we use weight average with the same static weight to calculate direct trust value i.e. $T_d = \frac{T_c + T_b}{2}$.

3.2 Simulation

We present first some related software and tutorials and software running environment, then the simulator architecture and implementation.

3.2.1 Related software and tutorials

- Veins

  Veins is an open-source framework for running vehicular network simulations. It is based on two well-established simulators: OMNeT++, an event-based data communication simulator, and SUMO, a road traffic simulator. Veins extends the two simulators mentioned above to provide a comprehensive suite of models for Inter-Vehicular Communication(IVC) simulation.

- Artery & Vanetza

  Artery framework enables V2X simulations based on ETSI ITS-G5 protocols like GeoNetworking and BTP. Artery's middleware includes some basic services, such as CAM and DENM. Vanetza is an open-source implementation of the ETSI ITS-G5 protocol stack. They were initially considered tools but were ultimately not selected due to the difficulty of using the CMake build system.

- Manual of Sumo/Matlab/Veins/INET/OMNeT++ Programming and interfacing, Tamás Ormándi, 2021 [15]
- VANET and OMNeT++ Tutorial Youtube Video Series by Dr. Joanne Skiles

  https://www.youtube.com/watch?v=tCs-K9AkDrQ&list=PLaBPUIXZ8s4AwAk5EelikvvyG4EzX2hpx

3.2.2 Software running environment

Instant Veins 5.2-i1 https://veins.car2x.org/documentation/instant-veins/

Instant Veins Virtual Machine is a virtual machine that integrates almost all the pre-installed software related to Veins simulation:

- Simulation modules
  - Veins 5.1
  - INET Framework 4.2.2
  - SimuLTE 1.2.0(plus a backported patch, 23c0936e31)
  - Veins_INET included with Veins 5.1
- Software
  - OMNeT++ 5.6.2
  - SUMO 1.8.0
  - Cookiecutter 1.6.0 for cookiecutter-veins-project
- Operating system
  - Debian 10, Linux 4, GNOME 3

The virtual machine avoids the trouble of configuring the development environment under different hardware or systems. Instant Veins is a suitable development platform for non-large scale projects or research, it is easy to use and quite friendly to novices and beginners.

3.2.3 Simulator architecture

The simulator is extended from the example of veins_inet subproject. On this basis, we generate output files for analyzing trust value with MATLAB.

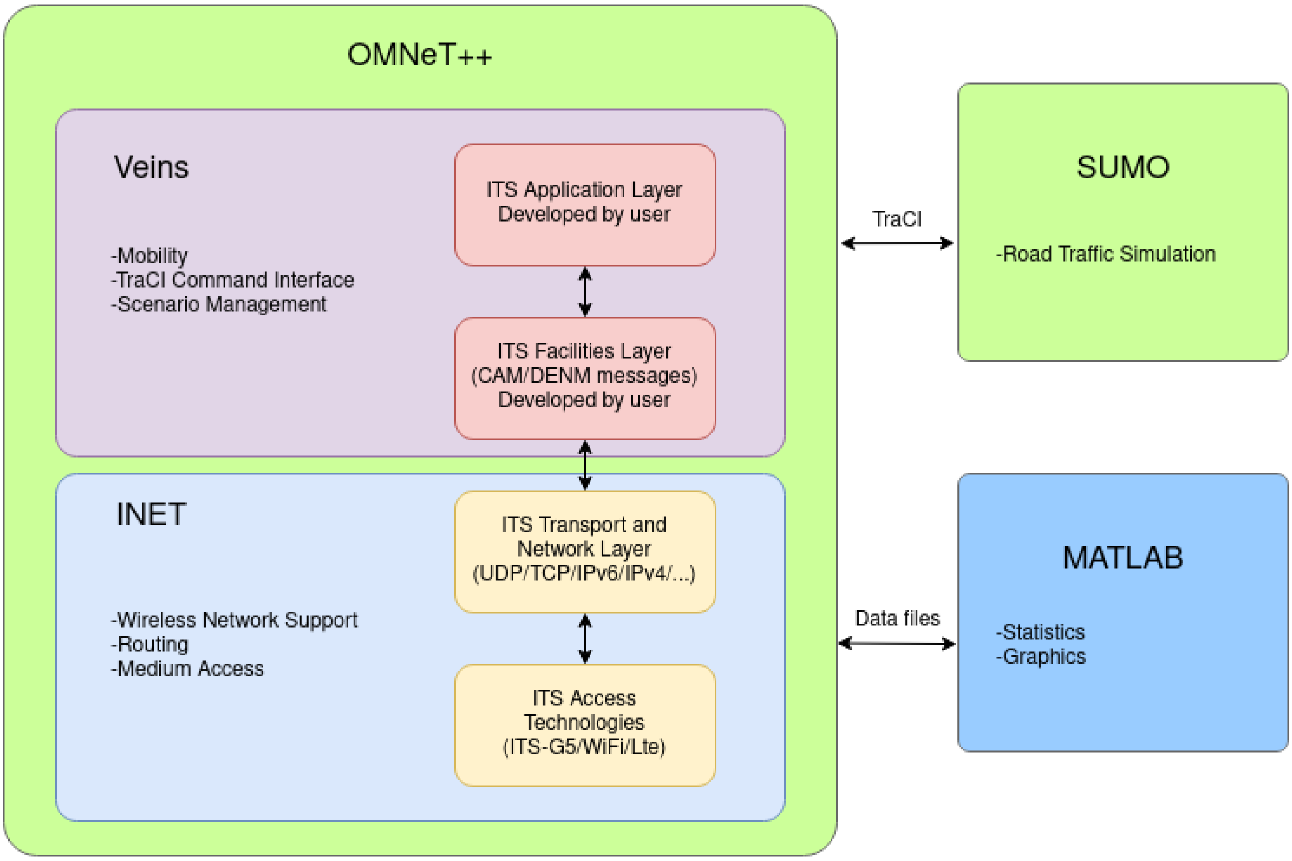

Figure 3.2: architecture of simulator

3.2.4 Simulator implementation

Creating a custom project refer to veins_inet subproject

Following the Instant Veins tutorial steps, the Veins example is ready to run. It can be run once to check if Instant Veins is configured successfully. After this, a custom project, called Trust_Management in this report, can be created with Veins and INET.

In the simulation, Veins contains the MAC(Media Access Control) and the physical layer of the communication, and the INET framework contains the protocols and application layer to communicate through the "wlan0" interface with the UDP protocol. To achieve this, the INET project and Veins project should be imported as Existing Projects into Workspace, then the custom project will reference them and based on the veins_inet subproject in the Veins example. For this, the "veins_inet" folder should be copied (located in /veins/subprojects/veins_inet/src) to Trust_Management/src and all the package paths in the NED files should be changed. More details on creating custom Veins projects can be found in [15] Section 7 and will not be repeated in this report.

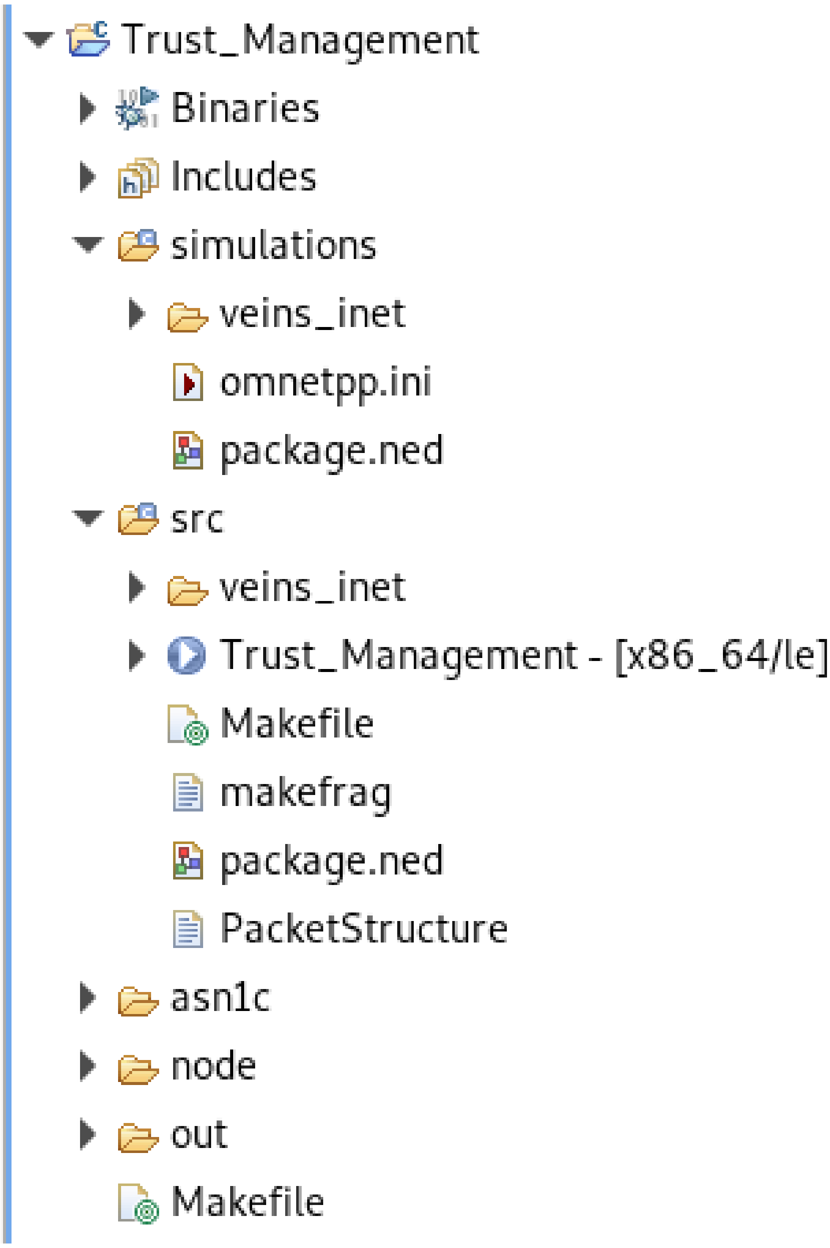

Figure 3.3: Structure of project

Implementation of CAM and DENM

In this simulation, CAM and DENM are created as an <inet::FieldsChunk> and will be processed by a series of special functions under the INET framework to be enveloped as UDP packets that are transported between nodes (vehicles).

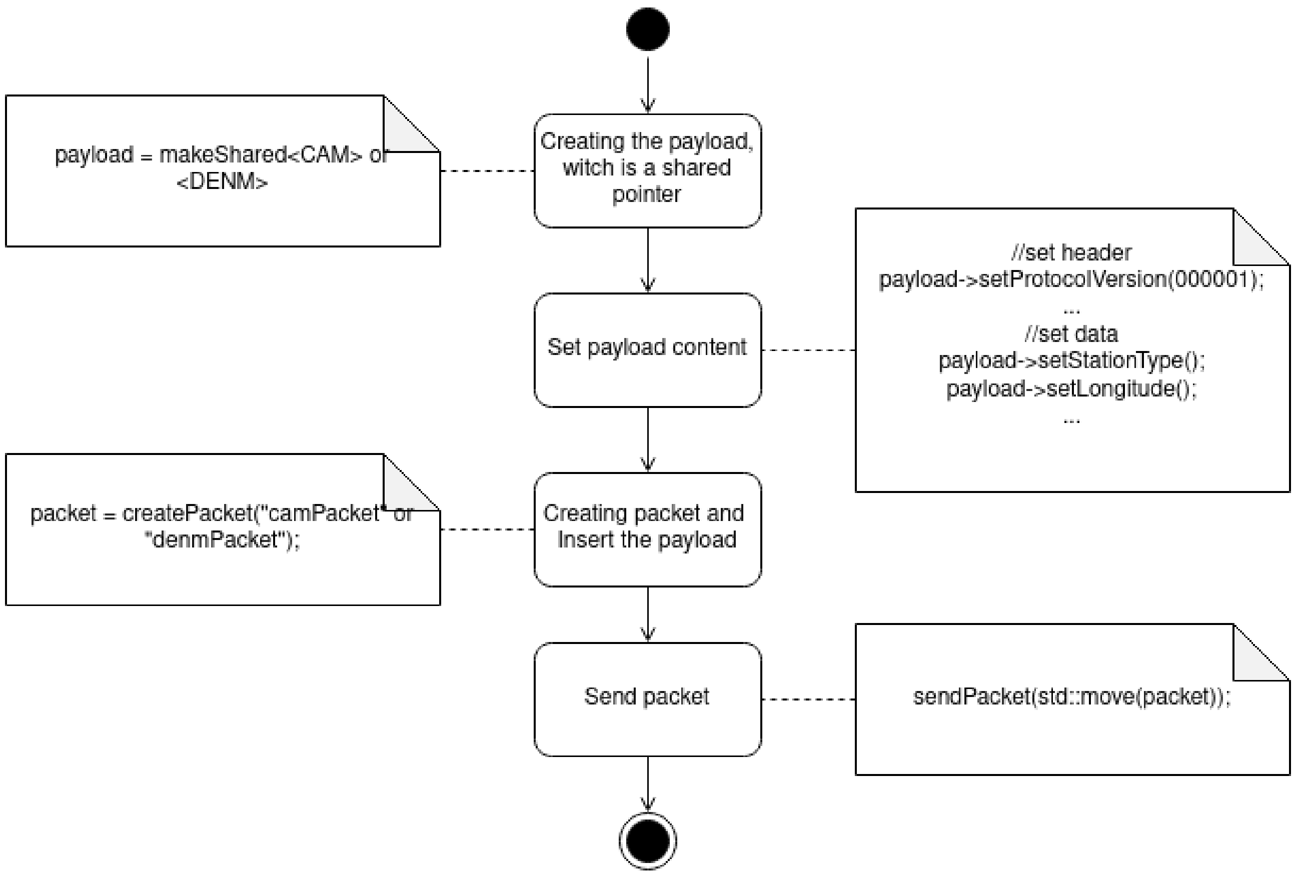

Figure 3.4: Flowchart of message sending

The implementation of the CAM is given below:

```
import inet.common.INETDefs;
import inet.common.packet.chunk.Chunk;

class CooperativeAwarenessMessage extends inet::FieldsChunk {
    // ItsPduHeader
    long            protocolVersion;
    long            messageID;
    long            stationID;
    // GenDeltaTime
    long            generationDeltaTime;
    // cam Parameters
    // BasicContainer
    long            stationType;
    // ReferencePosition
```

```
    long        latitude;
    long        longitude;
    long        altitude;
    // HighFrequencyContainer
    long        heading;
    long        speed;
    long        driveDirection;
    long        vehicleLength;
    long        vehicleWidth;
    long        longitudinalAcceleration;
    long        yawRate;
}
```

And the implementation of the DENM:

```
import inet.common.INETDefs;
import inet.common.packet.chunk.Chunk;

class DecentralizedEnvNotificationMessage extends inet::FieldsChunk {
    string      roadId;
    string      laneId;
    // ItsPduHeader
    long        protocolVersion;
    long        messageID;
    long        stationID;
    // DENM
    // ManagementContainer
    long        actionID;
    long        detectionTime;
    long        referenceTime;
    long        eventPositionLong;
    long        eventPositionLat;
    long        validityDuration;
    long        stationType;
    // SituationContainer
```

```
    long          informationQuality;
    long          eventType;
    long          linkedCause;
    // LocationContainer
    long          eventSpeed;
    long          eventPositionHeading;
    long          Traces;
    long          roadType;
}
```

Defined by ETSI EN 302 637-2 V1.4.1 and ETSI EN 302 637-3 V1.3.1, CAM and DENM shall be serialized and deserialized by ASN1 (Abstract Syntax Notation One) Specification for cross-platform needs. However, the serialization step is not relevant to the purpose of this simulation, in addition, the generic ASN1 C library is not fully compatible with the OMNeT++ project (C++), so this part is omitted in this work.

Data structures used in the simulation

In the envisaged model, the trust value of a node changes over time, just as trust between people in social networks usually grows over time. So naturally, every node needs vehicle info storage. Considering the dynamic nature of VANET, a linked list is chosen as the basic structure to facilitate the insertion and deletion of the information at random locations, and a deque(double-ended queue) is chosen for storage of the series of path points.

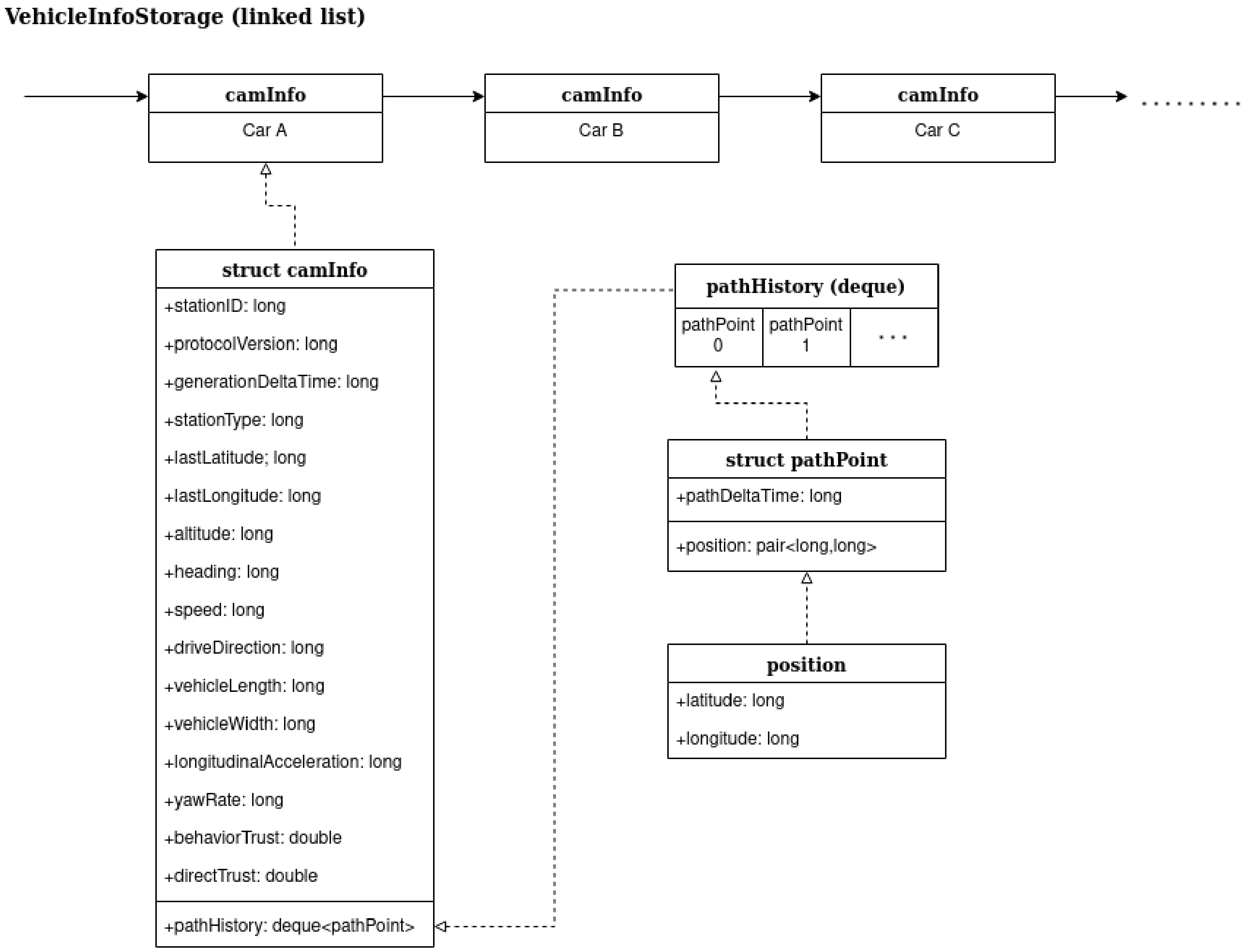

Figure 3.5: Vehicle Info Storage

Implementation of direct trust

The code for generate the communication trust is given below:

```
    double rho = 0.5; // Attenuation factor for 1 second
    double lambda = 5.0; // Scaling factor
/*---------------- Communication Trust ----------------*/
    p1 = 0; //part of time decay
    p2 = 0; //part of number of Communication
    Tn = (long)(simTime().dbl()*100);
    int pathHistorySize = camInfo.pathHistory.size();
    int consideredSize = 5; //consider the latest 5 pathpoints

    for (int i=0; i<(pathHistorySize>consideredSize ? consideredSize : path
```

```
    HistorySize); i++){
        Tk = camInfo.pathHistory[i].pathDeltaTime;
        //apply equation (2) and uniform time unit from 10ms to seconds
        p1 = p1 + pow(rho,(Tn-Tk)/100);
    }
    p1 = p1*(1-rho);

    if (pathHistorySize==1) {
        p2 = 0;
    }
    else {
        p2 = pow(rho,(lambda/pathHistorySize));
    }

    ct = sqrt(p1*p2); // communication trust
```

And the code for behavior trust:

```
double VehicleApplication::generateBehaviorTrust(struct camInfo camInfo){
    double bt = 0.0;   // Behavior Trust
    double p1,p2,d;

    p1 = this->getVehicleAtLong().first - camInfo.lastLongitude;
    p2 = this->getVehicleAtLong().second - camInfo.lastLatitude;
    d = sqrt(pow(p1/1000,2)+pow(p2/1000,2));
    // communication distance = 400m
    if (d>traciVehicle->getSpeed()*2 && d<400){
        if (camInfo.behaviorTrust<0.9){
            bt = camInfo.behaviorTrust + 0.1;
        }
        else {
            bt = camInfo.behaviorTrust;
        }
    }
    else {
```

```
        if (camInfo.behaviorTrust>0.2){
            bt = camInfo.behaviorTrust - 0.2;
        }
    }
    return bt;
}
```

Application layer

The application layer is the core of the simulator, it defines what the vehicle should do and when it does. Thus it will change as the scenario changes. An example of *VehicleApplication.h* and *VehicleApplication.cc* of a scenario containing only CAM communication used to validate the trust model previously mentioned is given in the appendix B and C.

3.3 Performance evaluation

3.3.1 Scenario setup

Our first scenario is just an overtaking scenario on a loop. Three cars are traveling in the same direction at equal intervals of 380 meters on a 700-meter diameter loop. In the whole simulation, the first two cars maintain the same slow speed throughout, and the third car overtakes the first two cars twice at a faster speed.

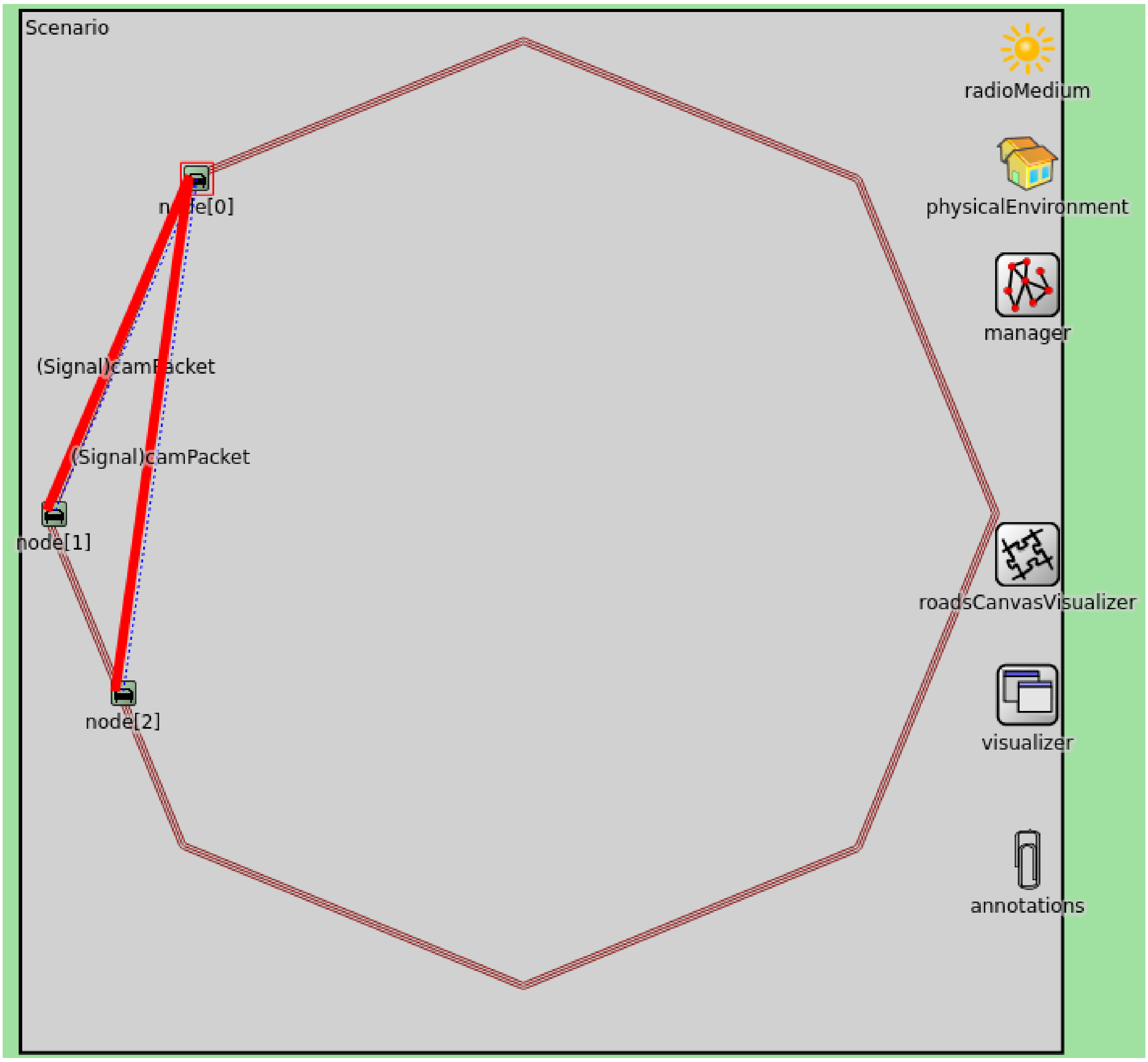

Figure 3.6: Scenario loop

And the corresponding configuration file *omnetpp.ini* given in the appendix D.

In our second scenario, we introduce a part of a real street map of Vandœuvre-lès-Nancy and we simulate avoiding emergency vehicles coming behind.

3.3.2 Performance evaluation of communication trust

For evaluating the performance of our trust model, two possible methods of attack in the envisioned VANET were tested and compared to normal nodes: New Comer Attack and On-Off Attack.

In multi-service Internet of things (IoT) architectures, On-Off attacks are considered as a selective attack type. A malicious device can provide good and bad services randomly to avoid being rated as a low trust node and An On-Off attacker (OA) can also behave differently with different neighbors to achieve inconsistent trust opinions of the same node. However, due to CAM multicast in VANET, an OA can only provide good and bad cooperative awareness basic services randomly.

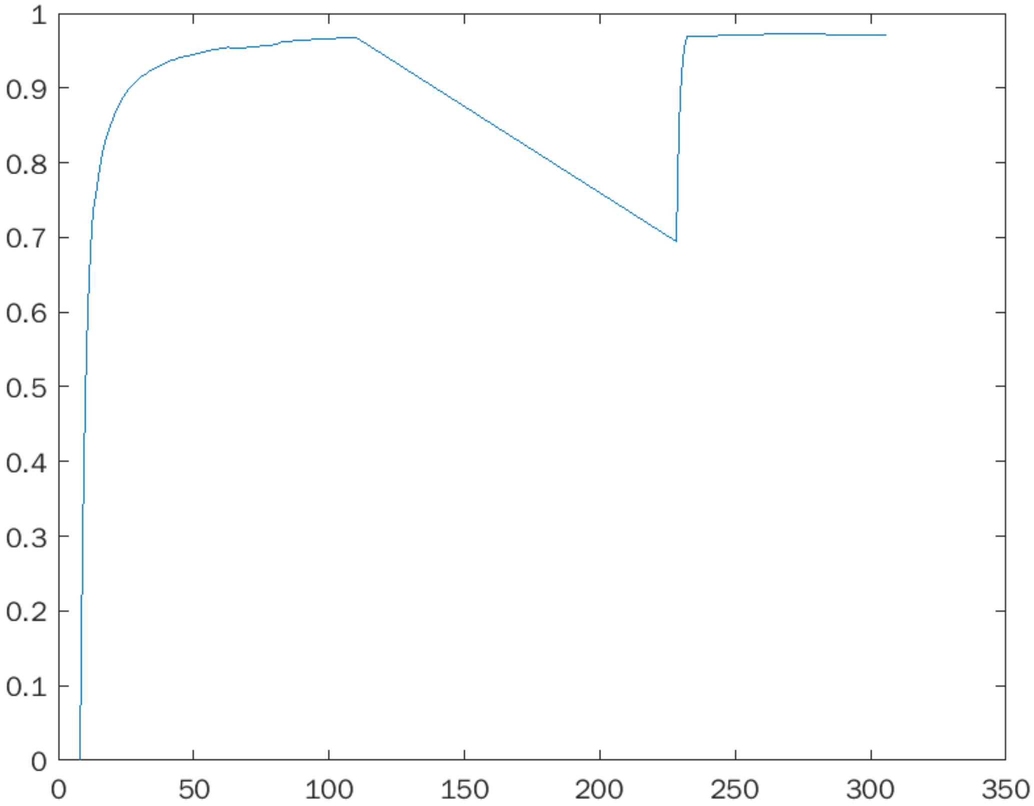

Figure 3.7: Communication trust value from node2 to node1

As Figure 3.6 shows, when the first time node2 approaches node1, it takes about 20 seconds to achieve a 0.9 communication trust value, and it maintains a high level of trust until node2 runs out of communication range. And the second time, its communication trust value increases faster than the first time (about 5 seconds), on contrary, if node1 reappears as a new node, its trust score will not be as high as keeping its original identity.

Then we set node1 in On-Off every 2 seconds mode, i.e., it send 1 CAM in every 2 seconds, and relaunch the simulation.

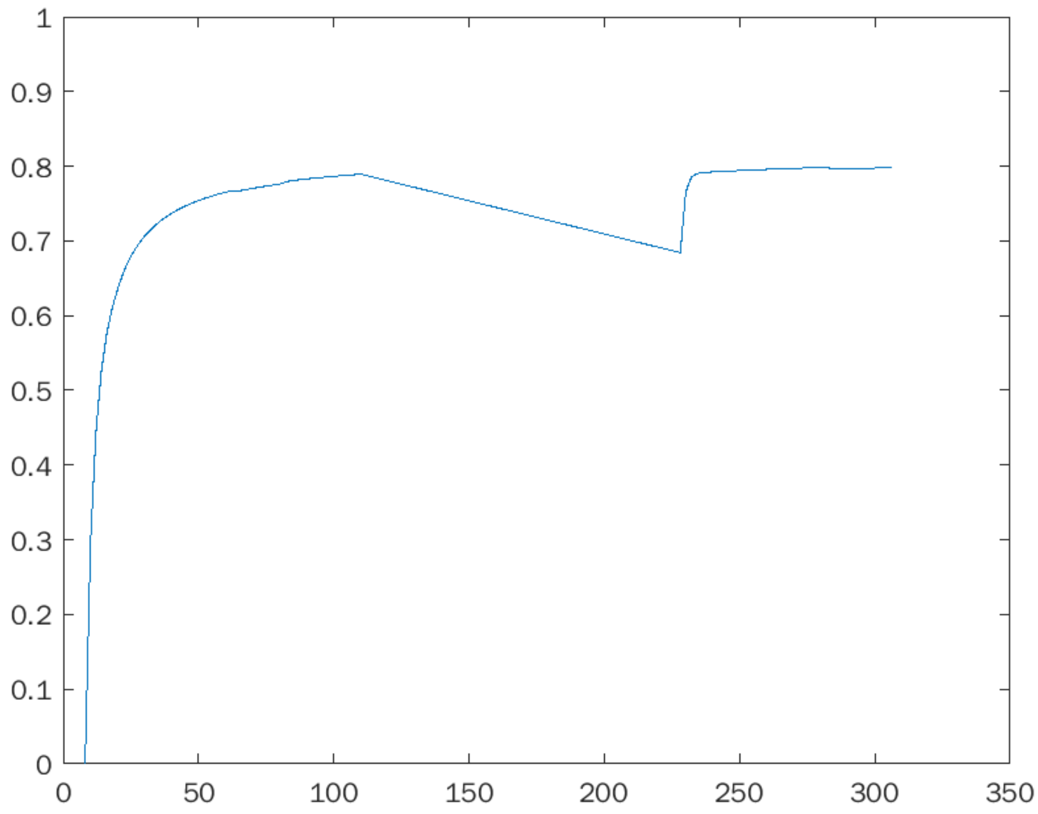

Figure 3.8: Communication trust value from node2 to node1 in node1 On-Off every 2 seconds mode

As Figure 3.7 shows, due to the introduction of the freshness concept, any untimely message delivery will greatly reduce the trustworthiness of the sender and even make it unable to reach the threshold of trust (0.9).

3.3.3 Performance evaluation of behavior trust

Combined with our definition of behavior trust in Section 5.6, the behavior trust curve for overtaking should be a trapezoid concave directly above, like a capital letter M as shows in Figure 3.8.

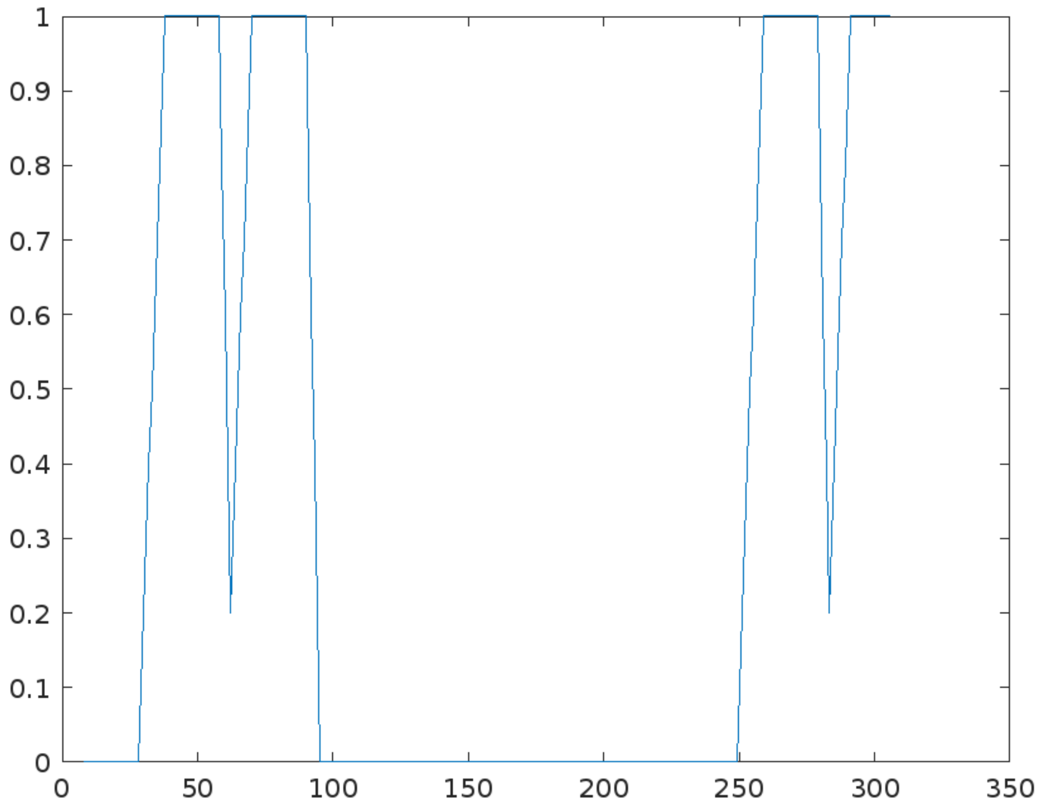

Figure 3.9: Behavior trust value from node2 to node1

3.3.4 Comprehensive evaluation of direct trust

The direct trust value is given as a arithmetic mean, i.e. $T_d = \frac{T_c+T_b}{2}$. Figure 3.9 shows the curve form.

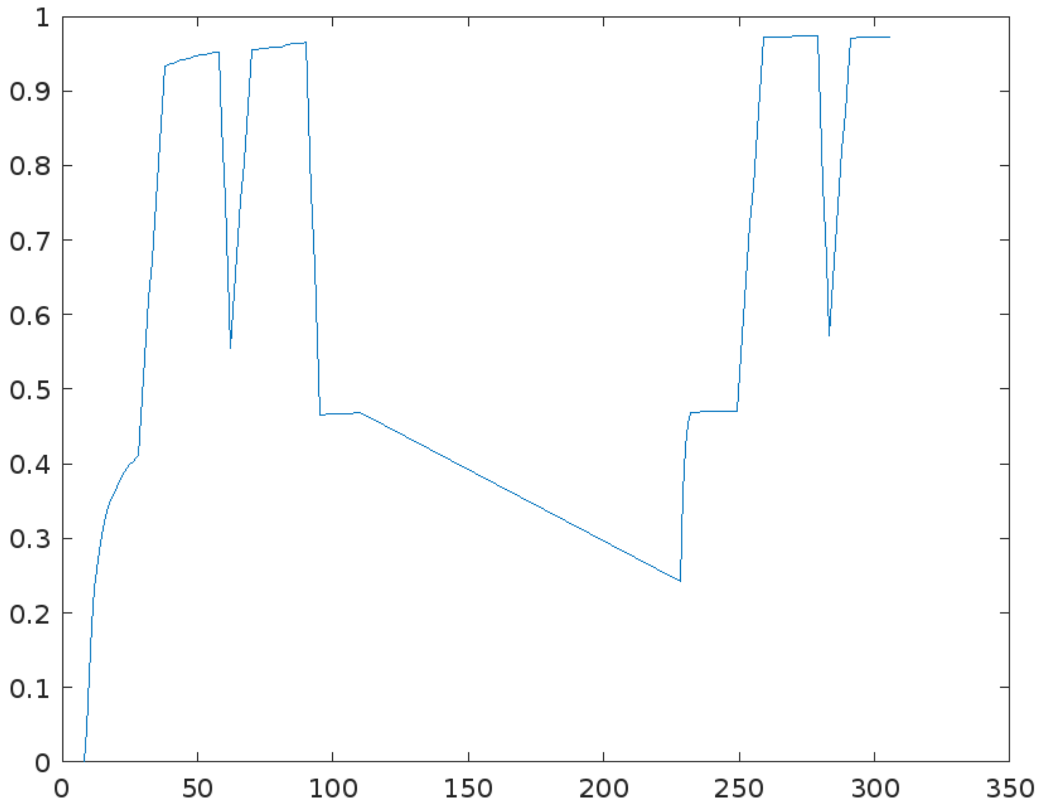

Figure 3.10: Direct trust value from node2 to node1

For the weighted average, "arithmetic mean" or "geometric mean" and even "harmonic mean" are all valuable means. "Arithmetic mean" is often used to report central tendencies, it is significantly influenced by outliers. "geometric mean" is suitable for parameters with different value ranges and can generate a smoother trust curve, but is susceptible to the initial and final values, and when any of the variables is equal to 0 or negative, the geometric mean is meaningless. Therefore, we chose the arithmetic mean to present the direct trust value between vehicles.

Figure 3.9 shows that communication trust determines the basis of direct trust, while behavior trust determines the continuity of trust, but both are indispensable at the same time.

3.3.5 Scenario with specific traffic situation

Next, we simulated an emergency vehicle passing scenario at a specific intersection. To verify the implementation of indirect trust and DENM. The intersection is chosen the Carrefour du Vélodrome in Nancy, France. The emergency vehicles pass from the main road, sending the corresponding DENM to city cars in front, and then the cars in front spread the DENM to make more cars change lanes to avoid the emergency vehicles.

In order to introduce the 3d map, the original version of the geo-environmental data was sourced from OpenStreetMap, and then to turn it into a barrier model that can be used for simulate communication attenuation, the *.xml* format polygon description file needs to be transformed with coordinates, formatting and implemented with a custom Matlab script. please see annexes for details.

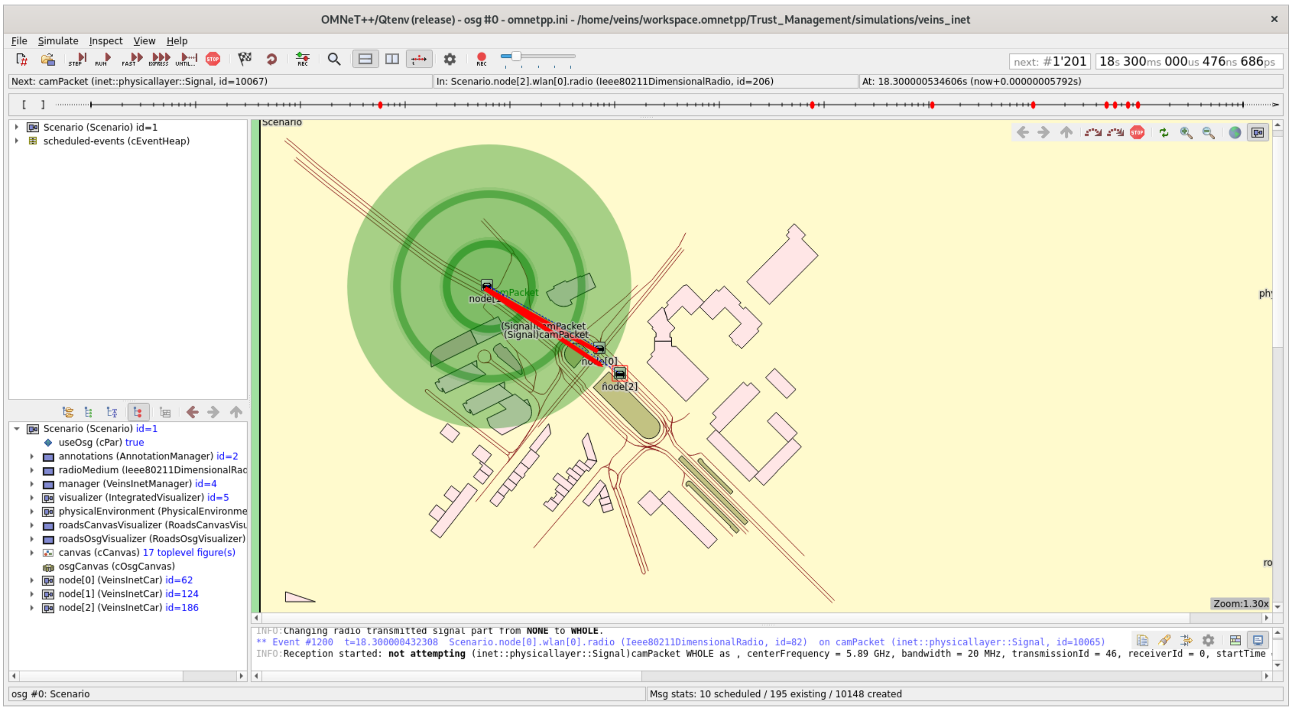

Figure 3.11: Scenario display under 2D visualizer

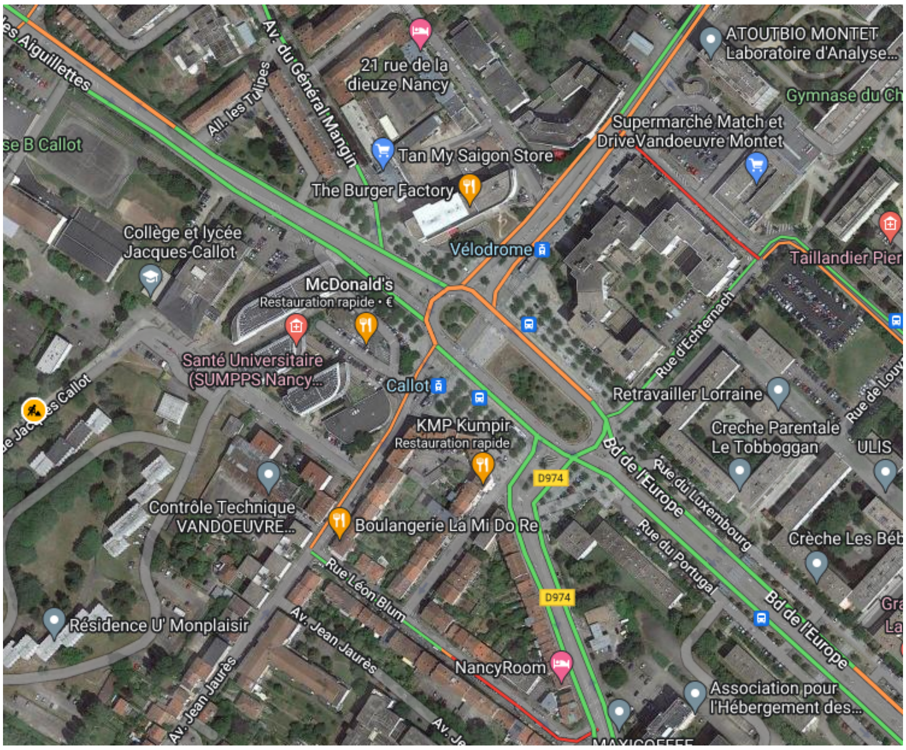

Figure 3.12: Corresponding real scene, source: Google Maps

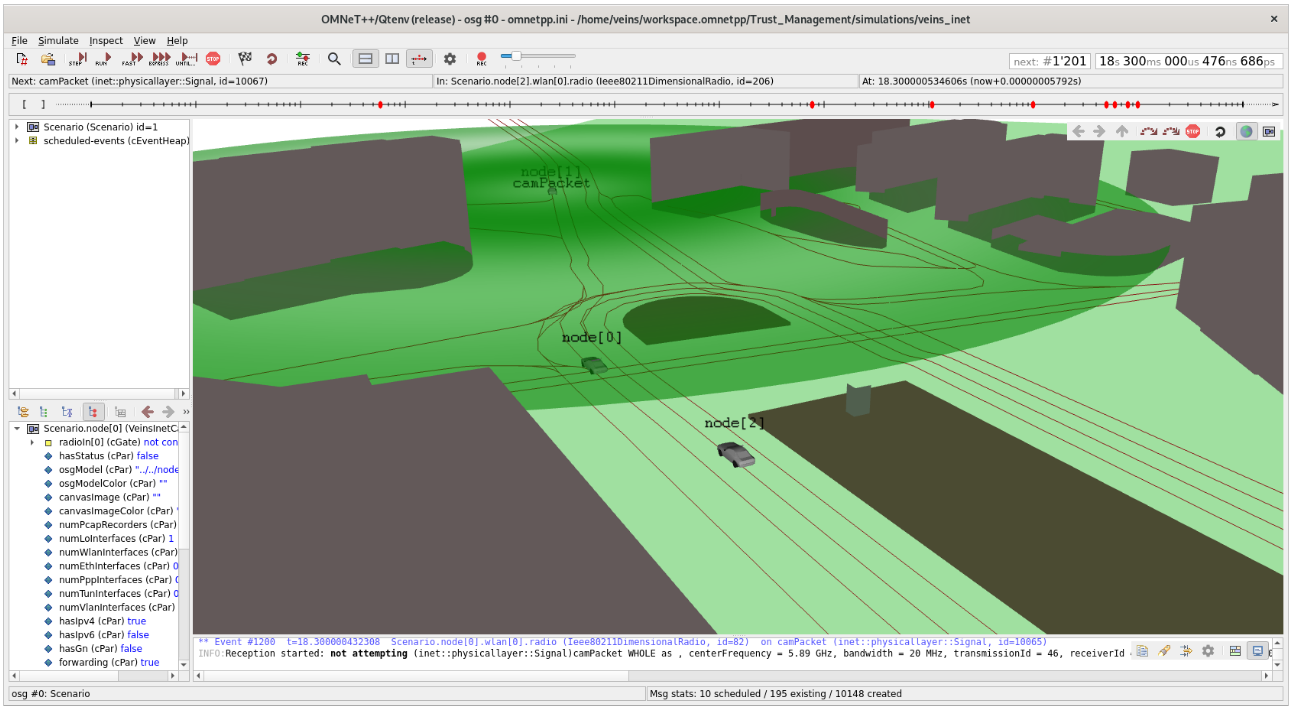

Figure 3.13: Scenario display under 3D visualizer

4 Conclusion and future work

Considering the connections among vehicles, an individual vehicle evaluates the trust of other vehicles, based on the strength of their connections and another vehicle's behavior. Frequent interactions between vehicles and compliant driving behavior usually indicate strong trust relations, while the trust relations among vehicles aid the indirect trust assessment. We propose a new trust model to achieve trust management needs in the ITS scene under CAM and DENM communication. Then we built a vivid and intuitive simulation scenario based on the OMNet++ framework and SUMO road traffic simulation platform to validate our proposed model and adapt various road events.

In simulations, our distributed trust model in VANET is found to be effective and efficient. In future work, we expect to continue to expand our simulator to include more types of events, and to be able to do comparative studies with real acquisition data to improve the simulation.

The direction of future work might be to introduce some dynamic trust measures into the trust model, capturing the dynamics of VANETs by allowing nodes to control trust management based on the situation at hand. For example, two important dynamics in the context of VANET: event/-task and location/time. Furthermore, we are also very interested in the scalability and responsiveness. We would like to do further testing our trust model on larger and faster computing platforms.

Bibliography

[1] Robert Mitchell and Ing-Ray Chen. A survey of intrusion detection techniques for cyber-physical systems. ACM Comput. Surv., 46(4), mar 2014. doi:10.1145/2542049.

[2] Christoph Sommer, Reinhard German, and Falko Dressler. Bidirectionally Coupled Network and Road Traffic Simulation for Improved IVC Analysis. IEEE Transactions on Mobile Computing (TMC), 10(1):3–15, January 2011. doi:10.1109/TMC.2010.133.

[3] Chai K Toh. Ad hoc mobile wireless networks: protocols and systems. Pearson Education, 2001.

[4] Fatih Sakiz and Sevil Sen. A survey of attacks and detection mechanisms on intelligent transportation systems: Vanets and iov. Ad Hoc Networks, 61:33–50, 2017. URL: https://www.sciencedirect.com/science/article/pii/S1570870517300562, doi:https://doi.org/10.1016/j.adhoc.2017.03.006.

[5] Sini Ruohomaa and Lea Kutvonen. Trust management survey. In Peter Herrmann, Valérie Issarny, and Simon Shiu, editors, Trust Management, pages 77–92, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[6] Jie Zhang. A survey on trust management for vanets. In 2011 IEEE International Conference on Advanced Information Networking and Applications, pages 105–112, 2011. doi:10.1109/AINA.2011.86.

[7] Rens Wouter van der Heijden, Stefan Dietzel, Tim Leinmüller, and Frank Kargl. Survey on misbehavior detection in cooperative intelligent transportation systems. IEEE Communications Surveys & Tutorials, 21(1):779–811, 2019. doi:10.1109/COMST.2018.2873088.

[8] Chaker Abdelaziz Kerrache, Nasreddine Lagraa, Carlos T Calafate, Juan-Carlos Cano, and Pietro Manzoni. T-vnets: A novel trust architecture for vehicular networks using the standardized messaging services of etsi its. Computer Communications, 93:68–83, 2016.

[9] José Santa, Fernando Pereñíguez, Antonio Moragón, and Antonio F Skarmeta. Experimental evaluation of cam and denm messaging services in vehicular communications. Transportation Research Part C: Emerging Technologies, 46:98–120, 2014.

[10] Zhongyuan Jiang Zhiquan Liu, Jianfeng Ma, Hui Zhu, and Yinbin Miao. Lsot: A lightweight self-organized trust model in vanets. Mobile Information Systems, page 15, 2016. `doi:10.1155/2016/7628231`.

[11] Avleen Kaur Malhi and Shalini Batra. Fuzzy-based trust prediction for effective coordination in vehicular ad hoc networks. International Journal of Communication Systems, 30(6):e3111, 2017. e3111 dac.3111. URL: `https://onlinelibrary.wiley.com/doi/abs/10.1002/dac.3111`, `arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/dac.3111`, `doi:https://doi.org/10.1002/dac.3111`.

[12] Tong Cheng, Guangchi Liu, Qing Yang, and Jianguo Sun. Trust assessment in vehicular social network based on three-valued subjective logic. IEEE Transactions on Multimedia, 21(3):652–663, 2019. `doi:10.1109/TMM.2019.2891417`.

[13] Fangyu Gai, Jiexin Zhang, Peidong Zhu, and Xinwen Jiang. Trust on the ratee: a trust management system for social internet of vehicles. Wireless Communications and Mobile Computing, 2017, 2017.

[14] Wikipedia contributors. Two-second rule — Wikipedia, the free encyclopedia, 2022. `https://en.wikipedia.org/w/index.php?title=Two-second_rule&oldid=1089170388`, Online accessed 5-August-2022.

[15] Tamás Ormándi. Practical manual of sumo/matlab/veins/inet/omnet++ programming and interfacing for v2x simulation with standard protocols. Technical report, Budapest University of Technology and Economics, November 2021.

[16] Lelio Campanile, Marco Gribaudo, Mauro Iacono, Fiammetta Marulli, and Michele Mastroianni. Computer network simulation with ns-3: A systematic literature review. Electronics, 9(2):272, 2020.

[17] Wikipédia. Network simulator — wikipédia, l'encyclopédie libre, 2020. [En ligne; Page disponible le 16-septembre-2020]. URL: `http://fr.wikipedia.org/w/index.php?title=Network_Simulator&oldid=174772816`.

[18] Avani Sharma, Emmanuel S Pilli, Arka P Mazumdar, and Poonam Gera. Towards trustworthy internet of things: A survey on trust management applications and schemes. Computer Communications, 2020.

[19] Rasheed Hussain, Jooyoung Lee, and Sherali Zeadally. Trust in vanet: A survey of current solutions and future research opportunities. IEEE transactions on intelligent transportation systems, 22(5):2553–2571, 2020.

[20] Hamssa Hasrouny, Abed Ellatif Samhat, Carole Bassil, and Anis Laouiti. Vanet security challenges and solutions: A survey. Vehicular Communications, 7:7–20, 2017.

[21] Weiwei Liu, Xichen Wang, Wenli Zhang, Lin Yang, and Chao Peng. Coordinative simulation with sumo and ns3 for vehicular ad hoc networks. In 2016 22nd Asia-Pacific Conference on Communications (APCC), pages 337–341, 2016. `doi:10.1109/APCC.2016.7581471`.

[22] Jie Zhang, Chen Chen, and Robin Cohen. Trust modeling for message relay control and local action decision making in vanets. Security and Communication Networks, 6(1):1–14, 2013.

[23] Chaker Abdelaziz Kerrache, Carlos T. Calafate, Juan-Carlos Cano, Nasreddine Lagraa, and Pietro Manzoni. Trust management for vehicular networks: An adversary-oriented overview. IEEE Access, 4:9293–9307, 2016. `doi:10.1109/ACCESS.2016.2645452`.

[24] Mohamed Nidhal Mejri, Jalel Ben-Othman, and Mohamed Hamdi. Survey on vanet security challenges and possible cryptographic solutions. Vehicular Communications, 1(2):53–66, 2014. URL: `https://www.sciencedirect.com/science/article/pii/S2214209614000187`, `doi:https://doi.org/10.1016/j.vehcom.2014.05.001`.

[25] En 319 411-1 - v1.3.1 - electronic signatures and infrastructures (esi); policy and security requirements for trust service providers issuing certificates; part 1: General requirements.

AnnexesAppendix

A
transform_obstacles.m

```
clear all
S = xmlread("Carr_du_Velodrome.obstacles.xml");
objects = S.getElementsByTagName("object");

docNode = com.mathworks.xml.XMLUtils.createDocument('shapes');
docRootNode = docNode.getDocumentElement;

offset_x = 25;
offset_y = offset_x;
for k = 0:objects.getLength-1
    shape = objects.item(k).getAttribute("shape");
    newStr = split(shape,' ');
    prism_hight = newStr(1:2);
    newStr(1:2) = [];
    newStr = str2num(newStr);
    newshape = reshape(newStr,2,[]);
    newshape=flip(newshape,2);
    x = newshape(1,:);
    y = newshape(2,:);
    x = x + offset_x;
    y = 464.98 - y + offset_y;
    newshape = [x;y];
    newshape = reshape(newshape,1,[]);
    newshape = num2str(newshape,'%10.6f ');
```

```
    newshape = string(prism_hight(1))+" "+string(prism_hight(2))+" "+
        newshape;
    objects.item(k).setAttribute("shape",newshape);

    position = objects.item(k).getAttribute("position");
    newStr = split(position,' ');
    newStr(1) = [];
    newStr = str2num(newStr);
    newStr(1) = min(x);
    newStr(2) = min(y);

    newposition = "min " + string(newStr(1)) + " " + string(newStr(2)) + " "
         + string(newStr(3));
    objects.item(k).setAttribute("position",newposition);

    thisElement = docNode.createElement('object');
    thisElement.setAttribute("position",objects.item(k).getAttribute("
        position"));
    thisElement.setAttribute("orientation",objects.item(k).getAttribute("
        orientation"));
    thisElement.setAttribute("material",objects.item(k).getAttribute("
        material"));
    thisElement.setAttribute("line-color",objects.item(k).getAttribute("line
        -color"));
    thisElement.setAttribute("fill-color",objects.item(k).getAttribute("fill
        -color"));
    thisElement.setAttribute("shape",objects.item(k).getAttribute("shape"));
    docRootNode.appendChild(thisElement);
end

xmlwrite('Carr_du_Velodrome.obs.xml',docNode);
```

B

VehicleApplication.h

```
/*
 * VehicleApplication.h
 *
 *  Created on: Mar 22, 2022
 *      Author: Yujun JIN
 */
#pragma once

#include "veins_inet/veins_inet.h"
#include "veins_inet/VeinsInetApplicationBase.h"
#include "veins_inet/CooperativeAwarenessMessage_m.h"
#include <fstream>

class VEINS_INET_API VehicleApplication : public veins::
    VeinsInetApplicationBase {

protected:
    virtual bool startApplication() override;
    virtual bool stopApplication() override;
    virtual void processPacket(std::shared_ptr<inet::Packet> pk) override;

public:
    VehicleApplication();
    ~VehicleApplication();
```

```
    void generateMessage();
    void generateCAM();   //function for CAM generation and sending
    void generateDENM(long Application_Request_Type);
    void printCAMInfo(struct camInfo camInfo);

    double generateTrustDirect(struct camInfo camInfo);
    double generateBehaviorTrust(struct camInfo camInfo);

    std::pair<double,double> getVehicleAtLong();   //function for getting
        SUMO vehicle GeoPosition
    std::ofstream fout;
    uint8_t msg_buffer[1024];   //buffer for CAM
    uint64_t generationDeltaTime;
    std::list<camInfo> vehicleInfoStack;   // linked list structure
    void getGenerationDeltaTime();
};
```

C

VehicleApplication.cc

```
/*
 * VehicleApplication.cc
 *
 *  Created on: Mar 22, 2022
 *      Author: Yujun JIN
 */


#include "veins_inet/VehicleApplication.h"

#include "inet/common/ModuleAccess.h"
#include "inet/common/packet/Packet.h"
#include "inet/common/packet/PacketFilter.h"
#include "veins_inet/CooperativeAwarenessMessage_m.h"
#include "veins_inet/DecentralizedEnvironmentalNotificationMessage_m.h"
#include <math.h>
#include <string.h>
// Time function dependencies
#include <chrono>
#include "/home/veins/workspace.omnetpp/Trust_Management/asn1c/date.h"
#include <cstdint>

// Clock structure for generationDeltaTime
struct myclock{
```

```
    using rep    = std::int32_t;
    using period= std::milli;
    using duration  = std::chrono::duration<rep, period>;
    using time_point = std::chrono::time_point<myclock>;
    static constexpr bool is_steady = false;

    static time_point now() noexcept
    {
        using namespace std::chrono;
        using namespace date;
        return time_point {duration_cast<duration>(system_clock::now()-
            sys_days{jan/1/2004})};
    }
};

// CAM structure for store station informations
struct pathPoint{
    long     pathDeltaTime;
    std::pair<long,long> position;
};
struct camInfo{
    long     stationID;
    long     protocolVersion;
    // GenDeltaTime
    long     generationDeltaTime; // TimestampIts mod 65536
    // cam Parameters
    // BasicContainer
    long     stationType;
    // ReferencePosition
    long     lastLatitude;
    long     lastLongitude;
    long     altitude;
    // HighFrequencyContainer
```

```
    long     heading;
    long     speed;
    long     driveDirection;
    long     vehicleLength;
    long     vehicleWidth;
    long     longitudinalAcceleration;
    long     yawRate;
    double   behaviorTrust;
    double   directTrust;
    std::deque<pathPoint> pathHistory;
};
using namespace inet;

Define_Module(VehicleApplication);

VehicleApplication::VehicleApplication()
{
}

bool VehicleApplication::startApplication()
{
    generateMessage();
    return true;
}

bool VehicleApplication::stopApplication()
{
    return true;
}

VehicleApplication::~VehicleApplication()
{
}
```

```
void VehicleApplication::processPacket(std::shared_ptr<inet::Packet> pk)
{
    EV_INFO << "------------------ProcessPacket: " << pk->getName() << "
        ------------------\n";
    if (strstr(pk->getName(),"camP")) {
        auto payload = pk->peekAtFront<CooperativeAwarenessMessage>();
        EV_INFO << "node" << getParentModule()->getIndex() << " receive
            packet from " << payload->getStationID() << "\n";
        getParentModule()->getDisplayString().setTagArg("i", 1, "green");

        if (getParentModule()->getIndex()!= payload->getStationID()){
            camInfo tInfo;
            tInfo.stationID = payload->getStationID();
            tInfo.protocolVersion = payload->getProtocolVersion();
            tInfo.generationDeltaTime = payload->getGenerationDeltaTime();
                // TimestampIts mod 65536
            tInfo.stationType = payload->getStationType();
            tInfo.lastLatitude = payload->getLatitude();
            tInfo.lastLongitude = payload->getLongitude();
            tInfo.altitude = payload->getAltitude();
            tInfo.heading = payload->getHeading();
            tInfo.speed = payload->getSpeed();
            tInfo.driveDirection = payload->getDriveDirection();
            tInfo.vehicleLength = payload->getVehicleLength();
            tInfo.vehicleWidth = payload->getVehicleWidth();
            tInfo.longitudinalAcceleration = payload->
                getLongitudinalAcceleration();
            tInfo.yawRate = payload->getYawRate();
            //tInfo.behaviorTrust = 0.0;
            pathPoint pp;
            pp.pathDeltaTime = tInfo.generationDeltaTime;
            pp.position.first = payload->getLatitude();
```

```
            pp.position.second = payload->getLongitude();
            //tInfo.pathHistory.push_front(pp);

            if (!vehicleInfoStack.empty()){

                std::list<camInfo>::iterator iter;
                for (iter = vehicleInfoStack.begin();iter !=
                    vehicleInfoStack.end();iter++) {

                    if (iter->stationID == payload->getStationID()) {
                        tInfo.pathHistory = iter->pathHistory;
                        tInfo.pathHistory.push_front(pp);
                        if (tInfo.pathHistory.size()>10) {
                            tInfo.pathHistory.resize(10);
                        }

                        tInfo.behaviorTrust = iter->behaviorTrust;
                        tInfo.behaviorTrust = generateBehaviorTrust(tInfo);
                        tInfo.directTrust = generateTrustDirect(tInfo);
                        vehicleInfoStack.erase(iter);
                        vehicleInfoStack.push_back(tInfo);

                        std::string filenametemp = "Node";
                        filenametemp += std::to_string(tInfo.stationID);
                        filenametemp += "ScoreFromNode";
                        filenametemp += std::to_string(getParentModule()->
                            getIndex());

                        char name[30];
                        char* filename = std::strcpy(name,filenametemp.c_str
                            ());
                        fout.open(filename, std::ios_base::app);
                        fout.precision(10);
```

```
                        fout << tInfo.generationDeltaTime << " " << tInfo.
                            directTrust << "\n";
                        //fout.flush();
                        fout.close();
                        break;
                    }
                }
                if(iter == vehicleInfoStack.end()) {
                    tInfo.behaviorTrust = generateBehaviorTrust(tInfo);
                    tInfo.directTrust = generateTrustDirect(tInfo);
                    vehicleInfoStack.push_back(tInfo);
                    std::string filenametemp = "Node";
                    filenametemp += std::to_string(tInfo.stationID);
                    filenametemp += "ScoreFromNode";
                    filenametemp += std::to_string(getParentModule()->
                        getIndex());

                    char name[30];
                    char* filename = std::strcpy(name,filenametemp.c_str());
                    fout.open(filename,std::ios_base::app);
                    fout.precision(10);
                    fout << tInfo.generationDeltaTime << " " << tInfo.
                        directTrust << "\n";
                    //fout.flush();
                    fout.close();
                }
            }
            else {
                tInfo.behaviorTrust = generateBehaviorTrust(tInfo);
                tInfo.directTrust = generateTrustDirect(tInfo);
                vehicleInfoStack.push_back(tInfo);
                std::string filenametemp = "Node";
                filenametemp += std::to_string(tInfo.stationID);
```

```
                    filenametemp += "ScoreFromNode";
                    filenametemp += std::to_string(getParentModule()->getIndex()
                        );

                    char name[30];
                    char* filename = std::strcpy(name,filenametemp.c_str());
                    fout.open(filename,std::ios_base::app);
                    fout.precision(10);
                    fout << tInfo.generationDeltaTime << " " << tInfo.
                        directTrust << "\n";
                    //fout.flush();
                    fout.close();
                }
            }
        }
        else if (strstr(pk->getName(),"denm")){
            EV_INFO << "DENM received. \n";
            auto payload = pk->peekAtFront<
                DecentralizedEnvironmentalNotificationMessage>();
            getParentModule()->getDisplayString().setTagArg("i", 1, "green");

            if (getParentModule()->getIndex()!= payload->getStationID()){
                EV_INFO << "vehicle road ID" << traciVehicle->getRoadId().c_str
                    () << "\n";
                EV_INFO << "payload road ID" << payload->getRoadId() << "\n";
                if (!strcmp(traciVehicle->getRoadId().c_str(),payload->getRoadId
                    ())) {
                    if (!strcmp(traciVehicle->getLaneId().c_str(),payload->
                        getLaneId())) {
                         //uint8_t lane = payload->getLaneId().end();
                         traciVehicle->changeLane(0x01, 1);
                    }
                }
```

```
        }
    }
}

void VehicleApplication::printCAMInfo(struct camInfo camInfo){
    EV_INFO << "------------------Print CAM Info-------------------\n";
    EV_INFO << "stationID  = " << camInfo.stationID << "\n";
    EV_INFO << "protocolVersion = " << camInfo.protocolVersion << "\n";
    EV_INFO << "generationDeltaTime = " << camInfo.generationDeltaTime << "\n";
    EV_INFO << "stationType     = " << camInfo.stationType << "\n";
    EV_INFO << "ReferencePosition\n";
    EV_INFO << " longitude     = " << camInfo.lastLatitude << "\n";
    EV_INFO << " latitude      = " << camInfo.lastLongitude << "\n";
    EV_INFO << " altitude      = " << camInfo.altitude << "\n";
    EV_INFO << "HighFrequencyContainer\n";
    EV_INFO << " heading            = " << camInfo.heading << "\n";
    EV_INFO << " speed              = " << camInfo.speed << " unit 0.01m/s\n";
    EV_INFO << " driveDirection     = " << camInfo.driveDirection << "\n";
    EV_INFO << " vehicleLength      = " << camInfo.vehicleLength << " unit mm\n";
    EV_INFO << " vehicleWidth       = " << camInfo.vehicleWidth << " unit mm\n";
    EV_INFO << " longitudinalAcceleration = " << camInfo.longitudinalAcceleration << " unit 0.1m/s^2\n";
    EV_INFO << " yawRate            = " << camInfo.yawRate << "\n";
    EV_INFO << "PathHistory\n";
//      if (camInfo.pathHistory.empty()){
//          EV_INFO << "No path history available\n";
//      }
//      else {
//          for (int  i=0;i<camInfo.pathHistory.size();i++){
//              EV_INFO << "PathDeltaTime: " << camInfo.pathHistory[i].
```

```
        pathDeltaTime << " Position: " << camInfo.pathHistory[i].position.
        first << ", " << camInfo.pathHistory[i].position.second << "\n";
    //          }
    //      }
}

double VehicleApplication::generateTrustDirect(struct camInfo camInfo){
    double td = 0.0;
    double ct = 0.0;   // Communication Trust
    double bt = 0.0;   // Behavior Trust
    double st = 0.0;   // Social level Trust
    double a = 0.5;    // Time Decay Assessment Weight
    double b = 0.5;    // Number of Communication Assessment Weight
    double rho = 0.5; // Attenuation factor for 1 second
    double lambda = 5.0;
    double tau = 1.0; // Time Constant
    double Tn,Tk,p1,p2;
    /*---------------- Communication Trust ----------------*/
    p1 = 0; //part of time decay
    p2 = 0; //part of number of Communication
    Tn = (long)(simTime().dbl()*100);

    for (int i=0; i<(camInfo.pathHistory.size()>5 ? 5 : camInfo.pathHistory.
        size()); i++){
        Tk = camInfo.pathHistory[i].pathDeltaTime;
        p1 = p1 + pow(rho,(Tn-Tk)*tau/100);
    }
    p1 = p1*(1-pow(rho,tau));

    if (camInfo.pathHistory.size()==1) {
        p2 = 0;
    }
    else {
```

```
        p2 = pow(rho,(lambda/camInfo.pathHistory.size()));
    }

    ct = sqrt(p1*p2); // a+b=1 if ct=a*p1+b*p2
    /*---------------- Behavior Trust ----------------*/
    bt = camInfo.behaviorTrust;
    /*---------------- Social level Trust ----------------*/
    st = 1.0;
    /*---------------- Direct Trust ----------------*/
    td = (ct+bt)/2;

    return td;
}

double VehicleApplication::generateBehaviorTrust(struct camInfo camInfo){
    double bt = 0.0;  // Behavior Trust
    double p1,p2,d;

    p1 = this->getVehicleAtLong().first - camInfo.lastLongitude;
    p2 = this->getVehicleAtLong().second - camInfo.lastLatitude;
    //    EV_INFO << " node position " << this->getVehicleAtLong().first <<
        "," << this->getVehicleAtLong().second << "\n";
    //    EV_INFO << " refe position " << camInfo.lastLongitude << "," <<
        camInfo.lastLatitude << "\n";
    d = sqrt(pow(p1/1000,2)+pow(p2/1000,2));
    EV_INFO << " distance = " << d << "\n\n";
    if (d>traciVehicle->getSpeed()*2 && d<400){
        if (camInfo.behaviorTrust<0.9){
            bt = camInfo.behaviorTrust + 0.1;
        }
        else {
            bt = camInfo.behaviorTrust;
        }
```

```
    }
    else {
        if (camInfo.behaviorTrust>0.2){
            bt = camInfo.behaviorTrust - 0.2;
        }
    }
    return bt;
}

void VehicleApplication::generateMessage()
{
    // Send CAM message from node[0], other nodes will forward the message
    int tStart;
//      tStart = getParentModule()->getIndex()*200+200;   // unit ms
    tStart = getParentModule()->getIndex()*5200+200;   // unit ms

    auto start = [this]() {
        auto callback = [this]() {
            if (getParentModule()->getIndex()==1 && simTime().operator >(
                SimTime(6,SIMTIME_S)) && simTime().operator <=(SimTime(7,
                SIMTIME_S))) {
                generateDENM(1);
                traciVehicle->setSpeed(24.0);
            }
            else {
                generateCAM();
            }
        };
        timerManager.create(veins::TimerSpecification(callback).interval(
            SimTime(1, SIMTIME_S)));
    };
    timerManager.create(veins::TimerSpecification(start).oneshotAt(SimTime(
        tStart, SIMTIME_MS)));
```

```
}

// Function to fill CAM data for every node and encode it in UPER then send
    it in every second.
void VehicleApplication::generateCAM()
{
    // Set node icon to red, indicating CAM generation
    getParentModule()->getDisplayString().setTagArg("i", 1, "red");
    /* CAM Structure */

    /* Get vehicle parameters */
    std::cout << "node" << getParentModule()->getIndex() << " start CAM
        generation...\n";

    // Creating the payload, witch is a shared pointer
    auto payload = makeShared<CooperativeAwarenessMessage>();

    // Defining payload Chunk length
    payload->setChunkLength(B(1024));

    /* CAM STRUCTURE */
    // HEADER
    payload->setProtocolVersion(000001);
    payload->setMessageID(1);
    payload->setStationID(getParentModule()->getIndex());

    // GenerationDeltaTime
    this->getGenerationDeltaTime();
    payload->setGenerationDeltaTime(generationDeltaTime);

    // BasicContainer
    payload->setStationType(1);

```

```
    // ReferencePosition
    payload->setLongitude(this->getVehicleAtLong().first);
    payload->setLatitude(this->getVehicleAtLong().second);
    payload->setAltitude(0);

    // HFContainer
    payload->setHeading(traciVehicle->getAngle()*1000);
    payload->setSpeed(traciVehicle->getSpeed()*100); // unit 0.01m/s
    payload->setDriveDirection(0); // forward (0), backward (1), unavailable
        (2)
    payload->setVehicleLength(traciVehicle->getLength()*1000); // unit 1
        millimeter
    payload->setVehicleWidth(traciVehicle->getWidth()*1000); // unit 1
        millimeter
    payload->setLongitudinalAcceleration(traciVehicle->getAcceleration()*10)
        ; // unit 0.1m/s^2
    payload->setYawRate(0); // Temporarily unavailable

    // Timestamp adds the newest generation time to the payload
    // (or if it already had one, it removes the old one)
    timestampPayload(payload);

    // Creating the packet
    auto packet = createPacket("camPacket");

    // Insert the payload at the end pointer of the packet
    packet->insertAtBack(payload);

    sendPacket(std::move(packet));
}

// Function to fill DENM data for every node and encode it in UPER then send
    it in every second.
```

```
void VehicleApplication::generateDENM(long Application_Request_Type)
{
    // Set node icon to blue, indicating DENM generation
    getParentModule()->getDisplayString().setTagArg("i", 1, "blue");
    /* DENM Structure */

    /* Get vehicle parameters */
    std::cout << "node" << getParentModule()->getIndex() << " start DENM
        generation...\n";

    // Creating the payload, witch is a shared pointer
    auto payload = makeShared<DecentralizedEnvironmentalNotificationMessage
        >();

    // Defining payload Chunk length
    payload->setChunkLength(B(1024));
    payload->setRoadId(traciVehicle->getRoadId().c_str());
    payload->setLaneId(traciVehicle->getLaneId().c_str());
    /* DENM STRUCTURE */
    // HEADER
    payload->setProtocolVersion(000001);
    payload->setMessageID(1);
    payload->setStationID(getParentModule()->getIndex());

    switch (Application_Request_Type)
    {
    case 1: // AppDENM_trigger
        // Detection Time
        this->getGenerationDeltaTime();
        payload->setDetectionTime(generationDeltaTime);
        // Event Position
        payload->setEventPositionLong(this->getVehicleAtLong().first);
        payload->setEventPositionLat(this->getVehicleAtLong().second);
```

```
            // Event Validity Duration   *optional
            // Repetition Duration       *optional
            // Transmission Interval     *optional
            // Repetition Interval       *optional
            // Situation Container       *optional
            // Location Container        *optional
            // Alacarte Container        *optional
            // Relevance Area            *optional
            // Destination area
            // Trafic class
        case 2: // AppDENM_update
            payload->setActionID(Application_Request_Type);
            // Detection Time
            this->getGenerationDeltaTime();
            payload->setDetectionTime(generationDeltaTime);
            // Event Position
            payload->setEventPositionLong(this->getVehicleAtLong().first);
            payload->setEventPositionLat(this->getVehicleAtLong().second);
            // Event Validity Duration   *optional
            // Repetition Duration       *optional
            // Transmission Interval     *optional
            // Repetition Interval       *optional
            // Situation Container       *optional
            // Location Container        *optional
            // Alacarte Container        *optional
            // Relevance Area            *optional
            // Destination area
            // Trafic class
        //case 3: // AppDENM_termination

        //default:

        }
```

```

    // ManagementContainer
    // ActionID = original StationID + sequence number  ex: 1000 + 1
    payload->setActionID(getParentModule()->getIndex()*1000);

    // ReferenceTime

/*
    // SituationContainer
    payload->setEventType(EventType);

    // ReferencePosition
    payload->setLongitude(this->getVehicleAtLong().first);
    payload->setLatitude(this->getVehicleAtLong().second);
    payload->setAltitude(0);

    // HFContainer
    payload->setHeading(traciVehicle->getAngle()*1000);
    payload->setSpeed(traciVehicle->getSpeed()*100); // unit 0.01m/s
    payload->setDriveDirection(0); // forward (0), backward (1), unavailable
        (2)
    payload->setVehicleLength(traciVehicle->getLength()*1000); // unit 1
        millimeter
    payload->setVehicleWidth(traciVehicle->getWidth()*1000); // unit 1
        millimeter
    payload->setLongitudinalAcceleration(traciVehicle->getAcceleration()*10)
        ; // unit 0.1m/s^2
    payload->setYawRate(0); // Temporarily unavailable
*/
    // Timestamp adds the newest generation time to the payload
    // (or if it already had one, it removes the old one)
    timestampPayload(payload);

```

```
    // Creating the packet
    auto packet = createPacket("denmPacket");

    // Insert the payload at the end pointer of the packet
    packet->insertAtBack(payload);

    sendPacket(std::move(packet));
}

/*
 * This function gets the current position of the node from OMNET++ in an
    inet::Coord type.
 * It transforms the inet::Coord type to the veins::Coord type.
 * Then it returns the Latitude and Longitude in an std::pair<double,double>
    type.
 * Longitude: pair.first
 * Latitude: pair.second
 */
std::pair<double,double> VehicleApplication::getVehicleAtLong()
{
    // Get the vehicle coordinates from OMNET++ stored in an INET Coord
    inet::Coord vehiclePosition = mobility->getCurrentPosition();
    // Veins Coord
    veins::Coord veinsCoord;
    // Veins Coord Longitude
    veinsCoord.x = vehiclePosition.x;
    // Veins Coord Latitude
    veinsCoord.y = vehiclePosition.y;
    std::pair<double,double> CoordPair = traci->getLonLat(veinsCoord);
    // Multiply values to get the correct form CAM
    CoordPair.first *= 1000.0;
    CoordPair.second *= 1000.0;
    // Return the Coordinate Pair
```

```
    return CoordPair;
}

// Function to generate the generationDeltaTime for the CAM message
void VehicleApplication::getGenerationDeltaTime(){
    //      auto tp = myclock::now();
    //      generationDeltaTime = tp.time_since_epoch().count();
    //      // Usage by standard:
    //      generationDeltaTime = generationDeltaTime % 65536;
    // TODO: Use SimTime?
    generationDeltaTime = (long)(simTime().dbl()*100);
}
```

D

omnetpp.ini

```
[General]
network = Scenario
sim-time-limit = 288s
debug-on-errors = true
cmdenv-express-mode = true
image-path = ../../../../images

# UDPBasicApp
*.node[*].numApps = 1
*.node[*].app[*].typename = "trust_management.veins_inet.VehicleApplication"
*.node[*].app[*].interface = "wlan0"

# Ieee80211Interface
*.node[*].wlan[*].opMode = "p"
*.node[*].wlan[*].radio.typename = "Ieee80211DimensionalRadio"
*.node[*].wlan[*].radio.bandName = "5.9 GHz"
*.node[*].wlan[*].radio.channelNumber = 3
*.node[*].wlan[*].radio.transmitter.power = 80mW
*.node[*].wlan[*].radio.bandwidth = 10 MHz
*.node[*].wlan[*].radio.antenna.mobility.typename = "AttachedMobility"
*.node[*].wlan[*].radio.antenna.mobility.mobilityModule = "^.^.^.^.mobility"
*.node[*].wlan[*].radio.antenna.mobility.offsetX = -2.5m
*.node[*].wlan[*].radio.antenna.mobility.offsetZ = 1.5m
*.node[*].wlan[*].radio.antenna.mobility.constraintAreaMinX = 0m
```

```
*.node[*].wlan[*].radio.antenna.mobility.constraintAreaMaxX = 0m
*.node[*].wlan[*].radio.antenna.mobility.constraintAreaMinY = 0m
*.node[*].wlan[*].radio.antenna.mobility.constraintAreaMaxY = 0m
*.node[*].wlan[*].radio.antenna.mobility.constraintAreaMinZ = 0m
*.node[*].wlan[*].radio.antenna.mobility.constraintAreaMaxZ = 0m

# HostAutoConfigurator
*.node[*].ipv4.configurator.typename = "HostAutoConfigurator"
*.node[*].ipv4.configurator.interfaces = "wlan0"
*.node[*].ipv4.configurator.mcastGroups = "224.0.0.1"

# VeinsInetMobility
*.node[*].mobility.typename = "VeinsInetMobility"

# VeinsInetManager
*.manager.updateInterval = 0.1s
*.manager.host = "localhost"
*.manager.port = 9999
*.manager.autoShutdown = true
*.manager.launchConfig = xmldoc("loop.launchd.xml")
*.manager.moduleType = "trust_management.veins_inet.VeinsInetCar"

# PhysicalEnvironment
*.physicalEnvironment.config = xmldoc("obstacles.xml")
*.radioMedium.obstacleLoss.typename = "IdealObstacleLoss"

# Misc
**.vector-recording = true

[Config plain]

[Config canvas]
extends = plain
```

```
description = "Enable enhanced 2D visualization"

# IntegratedCanvasVisualizer (2D)
*.visualizer.*.obstacleLossVisualizer.displayIntersections = true
*.visualizer.*.obstacleLossVisualizer.displayFaceNormalVectors = true
*.visualizer.*.obstacleLossVisualizer.intersectionLineColor = "yellow"
*.visualizer.*.mediumVisualizer.signalPropagationAnimationSpeed = 500/3e8
*.visualizer.*.mediumVisualizer.signalTransmissionAnimationSpeed = 50000/3e8
*.visualizer.*.mediumVisualizer.displaySignals = true
*.visualizer.canvasVisualizer.mediumVisualizer.displaySignalDepartures =
    false
*.visualizer.canvasVisualizer.mediumVisualizer.displaySignalArrivals = false
*.visualizer.*.physicalLinkVisualizer.displayLinks = true
*.visualizer.osgVisualizer.typename = ""

[Config osg]
extends = canvas
description = "Enable enhanced 2D and 3D visualization using OSG"

*.useOsg = true

# IntegratedOsgVisualizer (3D)
*.visualizer.osgVisualizer.typename = IntegratedOsgVisualizer
*.node[*].osgModel = "veins_inet/node/car.obj.-5e-1,0e-1,5e-1.trans.450e-2,
180e-2,150e-2.scale"
# offset .5 back and .5 up (position is front bumper at road level), make
450cm long, 180m wide, 150m high
```