ALMA MATER STUDIORUM · UNIVERSITY OF BOLOGNA

School of Engineering and Architecture
Department of Computer Science and Engineering
Master's degree in Computer Engineering

Master's Degree Thesis
in
Mobile Systems M

Integrating V2X Communication and Infrastructure Sensing for Urban Digital Twins

SUPERVISOR
Prof. Paolo Bellavista

CANDIDATE
Baraldi Leonardo

CO-SUPERVISORS
Dott. Alessandro Calvio
Dott. Angelo Feraudo

Session I • A.Y. 2025 - 2026

Ai Taffi. . .

Abstract

This thesis introduces a proof-of-concept for a Digital Twin supported by technologies defined in the European ITS-G5 standard, primarily Cooperative Awareness Messages (CAMs) and Decentralized Environmental Notification Messages (DENMs), through a series of Roadside Units (RSUs) that participate in vehicle communication and forward all received messages to the system.

CAMs, used in Intelligent Transport Systems (ITS) to augment the perception of all entities, enable the Twin to track the movement of cars on a virtual map. DENMs, generated in the presence of hazardous situations, on the other hand allow the system to detect adverse events in the network and respond accordingly. To demonstrate the ability of the Twin to act on the data it gathers, a new simulation tool is introduced to test optimal strategies for redirecting traffic after a road closure. The system also features a component to analyze the collected data, a web application to visualize it, and an interface to configure the simulation environment by changing the network, traffic scenario, and percentage of intelligent vehicles.

An introduction to ITS is followed by a background analysis of the available technologies, the state-of-the-art of Digital Twins, and the chosen simulation framework (the SUMO/Artery/OMNeT++ stack). The design of the custom Twin is then described in detail.

Tests are conducted to assess the accuracy of the data collection strategy and evaluate how decreasing the penetration rate of the Cooperative Awareness (CA) service affects the creation of local traffic models. Use cases for both the traffic simulation tool and the Twin itself are presented at the end of the document.

Contents

Chapter 1

Introduction

The revolution that, in recent decades, has transformed technologies related to the personal sphere, bringing us smartphones and smart homes, does not yet seem to have fully reached the field of mobility. The heterogeneity of environments and the entities that navigate them, the intrinsic chaos that arises when a multitude of people are involved in the same activity, and the necessary but overwhelming burden of bureaucracy all contribute to making agreeing on standards — and even more so, deploying new technologies in this field — a monumental task.

Smart vehicles are not a novelty [1], yet real examples of smart cities remain rare [2]. Although electric vehicle sales continue to increase [3], and companies strive to perfect self-driving technology [4] or equip their vehicles with increasingly powerful computers [5], a future in which all cars are connected and cooperate still seems distant, if only due to the sheer scale of the global urban network.

This raises the question: What can be done *today*? Given that the technology is largely available from a research standpoint, and the need for new traffic management systems is clear [6], what can already be improved in our cities? What role can smart vehicles already on the road play? What does a real intelligent vehicular network look like, even on a small scale?

Urban mobility is slowly undergoing a profound transformation, driven by the convergence of communication technologies, distributed sensing, and the growing demand for smarter, safer, and more sustainable transportation systems. At the core of this transformation lies the concept of Intelligent Transport Systems (ITS) [7], technological frameworks that integrate vehicles, infrastructure, and control systems to optimize the movement of people and goods.

A key enabler of ITS is Vehicle-to-Everything (V2X) communication [8], which allows vehicles to exchange real-time information with other vehicles (V2V), roadside infrastructure (V2I), pedestrians (V2P), and network services (V2N). V2X provides the foundational capability for cooperation and coordination on the road, enabling faster response to hazards, adaptive traffic management, and efficient use of limited urban space.

These capabilities are supported by Vehicular Ad Hoc Networks (VANETs) — decentralized, rapidly changing networks formed among moving vehicles and fixed infrastructure [9]. Unlike traditional communication systems, VANETs must operate under high mobility, variable density, and unpredictable conditions, while ensuring low latency and high reliability. Solving these challenges is essential for unlocking the potential of Connected and Automated Vehicles (CAVs), which rely on local sensing together with V2X data to perceive their environment and make autonomous decisions.

In parallel, the concept of the Digital Twin (DT) has emerged. Simulations remain the most viable approach to producing meaningful results while mitigating or avoiding the need for widespread adoption and massive upfront investments. Numerous tools have been developed to manage the growing complexity of networked and digital systems by mimicking traffic and testing communication technologies, but among these the most powerful has proven to be the Digital Twin [10]. DTs aim to create virtual replicas of physical systems, updated in real time with data from their physical counterparts, evolving alongside them throughout their life cycle. DT technology promises to significantly enhance the capabilities of ITS, improving both safety and mobility [11]. The virtual model needs quality data to be constructed efficaciously, which is usually sourced either from databases or captured in real-time. In the context of urban mobility, a DT of a city aims at creating a virtual representation of traffic conditions and vehicle behavior by analyzing available data through simulations and machine learning. Many cities have already adopted traffic detectors, like induction loops under the surface of the most-used roads that count vehicles traveling above them. This kind of data is then periodically saved to a database and can be accessed through an API. For example, the city of Bologna keeps track of the hourly counts for its loops throughout the metropolitan area and makes them available in its “OpenData” portal [12] every month. If freshness of data is a concern, another strategy is to collect data directly, employing cellular technologies like 5G or sensors such as cameras and Lidars (light-based radars). The digital counterpart can then enable the city administrations to test interventions, monitor performance, and make proactive decisions, without relying solely on large-scale deployment or costly infrastructure upgrades.

One tried and tested approach to defining or deploying DTs is sourcing data directly from Roadside Units (RSUs). While their primary role is to aid vehicle communication by providing a support infrastructure that is both static and densely scattered, these same characteristics make them the perfect subjects for research that moves beyond their V2X duties. DTs can leverage RSUs as a data source either by mounting sensors on them [13][14] or by taking advantage of their communication capabilities, by employing custom protocols [15] and making predictions based on the received messages [16].

This thesis investigates how V2X communication and infrastructure-based sensing can be integrated to build effective and cost-efficient Digital Twins in complex urban environments, by creating a proof of concept for a DT with those characteristics using existing V2X protocols and simulation tools to bridge the gap between today’s roads and tomorrow’s smart cities, evaluating the feasibility and accuracy of such a system even under partial adoption scenarios. The objective is to use Cooperative Awareness Messages (CAMs) and Decentralized Environmental Notification Messages (DENMs) to retrieve vehicle data. Cooperative Awareness (CA) is a core component of the “ITS-G5” proposal for a European ITS standard, wherein messages are exchanged between CAVs to enhance their awareness of the surrounding environment. Since the adoption of such a technology is still in the early stages, the matter becomes how accurate traffic models generated from such data are. Despite its relative simplicity, this approach is scarcely represented in the literature, especially when applied to sophisticated networks such as city centers. By comparing real data from simulations with data sensed by connected vehicles, the analysis on how accurately traffic conditions are captured can be carried out, simulating various penetration levels of connected vehicles, and comparing perceived versus actual traffic states. The potential for the proposed system to be employed as a real DT is also demonstrated by showcasing the ability to construct simulations from the collected data, using a custom program to test traffic redirections after a road closure. Upon receiving sufficient event messages from the DENM protocol, the twin can request the best course of action for the congested area given the current traffic situation.

The document is structured as follows. Chapter 2 presents the necessary background for understanding the project. It begins with an introduction to intelligent transportation systems and a concise overview of European standards for Vehicle-to-Everything communication, focusing on ITS-G5 and the Cooperative Awareness and Decentralized Environmental Notification facilities. This is followed by a discussion on Digital Twins, their architecture, challenges, and real examples. The Artery-V2X simulation framework and its key components — SUMO, the mobility simulator, and OMNeT++, the network simulator — are then introduced.

Chapter 3 illustrates the actual project, a digital twin that tracks vehicles based on Cooperative Awareness and Decentralized Environmental Notification (DEN) messages, covering its architecture and noteworthy implementation choices. The chapter introduces all its components: the vehicle tracker (*Vesuva*), a data analysis library (*Tolaria*), a visualization web app (*Academy*), a graphical simulation manager (*Dominaria*), and a tool designed to test the impact of traffic redirection named *Agamotto*.

Finally, Chapter 4 presents the results, and Chapter 5 depicts some noteworthy use cases.

Chapter 2

Background

2.1 Vehicle-to-Everything

ITS have increasingly gained interest from both academic researchers and urban administrations due to their potential to drastically improve the experience and safety of all entities that make up vehicular networks. Traditionally, many vehicle safety features have been designed to operate independently, confined to onboard systems such as cruise control, ABS (Anti-Lock Braking System), lane departure warnings, or autonomous emergency braking. Although these systems are undeniably useful, their capabilities are inherently limited by the quantity and quality of the information that vehicles can collect through their sensors. To achieve higher degrees of automation and safety, vehicles must go beyond their isolated perception capabilities and engage in cooperative behavior through data sharing with external sources. This necessity brings forth the concept of vehicular networks, which requires a type of Mobile Ad Hoc Mobile Network (MANET) optimized for vehicle communication: a VANET [17].

Vehicular Ad Hoc Networks are specifically designed for high mobility and rapidly changing topologies, and have strict requirements for low latency and high reliability. They provide the foundation for V2V and V2I communication, in which vehicles act as network nodes. However, modern ITS design has expanded this scope even further with the emergence of the so-called V2X paradigm, a “Internet of Vehicles” of some kind [18].

2.1.1 V2X Overview

V2X refers to a family of communication technologies that allow vehicles to interact not only with other cars (V2V), but also with roadside infrastructure (V2I), pedestrians (V2P), and centralized networks such as cloud services (V2N). The goal of V2X is to provide vehicles with an augmented view of their environment, enabling more accurate and timely decisions for driving safety and traffic coordination.

Although the basic ideas underlying V2X can be traced back to research in the 1970s [19], it was not until 2010 that a widely accepted standard, IEEE 802.11p, was published [20]. This standard, referred to as Dedicated Short-Range Communications (DSRC) in the US [21], was designed to facilitate low-latency, direct wireless communication in vehicular environments. Despite this early standardization, the first mass-produced V2X-enabled vehicle only appeared in 2016, introduced by Toyota [22].

The 3rd Generation Partnership Project (3GPP) consortium for international mobile communications standards identifies several use cases for V2X technology [23], categorized in 4 main groups [24]:

Vehicle Platooning supports the formation of a group of vehicles that are interconnected in a virtual chain. This use case includes Platoon formation, different levels of data resolution for Information Exchange, Cooperative Driving, and Changing Driving-Mode.

Remote Driving allows vehicles to be controlled remotely, whether by humans or cloud/edge computing applications, in the event that an autonomous vehicle is not able to drive autonomously, must operate in dangerous and harsh conditions or just so that the driver does not have to be physically present in the vehicle.

Extended Sensors enable the exchange of data (both raw and processed) collected through local sensors, live video images, RSUs, pedestrian devices, and V2X application servers. Vehicles can increase the awareness of their environment beyond what their sensors can detect. This group includes three use cases: Sensor and State Map sharing, Collective Perception of Environment, and Video Data sharing for Automated Driving.

Advanced Driving enables a high Level of Automation (LoA) to reach fully automated driving. Each entity shares data obtained from its local sensors with vehicles in proximity and notifies its driving intentions, thus allowing them to coordinate their trajectories and maneuvers. The benefits of this use case group are safer traveling, collision avoidance, and improved traffic efficiency. Use cases include Cooperative Collision Avoidance, Emergency Trajectory Alignment, Cooperative Lane Change, and 3D video composition.

Several technical challenges arise when considering these use cases and the environment as a whole [19]. A central challenge of VANETs is that no communication coordinator or handshaking protocol can be assumed. Although some applications may involve the use of infrastructure, most of them are expected to operate reliably using decentralized communication. Given that many applications will be broadcasting information of interest to all surrounding cars in a short period, the necessity of a single, shared control channel can be derived, as making use of more than one channel would lead to multi-channel synchronization and co-channel interference problems.

Other challenges include the dynamic network topology, which is based on the mobility and speed of the vehicles, and the environmental impact on radio propagation. The latter must take into account that the low antenna heights and the attenuation/reflection of all the moving metal vehicle bodies result in adverse radio channel conditions. Altogether, VANETs must work properly in a wide range of conditions, including sparse and dense vehicular traffic.

Two mainstream approaches emerged to support V2X and its demands for highly efficient low-latency connections [25]:

- **WLAN-based V2X:** includes standards, such as the aforementioned IEEE 802.11p, which enable direct and infrastructure-less communication between nodes, thanks to an approach to Medium Access Control (MAC) based on Carrier Sense Multiple Access (CSMA).
- **Cellular V2X (C-V2X):** takes advantage of cellular networks (e.g., 4G, 5G) to provide V2X functionality. It enables both direct communication and broader access to cloud services, with support for network slicing and URLLC (ultra-reliable low-latency communication).

2.1.2 Technologies

This Thesis focuses on WLAN-based V2X and particularly ETSI (European Telecommunications Standards Institute) ITS-G5 [26], a standard based on 802.11p (a variant of Wi-Fi optimized for vehicular environments) that operates in the 5.9 GHz band. ITS-G5 is designed to support low-latency, high-reliability communication between vehicles and infrastructure, enabling applications such as traffic safety, traffic management, and infotainment.

ITSC, or Intelligent Transport System Communications, is a type of communication system dedicated to transportation scenarios, as exemplified in Figure 2.1. These systems form the backbone of modern ITS deployments, enabling a wide range of cooperative functionalities in both urban and highway contexts.

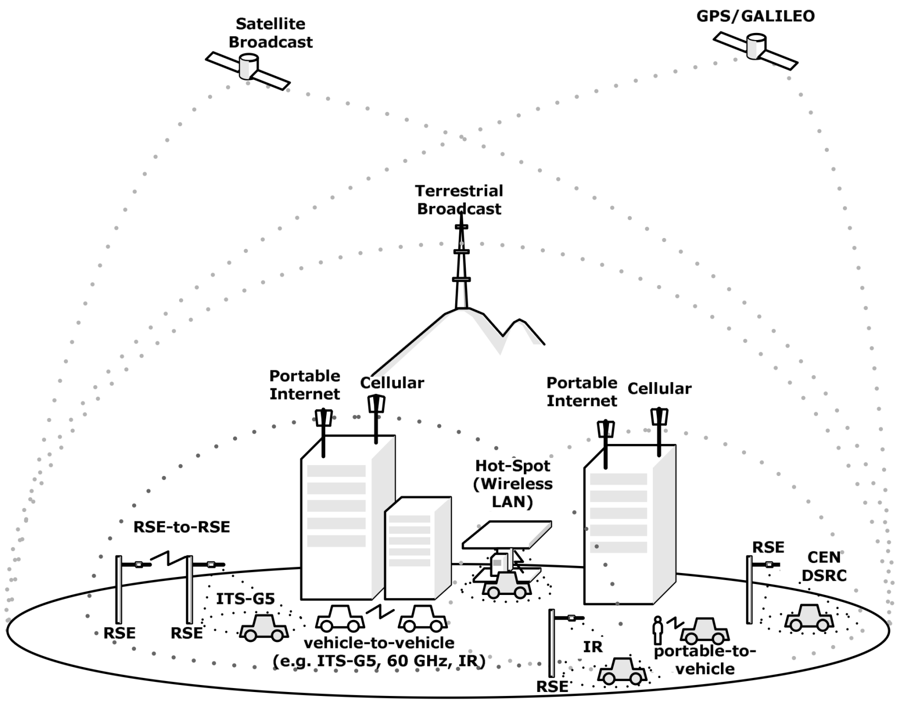


Figure 2.1: Example of an ITS communication scenario including vehicles, infrastructure, and pedestrians [27]

ITS-G5 meets the criteria of being a decentralized, broadcast-based architecture that allows vehicles to communicate directly with each other and with RSUs in their radio range without relying on a central infrastructure. ITS-G5 is especially suited for time-critical applications like hazard notifications and Cooperative Awareness, and as such supports various applications, including CAMs, DENMs, and CPMs (Collective Perception Messages). This Thesis will focus on CAMs, used to exchange information about the vehicle's status, and DENMs, used to notify vehicles of hazardous situations.

According to the standard [27], an ITS network is made up of multiple types of ITS-Stations (ITS-S), such as:

- **Vehicle ITS-S:** Installed in cars, trucks, or buses.
- **Roadside ITS-S:** Installed on infrastructure such as RSUs.
- **Central ITS-S:** Part of backend traffic control or monitoring systems.
- **Personal ITS-S:** Found on user devices such as smartphones.

Each ITS-S follows a layered architecture derived from the OSI model, with adaptations specific to ITS requirements. A simplified architectural model is shown in Table 2.1.

ITS-S Layer	**Corresponding OSI Layers**
Access	Physical and Data Link (Layers 1-2)
Networking & Transport	Network and Transport (Layers 3-4)
Facilities	Session, Presentation, Application (Layers 5-7)

Table 2.1: Mapping between ITS-S architecture and OSI layers

The Facilities layer is the one of interest for this Thesis, as it is responsible for managing higher-level services such as message construction, event detection, and communication policies, and contains the ITS applications of interest.

CAM - Cooperative Awareness Messages

Cooperative awareness [28] within road traffic means that road users and the roadside infrastructure are informed about each other's position, dynamics, and attributes. Information about the local environment is essential in cooperative ITS systems, as applications require knowledge both on moving objects, such as other vehicles nearby, and on stationary objects, such as traffic road signs. Common information required by different applications can be maintained in a conceptual data store located within an ITS station called a Local Dynamic Map (LDM), which contains information relevant to the safe and successful operation of ITS applications [29].

Data can be received from a range of different sources and is organized into four types:

- Type1: permanent static data, usually provided by a map data supplier.
- Type 2: transient static data, obtained during operation, e.g., changed static speed limits.
- Type 3: transient dynamic data, e.g., weather situation, traffic information.
- Type 4: highly dynamic data (CA).

The dynamic information to be exchanged for Cooperative Awareness is packed up in the periodically transmitted CAM. The construction, management, and processing of CAMs is done by the Cooperative Awareness base service (CA base service), which is part of the facilities layer within the ITS communication architecture. The CA base service is a mandatory facility for all kinds of ITS-S, which take part in road traffic (vehicle ITS-S, personal ITS-S, etc.) [28].

CAMs contain information about the vehicle's position, speed, heading, and other relevant data, allowing vehicles to be aware of each other's presence and status. Vehicles send updates typically every second, broadcasting to cars and RSUs within the radio range to allow real-time awareness of the surrounding environment. The information contained in the messages can be used to detect potential collisions, monitor traffic conditions, and support ADASs (advanced driver assistance systems).

CAMs are transmitted point-to-point, and received messages are never propagated. Generation frequency is managed by the CA base service; it defines the time interval between two consecutive creations, setting the upper and lower limits at 100ms and 1000ms. Within these limits, CAM generation is triggered depending on the originating ITS-S dynamics and the channel congestion status. In case the formers lead to a reduced generation interval, this is maintained for many consecutive messages.

Two parameters govern the generation logic:

- **T_GenCam_Dcc:** Minimum interval between two consecutive CAMs.
- **T_GenCam:** Currently valid upper limit of the CAM generation interval.

A CAM is immediately generated if:

1. The time elapsed since the last generation is $\geq$ T_GenCam_Dcc and one of the following dynamics-related conditions is true:
    - the absolute difference between the current and the previously transmitted heading exceeds 4°;
    - the distance between the current and the previously transmitted position exceeds 4m;
    - the absolute difference between the current and the previously transmitted speed exceeds 0,5 m/s.
2. The time elapsed since the last generation is $\geq$ T_GenCam and $\geq$ T_GenCam_Dcc.

CAM data is stored in containers constructed by the CA base service. Some include the highly dynamic information of the originating ITS-S and are mandatory:

- **basicContainer**: Contains basic information about the vehicle, such as its position, speed, and heading.
- **highFrequencyContainer**: Contains time-critical information that changes frequently, including the vehicle's acceleration, yaw rate, and other dynamic attributes.

Additionally, a CAM may include optional data:

- **lowFrequencyContainer**: Contains information that is not time critical, for instance, vehicle dimensions, vehicle type, or other static attributes.
- **specialVehicleContainer**: Contains information specific to certain types of vehicles, such as emergency or public transport vehicles.

Some of the mandatory and optional fields contained in a CAM are:

- **Position**: The position of the vehicle in the WGS84 coordinate system, expressed as latitude and longitude.
- **Speed**: The speed of the vehicle in meters per second.
- **Heading**: The direction of the vehicle in degrees relative to true north.
- **Vehicle Type**: The type of vehicle, such as car, truck, or bus.
- **Vehicle Length / Width / Height**: The dimensions of the vehicle in meters.
- **Vehicle ID**: A unique identifier for the vehicle.
- **Timestamp**: The time at which the CAM was generated.

The time field is called generationDeltaTime, but, despite the name, does not represent the time elapsed since the last CAM generation. It is the time of the reference position, considered as the time of the message creation. Its value is set as the remainder of TimestampIts divided by 65536 (2^{16}), where TimestampIts represents an integer value in milliseconds since 2004-01-01T00:00:00:000Z. GenerationDeltaTime is thus a number of milliseconds wrapped to 65536, so it does not represent a moment in time by itself.

DENM - Decentralized Environmental Notification Messages

The DEN basic service [30] is another application support facility provided by the Facilities layer that constructs, manages, and processes DENMs. The construction of a message is triggered by an ITS-Station application and contains information related to a road hazard or abnormal traffic conditions, such as its type and position.

Examples of scenarios that trigger a DENM include:

- Detection of an obstacle or accident.
- Roadworks or temporary lane closures.
- Weather hazards (fog, ice, flooding).
- Emergency vehicle approaching.

Typically, a DENM is disseminated to ITS-S that are located in the same geographic area. On the receiving side, the DEN basic service of a receiving Station processes the received message and provides its content to an application that may present the information to the driver if it is assessed to be relevant.

DEN dissemination may be repeated and persist as long as the event is present. A DENM may be forwarded (by the ITS Networking & Transport layer) by intermediate stations in order to disseminate messages from the originating ITS-S to the receiving ITS-S, if the receiving station is not located in the direct communication range of the originating one. In addition, the DEN basic service may provide forwarding functionality at the facilities layer, to maintain the DENM retransmission in certain situations, for example, when the originating ITS-S has lost the capability to repeat message transmission. The messages appear to linger in an area, as they are repeatedly sent in the vicinity of the event until it is terminated.

Termination is either automatically achieved by the facilities layer, for example, when a predefined expiry time is reached, or by an ITS-S application that requests the generation of a message to notify that the event has terminated.

The following DENM types are defined:

- **New DENM**: generated when an event is detected by an originating ITS-S for the first time.
- **Update DENM**: includes update information of an event, sent by the same originating ITS-S which had generated the event.
- **Cancellation DENM**: informs the termination of an event, sent by the same originating ITS-S that had generated the event.
- **Negation DENM**: informs the termination of an event, sent by a different ITS-S than the one that had generated the event (in case the originating station has left the zone).

To support dissemination, event identification is enabled by the parameter actionID. Each time a new DENM is generated upon an application request, a new actionID value is assigned.

Other parameters included in a message are:

- **Event Position**: The position of the event in the WGS84 coordinate system, expressed as latitude and longitude.
- **Event Type**: The type of event, such as accident, roadworks, or traffic jam.
- **Relevance Distance**: The distance at which event information is relevant for the receiving ITS-S, starting from the event position.
- **Detection Time**: The time at which the event was first detected.
- **Reference Time**: The time at which a new or update DENM was generated.
- **Validity Duration**: Set by the originating ITS-S, it represents an estimate of how long the event may persist. It translates to the duration over which the DENM should be kept at the DEN basic service of the receiving station, and the DENM dissemination should be maintained in the relevance or destination area. In case the expiry time of the event cannot be estimated, a default value is used. This field may be renewed if the preset expiry time has reached its limit and the originating station detects that the event persists.

2.2 Digital Twins

A Digital Twin is a virtual model of a real object or system that can be used to simulate and analyze its behavior. It is a powerful tool for understanding complex systems, as it allows the testing of different scenarios and the prediction of outcomes without the need for physical experimentation. Digital Twins are at the forefront of the Industry 4.0 revolution facilitated by advanced data analytics and the Internet of Things (IoT) connectivity, which has increased the volume of data usable from manufacturing, healthcare, and smart city environments [31].

Formal ideas around Digital Twins have been around since the early 2000s, with the first terminology given in a 2003 presentation at NASA (later documented in a white paper [32]) about a space vehicle twin. The concept of a digital twin was later expanded to include more types of systems, with applications in predictive maintenance and fault detection in manufacturing processes, anomaly detection in patient care, and traffic management in smart cities.

Definitions can vary from one author to another, but the core idea remains the same:

> A Digital Twin is a virtual instance of a physical system (twin) that is continually updated with the latter's performance, maintenance, and health status data throughout the physical system's life cycle. [33]

There exist some misconceptions about digital twins, as not all digital copies of real systems can be considered true twins. Depending on the different level of data integration between the physical and digital model, a classification can be derived [34]:

- **Digital Model**: digital version of a preexisting or planned physical object, with no automatic data exchange between the physical and digital model. Once the digital model is created, a change made to the physical object has no impact on the digital model and vice versa.
- **Digital Shadow**: a virtual representation that has a one-way flow between the physical and digital object. A change in the state of the physical object leads to a change in the digital object, but not the other way.
- **"True" Digital Twin**: a virtual representation that has a two-way flow between the physical and digital object. A change made to the physical object automatically leads to a change in the digital object, and the digital copy can (directly or indirectly) act on the physical counterpart.

The manufacturing sector saw the most significant impact of this technology, which allows for the simulation of production processes, the optimization of supply chains, and the prediction of equipment failures - all ways to save time and money. The current trend follows the growth of the Industry 4.0 concept, coinciding with the fourth industrial revolution. This approach connects devices and systems, generating a volume of unstructured, heterogeneous big data never seen before, opening up opportunities for smart manufacturing and making it possible to leverage Digital Twins to help monitor and improve processes in real time [35].

The typical structure for DTs in construction applications is shown in Figure 2.2. It includes a multi-layered framework that integrates sensing, communication, storage, analytics, and visualization capabilities.

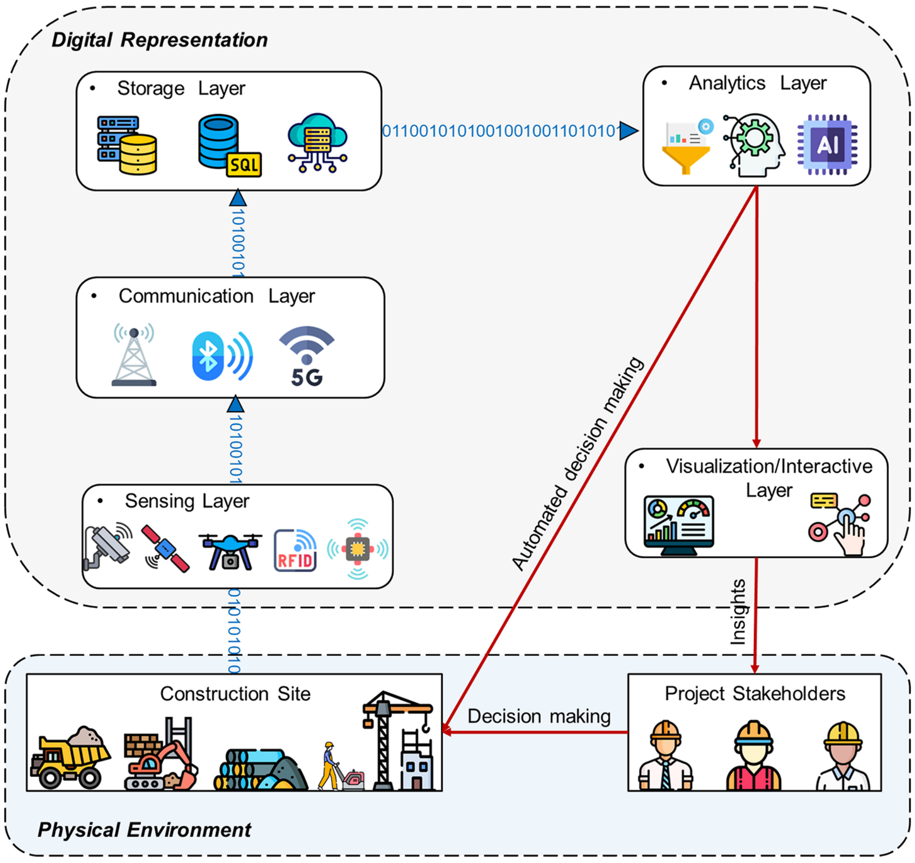


Figure 2.2: Structure for a Digital Twin of a construction site [36]

Smart cities are the next big logical application for Digital Twins, and their potential to be dramatically effective within a smart city is increasing due to rapid developments in connectivity through IoT and the growing number of smart cities developed. By analyzing the main challenges associated with Digital Twins [31], it is possible to identify the main areas of research and development that need to be addressed to make them a reality in smart cities:

- **IT Infrastructure**. A connected and well-thought-through infrastructure is needed to support the effective running of a Digital Twin. This is also true for data analytics and privacy, which are crucial to ensure that the data collected is accurate and secure. A significant challenge for IoT systems is connecting legacy machines to the IoT environment, particularly in urban settings and the vehicular network as a whole.

- **Quality of data**. A successful system needs useful data that is noise-free and provided in a constant, uninterrupted stream. If the data is poor and inconsistent, it runs the risk of the Digital Twin greatly underperforming. Planning and analysis of device use are hence needed to ensure that the right data is collected and correctly filtered.
- **Privacy**. Within an industry or city setting, privacy and security associated with Digital Twins are a challenge. Sensitive system data is at significant risk, aggravated by the sheer volume of information that needs to be handled. To overcome this, the key enabling technologies - data analytics and IoT - must follow the current practices and updates in security and privacy regulations, like GDPR (General Data Protection Regulation).
- **Trust**. An often-overlooked aspect, trust is crucial for the success of any new technology, both for organizations and the end users. This comes from adequate discussion and explanation at a foundation level and from demonstrating that Digital Twins can perform at the expected level of quality. Setting and managing expectations are thus necessary.
- **Standardization**. From initial design to the final product, there needs to be a standard approach to ensure domain and user understanding while ensuring information flow between each stage of the development and implementation of a Digital Twin. Since the models of the physical world that compose the digital twin belong to different domains, they require different tools and formalisms whose interoperability can only be achieved with great effort.
- **Domain Modeling**. As a result of the need for standardization, ensuring that information related to domain use is transferred to each development and functional stage of a Digital Twin's modeling guarantees compatibility with domains such as IoT and data analytics.

The architecture of an urban DT closely resembles the example of Figure 2.2, related to the construction field. In fact, it is possible to synthesize the layers [36] present in any Digital Twin (Figure 2.3) as:

- **Perception layer**: responsible for the collection and management of data from the physical world. Depending on the deployment strategy, it can provide light filtering or full-on analysis before committing data to storage. It contains two additional layers:
  - **Sensing layer**: the first and most fundamental layer of any DT architecture, that encompasses one or more types of data-acquisition devices responsible for collecting raw data from the physical environment in which they are deployed.
  - **Communication layer**: it serves as the bridge between the physical site and its digital representation, facilitating the transmission of the data collected by the sensing layer and transferring it to the digital hub to be stored, processed, and visualized.
- **Storage layer**: once data is collected and transmitted, it needs a secure and scalable storage infrastructure. The storage layer stores vast amounts of sensing data along with historical records, project documentation, and other relevant information that can collectively be analyzed to draw insights.
- **Analytics layer**: it forms the intelligence hub of the digital twin system, responsible for processing and translating the data. Here, advanced algorithms, machine learning models, and statistical analytical tools process the raw data to derive insights.
- **Visualization layer**: the DT frontend, where the processed data gets presented in insightful and proper formats that enable easy access for quick decision-making.
- **Player**: the end-user of a Digital Twin is whoever is responsible for deployment and decision making. In the urban environment, city administrators and law enforcement typically fulfill this role.

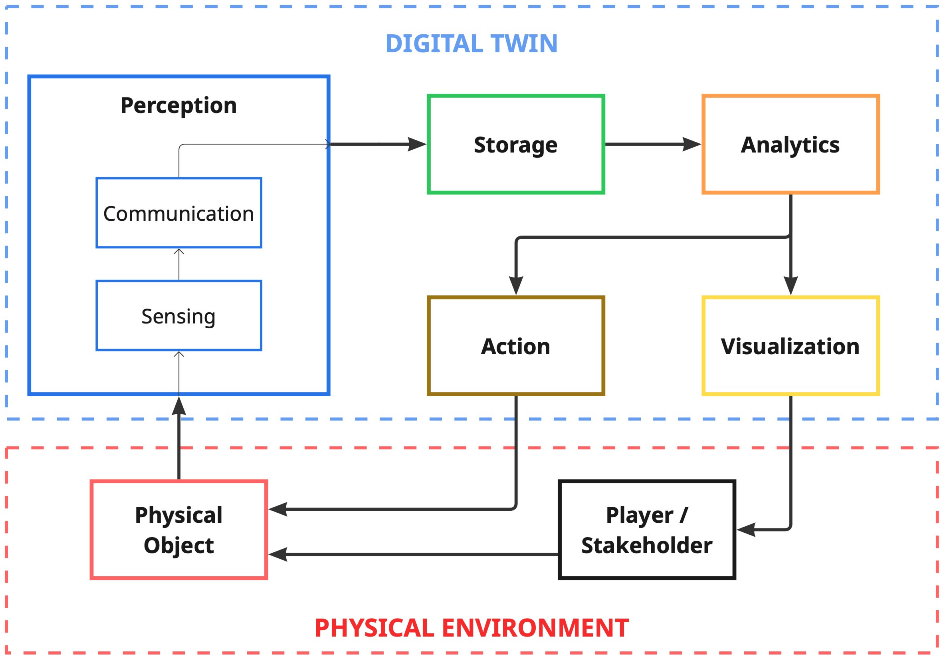


Figure 2.3: Digital Twin architecture

There are very few examples of already working (or planned) real Digital Twins for cities around the world. The approach is usually to strive to obtain the most accurate digital version of the source, to help city planning through simulations. For example, the Boston Digital Twin [37] has the goal of aiding the design of next-generation 6G networks, and is capable of instantaneous rendering and programmatic access to the building models, and can accurately represent the electromagnetic propagation environment in the real-world city of Boston. The nation of Singapore is also developing a DT to simulate the areas most at risk of flooding [38], while in Helsinki, the ever-evolving “Helsinki-3D” project [39] helps planning sustainable urban expansions. There are also Twins designed with traffic prediction and increased road safety in mind. A monumental effort of mapping the largest city in China led to the creation of the Shanghai Twin [40], assisting the local police during emergencies by overlaying live surveillance footage, traffic movement, and heat maps.

The goal of this Thesis is to analyze the level of quality achieved by using an unconventional source for the data, such as Cooperative Awareness Messages, for a Digital Twin. CAVs, or smart vehicles, are not a rare sight, but their diffusion is probably far from ideal to guarantee the required flow of data. Still, the cost-effectiveness of such a solution compared to those that rely on computer vision and sensor-based sensing is worth looking into, as the accuracy of traffic models starting from partial data may turn out to be adequate for most applications.

2.3 Simulation environment

Complex fields benefit significantly from the use of simulations, which can generate large amounts of valuable data without the need to deploy in the real world test solutions that would require several thousand entities.

This work deals with both traffic analysis and novel telecommunications technologies: simulated vehicles need to traverse an urban network while accurately simulating radio communication. Fortunately, many tools have been developed to assist in these tasks. Among them, one joins two of the most used applications for testing traffic and wireless networks: Artery-V2X.

2.3.1 Artery-V2X

Artery-V2X (from here on referred to as "Artery") is a simulation framework based on ETSI ITS-G5 protocols like GeoNetworking and BTP (Basic Transport Protocol). The project initially started as an extension of "Veins" (Vehicles in Network Simulation), which put emphasis on the US counterpart of ITS-G5, WAVE (Wireless Access in Vehicular Environments). Artery simulations see two main programs working together, as shown in Figure 2.4.

One is OMNeT++ [41], an extensible, modular, component-based C++ simulation library and framework, primarily for building network simulators. The other is SUMO [42], the popular open-source traffic simulator suite developed initially by the German Aerospace Center in 2001 and since 2018 a project of the Eclipse Foundation. SUMO allows modeling of inter-modal traffic systems - including road vehicles, public transport, and pedestrians. A wealth of supporting tools, many developed by the community, are bundled with the program and help automate core tasks for the creation, execution, and evaluation of traffic simulations, such as network import, route calculations, visualization, and emission calculation.

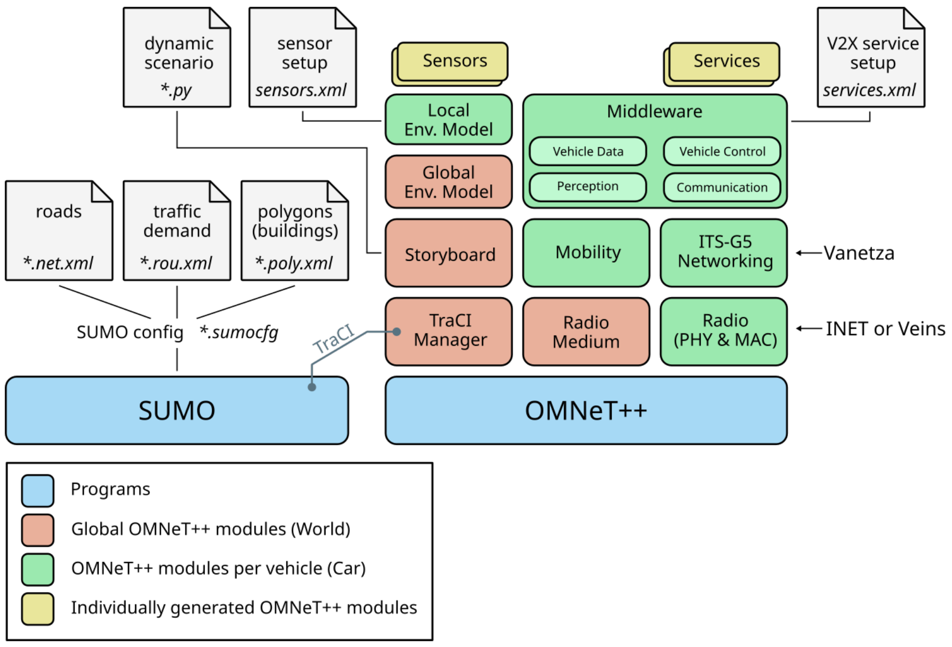


Figure 2.4: Artery-V2X architecture

OMNeT++ is responsible for driving the simulation forward, controlling SUMO through its “TraCI” interface. Artery also does not provide any simulation model of wireless communication itself, but employs existing models, such as INET’s IEEE 802.11 implementation [43].

2.3.2 OMNeT++

OMNeT++, or OMNeTpp, is a discrete event simulator written in C++ [41]. It supports a wide range of domains, including sensor networks, wireless ad-hoc networks, Internet protocols, performance modeling, photonic networks, and more. It has gained widespread popularity as a simulation platform for communication networks in both academic and industrial settings, resulting in a large user base.

OMNeT++ provides a flexible, component-based architecture for building models. Components - called *modules* - are implemented in C++ and can be composed into larger hierarchical systems using a high-level topology description language called NED (NEtwork Description). OMNeT++ also features an Eclipse-based IDE, a command-line interface, interactive graphical simulation tools, and various result processing utilities.

The most commonly used protocol library for OMNeT++ is the INET Framework, which provides models for a wide variety of Internet protocols and wireless technologies. INET is often extended or customized to suit the specific needs of a simulation study, for example, with extensions like Artery for simulating V2X communications.

Workflow

The official OMNeT++ documentation [44] outlines a typical workflow to develop and run simulations:

1. Build an OMNeT++ model from components (modules) that communicate by exchanging messages. When creating the model, the system needs to be organized into a hierarchy of communicating modules.

2. Define the model structure in the NED language. NED files can be edited in a text editor or in the graphical editor of the Eclipse-based OMNeT++ Simulation IDE.

3. Program the active components of the model (simple modules) in C++, using the simulation kernel and class library. Classes that represent protocol headers are described in MSG files, which are then translated into C++ code.

4. Provide a suitable omnetpp.ini file to hold OMNeT++ configuration and parameters for the model. One .ini file may hold several configurations that can build on one another, and may even contain parameter studies.

5. Build the simulation program and run it. Code is linked with the simulation kernel and one of the user interfaces OMNeT++ provides, such as CMDEnV (command line), and QTEnv (GUI).

6. Simulation results are written into the output vector and output scalar files. These can be analyzed and plotted using the Analysis Tools powered by Pandas and Matplotlib in the Simulation IDE. Result files are text-based, so they can be processed by any tool.

Messages and Signals

OMNeT++ uses messages to represent events, sent from one module to another - this means that the place where the “event will occur” is the message’s destination module, and the model time when the event occurs is the arrival time of the message. Events like “timeout expired” are implemented by the module sending a message to itself.

Signals are similar to messages, since they are also sent from one module to another, but they are used to broadcast information. Signals can be used to notify other modules about events without requiring the explicit sending of a message. Signals are emitted by components (modules and channels) and propagate on the module hierarchy up to the root. At any level, listeners can be registered (objects with callback methods). These listeners will be notified by calling their appropriate methods whenever a signal value is emitted. The result of upwards propagation is that listeners registered at a compound module can receive signals from all components in that submodule tree, and a listener registered at the system module can receive signals from the whole simulation.

Simulation signals can be used for:

- exposing data for statistics collection purposes, without specifying how, where, and whether to record them.
- receiving notifications about simulation model changes at runtime, and acting upon them.
- implementing a publish-subscribe pattern of communication among modules.
- emitting information for other purposes, for example, as input for custom animation effects.

NED Files

The NED (Network Description) language is used in OMNeT++ to define the static structure of a model — its components, how they are connected, and how they are nested. NED files have a ‘.ned‘ extension and support both graphical and textual editing within the OMNeT++ IDE.

Modules, whose logic is defined in C++ files, can be either *simple* or *compound*. Each module can declare parameters, gates (input/output ports for message passing), and more. Simple modules are then wired together in compound modules to form a complete network.

```
module Host {
    parameters:
        int capacity;
    gates:
        input in;
        output out;
    submodules:
       tcp: TCP;
       ip: IP;
    connections:
       tcp.ipOut --> ip.tcpIn;
       tcp.ipIn <-- ip.tcpOut;
}
```

Listing 2.1: .ned file example

Parameters are variables that belong to a module. Parameters can be used in building the topology (number of nodes, etc.), and to supply input to C++ code that implements simple modules and channels. Parameters can get their value from NED files or from the configuration (omnetpp.ini file).

NED files can be organized into packages, which are directories that contain NED files and other resources. Packages can be imported into other NED files using the 'import' statement, allowing for modular design and reuse of components, thanks to the ability to inherit from previously defined modules, extending their functionality.

omnetpp.ini file

The omnetpp.ini file serves as the main configuration file for a simulation run. It defines global and module-specific parameters, selects which network to instantiate, and configures output options, random seed settings, and more. The file is written in an INI-style format with key-value pairs grouped under named configurations.

Keys may be classified according to their syntax:

- Keys without dots (e.g., 'sim-time-limit') represent global or per-run configuration options.
- Keys with dots (e.g., '*.module.parameter') represent per-object configuration options.
- Otherwise, the key represents a parameter assignment.

```
[General]                          # section heading
network = Foo                      # configuration option
debug-on-errors = false
sim-time-limit = 500s

# per-object configuration options with wildcards
**.vector-recording = false
**.app*.typename = "HttpClient"
**.app*.interval = 3s

[Config Test1]
**.app*.interval = 5s
# overwrites the previously set value
        if this configuration is selected
```

Listing 2.2: omnetpp.ini example

Parameters can be specified using wildcard expressions (e.g., *.speed = 25) to apply settings across multiple modules, navigating the entire hierarchy.

OMNeT++ allows multiple named configurations within the same INI file using headers such as [Config Test1]. These configurations can inherit from others using the "extends" keyword, to factor out the common parts of several configurations into a "base" configuration, and to reuse existing configurations without copying, by using them as a base.

The omnetpp.ini file plays a central role in reproducibility and experimentation, allowing users to easily switch between scenarios, toggle logging settings, and perform parameter studies — all without changing the simulation code or topology.

2.3.3 SUMO - Simulation of Urban MObility

SUMO [42] is not simply a building block of Artery; it is also at the core of the project developed for this Thesis.

SUMO is an open-source, microscopic, multi-modal traffic simulation framework [45]. It allows for the simulation of how a given traffic demand, consisting of single vehicles, moves through a given road network. The simulation allows for addressing a large set of traffic management topics, and it is purely microscopic: each vehicle is modeled explicitly, has its own route, and moves individually through the network. Simulations are deterministic by default, but there are various options for introducing randomness.

The following Section defines the nomenclature adopted throughout the remainder of this document. SUMO models road networks as a directed graph, where nodes, or "junctions", represent intersections, and "edges" represent roads or streets. Every street (edge) is a unidirectional entity that includes a collection of lanes, along with their position, shape, and speed limit. The allowed turns from one lane to another when at a junction are called "connections". Junctions, under the hood, contain "internal edges" and "internal junctions", which aid with more granular control of vehicle behavior at an intersection. Right-of-way is also correctly modeled, and networks can also contain working traffic light definitions.

A simulation is run in a series of "steps" of 1 second by default. After every step, vehicles are moved, and data is collected.

Configuration

SUMO networks can be fully described by a collection of XML files. The list can either be passed to the sumo command in the terminal or bundled inside a configuration file (.sumocfg). Networks must use Cartesian, metric coordinates where the leftmost node is at x=0 and the node that is most at the bottom is at y=0. This means that, when being imported, if the original network was not using Cartesian and/or metric coordinates, it needs to be projected to the correct system.

Networks are useless without vehicles traversing them. Traffic can be modeled in many ways, depending on the amount of information required to govern the visited edges. A trip is a vehicle movement from one place to another defined by the starting edge (street), the destination edge, and the departure time. A route is an expanded trip that contains not only the first and the last edge, but all edges through which the vehicle will pass. When the simulation is running, routing algorithms will decide how trips are converted to routes. If trips are to be used by many vehicles, they can be organized into flows: the departure time will be distributed according to a chosen heuristic, but all vehicles will depart and arrive at the same edges.

If the frequency of updates is too low, SUMO can be told to increase the number of steps it simulates in a second. This is governed by the field `step-length`, which, like all configuration options, can either be passed to the terminal command or written in the sumocfg file. Reducing the time simulated in every step enables achieving greater fidelity, albeit at the expense of execution speed.

TraCI

SUMO can be enhanced with custom models and provides various APIs to control the simulation remotely. TraCI, or "Traffic Control Interface", gives access to a running road traffic simulation, allowing to retrieve values of simulated objects and to manipulate their behavior "online". TraCI bindings are available for many programming languages and can act upon almost the totality of the entities of the simulation. This is the interface used by Artery to control the SUMO simulation from OMNeT++, and it is made available to custom-defined scenarios too.

TraCI operates using a client-server architecture: SUMO acts as the server, while external applications (clients) connect to it via a TCP socket. During the simulation, the client can issue commands to SUMO at each simulation step, such as adding or removing vehicles, changing routes, modifying traffic light states, or retrieving information about the current state of the simulation (e.g., vehicle positions, speeds, emissions, etc.).

The typical workflow involves starting SUMO in TraCI server mode, then running a client script (often written in Python using the `traci` library) that connects to SUMO, steps the simulation forward, and interacts with the simulation as needed. This enables advanced use cases such as closed-loop control, online data collection, and integration with other simulators or optimization algorithms.

Some of the main features accessible via TraCI include:

- **Vehicle control:** Add, remove, reroute, or modify vehicles in real time.
- **Traffic light management:** Dynamically change traffic light phases, durations, or programs.
- **Edge and lane manipulation:** Retrieve or modify properties of roads and lanes.
- **Subscription mechanism:** Monitor changes in simulation objects over time.
- **Simulation control:** Pause, resume, or step the simulation, and synchronize with external processes.

Other tools

A tool that can significantly speed up network generation is OSMWebWizard.py, a web tool to construct networks with data from the free and open-source Open Street Map [46] project. Some random routes are included, but SUMO is bundled with a series of much better tools to generate routes, that span from completely random to determined from density and turn probability data. The latter are the exact data this Thesis aims to generate, so it is best to investigate those that use that. Among the available tools, there are:

- **Duarouter**: Creates routes from demand files.
- **Dfrouter**: Creates routes according to induction loop counts. It requires data for each edge and lacks support for complex networks.
- **Flowrouter**: Drop-in replacement for Dfrouter that tries to deduce traffic information for missing induction loop data.
- **Jtrrouter**: Creates routes from vehicle flows and turn probabilities for each junction.
- **RouteSampler**: Picks routes from an existing source to satisfy given metrics.

2.3.4 Scenarios and services

Artery simulations are defined in folders called “scenarios”. These contain all the files necessary to run a simulation, including SUMO networks, OMNeT++ NED files, and configuration files.

The omnetpp.ini file must contain the field ‘*.traci.launcher.sumocfg’ to inform the middleware of the SUMO scenario to launch. Additional OMNeT files can be provided, such as services.xml, to specify the services that vehicles and other entities will run. Custom services can be placed directly in the scenario folder alongside other files.

Traffic scenarios modeled with SUMO are usually “static”. Even though vehicles are moving around, traffic density varies, and traffic jams may build up, SUMO has no notion of sudden changes. However, V2X use cases often depend on said sudden changes, for example, unexpected heavy rain slowing down traffic in one region of the map, or accidents and vehicle breakdowns happening. Those quite dynamic changes in the environment can be modeled with Artery’s storyboard feature. The ‘storyboard’ module integrates a Python interpreter and loads a storyboard script, supplied in the scenario folder, with the function `createStories(board)`, which the storyboard will invoke at the beginning. Multiple stories can be registered via this script, each consisting of conditions and effects.

Conditions are properties that are evaluated at runtime and need to be fulfilled. Effects are applied to the matching vehicle when the related conditions are fulfilled. They can even be signals, which can be used to trigger events in the vehicle’s services - further linking the SUMO simulation to the OMNeT++ simulation.

Chapter 3

System Design

3.1 Overview of the project

As discussed in the previous chapters, the goal for this Thesis is to develop the proof of concept for a Digital Twin that is supported by data provided by the technologies in the ITS-G5 standard, like CAM and DENM, with messages streamed from RSUs scattered throughout the urban network. Cooperative Awareness Messages are used to create local traffic models, through an analysis akin to how induction loops in real cities are employed to count vehicles on the streets. Receiving DENMs also allows the DT to act on hazardous events, as exemplified in Chapter 5.

The core of the project is thus the creation of a competent simulation environment that can track real vehicles on a virtual map. Data, saved in a database, will be used to extract traffic models that can be analyzed by external programs or directly adopted by the DT itself to reason about the current situation in the city.

Since real-world testing was not feasible, all vehicles and infrastructures were simulated; however, work on the Twin was carried forward as if the source of data were a real city. This adds some complexity when thinking about the problem at hand: the Twin needs to support an environment that, due to its simulated nature, is unrealistically precise and has a flow of time that is uneven and different from the real world. However, the resulting solution is more elegant and could actually be deployed in a real urban scenario.

The following requisites can thus be outlined:

- The "City" hosts some RSUs. They receive all Cooperative Awareness Messages from vehicles in their radio range.
- The RSUs must also be able to receive Decentralized Environmental Notification Messages.
- RSUs must stream all data through a single WebSocket.
- Simulated RSUs must send the "true" position of every vehicle in the network.
- The Twin supports all message types sent by RSUs.
- The Twin maps vehicles on a SUMO network, to track their movements.
- The Twin must run in parallel with the simulation, without blocking it.
- The Twin must save all data to a database.
- The Twin must be able to launch external tools, for example, to react to DEN events.
- The Twin must be modular, so that it can be reused in other projects.
- A standalone application must be able to analyze the data received and provide a way to visualize it.

Figure 3.1 shows a coarse overview of the logical architecture of this Thesis's work, which can be directly compared with the general structure of Figure 2.3.

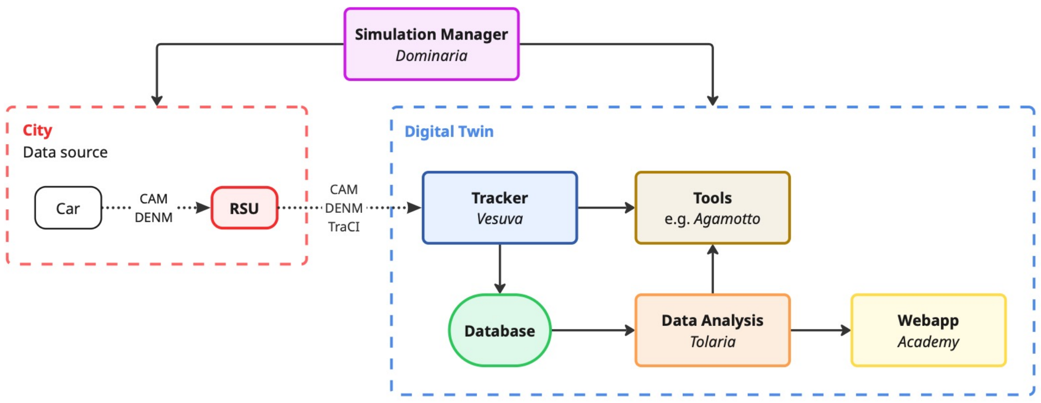


Figure 3.1: Abstract overview of the architecture

The project is divided into two distinct environments, the city and the DT. The virtual urban network sends messages to the Twin thanks to a series of custom OMNeT++/Artery services. Upon arrival, the data is analyzed and then saved in a database for easy access by a custom analysis library. A simple web application provides a frontend to better investigate the results of the simulations. A simulation manager is also included to streamline the process of preparing and launching simulations.

The project is thus divided into the following components:

- **Data source**, the (simulated) city
- ***Vesuva***, the vehicle tracker and core of the twin
- ***Tolaria***, the data analysis library
- ***Academy***, the data visualization frontend
- ***Dominaria***, the simulation manager

Since high performance was not a requirement, the language of choice for the project was Python, which has the best version of the library available to work with SUMO networks: `libsumo`.

There was no constraint on the kind of database to be used. Periodic information sent by vehicles could be seen as time-series data, and as such, an optimized database like InfluxDB or AWS Timestream could be appropriate. However, this project had needs that go beyond the ones that most dedicated time-series databases can fulfill. Hence, the choice fell on a more general-purpose, self-hostable solution: MongoDB.

3.2 Data source

This Thesis focuses on vehicles capable of sending Cooperative Awareness Messages, and wants to study the quality of the information exchanged by recording them through Roadside Units. Artery already includes a service that models the basic functionality of CA both for vehicles and RSUs. The matter becomes how to make that data readily available to an external entity.

Thinking about deployment in a real environment, one can imagine two scenarios:

- Centralized model, with all RSUs sending data to a single system.
- Distributed model, where either single RSUs or a federation of nodes all host part of the digital twin service.

The simulation setting lends itself well to both of these approaches: the unified environment allows no latency in the aggregation of data; at the same time, entities are all modeled individually and correctly separate concerns and areas of influence, so a distributed pattern is perfectly feasible. Since some pre-processing from the side of RSUs can greatly improve the efficiency of the system, by acting as a filter for unimportant data, the solution chosen for this Thesis is a middle ground between the two models (shown in Figure 3.2). Data from each RSUs is streamed to the external entity, with some logic left in Roadside Units themselves. Building a true distributed model would have been outside the scope of the project, but it is a notable future extension opportunity.

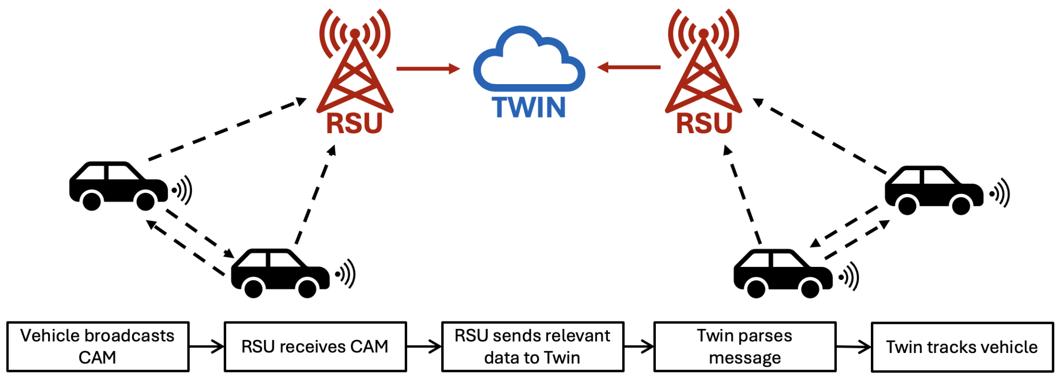


Figure 3.2: Overview of the communication between the entities

The desired features can be implemented following two different approaches: extending the existing Artery services to stream data to an external entity, or creating a new scenario that includes all the required components and services. The first solution has several drawbacks, including the need to reapply all changes on a new Artery release and a lack of portability. Therefore, the second approach was chosen, bundling all new features into a new self-contained scenario with all custom components.

3.2.1 Components

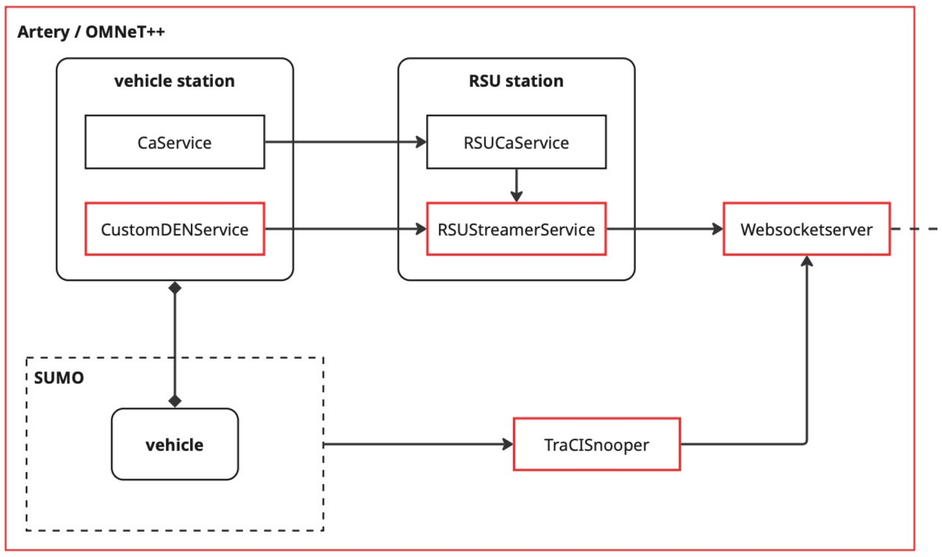


Figure 3.3: Components used in the Artery simulation

The component that streams data externally is a custom service called **WebSocketServer**. As the name suggests, it manages a WebSocket and handles client connections to which it sends all data provided by the other components.

Artery already features services for Cooperative Awareness, one for vehicles (CaService) and one for RSUs (RsuCaService). Whilst the first can be used as-is, since it already generates the messages in the form we need (following the CAM standard), the latter had to be enhanced, as it needed to call the WebSocket service to stream all the data it gathers. Following the rationale discussed in the previous Section, instead of modifying the existing service, the choice was taken to create a new component that complements RsuCaService and extends it with the required functionality.

The new service, called **RsuStreamerService**, subscribes to the signals emitted by RsuCaService to capture all CAM received, forwarding them to the server. Additionally, each time a CAM is sent by the RSU itself, that message also needs to be sent outside, to notify about the geographic position and status of the Unit. The WebSocketServer does not send messages as they are; instead, a JSON message is constructed to send only the information of interest to the Twin.

Running simulations in Artery, which has a SUMO environment attached, presents the bonus of being able to easily stream, alongside the messages captured by the RSUs, the real movement data from all the vehicles. Thanks to OMNeT++'s feature of component hierarchy traversal, which allows direct access between modules, the TraCI interface controlling the urban simulation is available to all services. By using that, after every SUMO step, all vehicle movements and updates can be collected and sent outside. The data can then be used to compare the accuracy of the inferred data, using only one connection and making sure that the basis for comparison is reliable. A service to gather this data is thus required, and is modeled in the **TraciSnooper** component.

Introducing support for DENM functionality in RSUs is not straightforward. Since DENM communication usually does not involve Roadside Units, Artery does not include a DenService for RSUs, only for vehicles. To avoid creating yet another service, the ability to receive DEN messages is included in the already presented RsuStreamerService as a small technical debt to speed up development. Forwarding all received messages as they are would have resulted in too many streams, most of which duplicates. This is due to the nature of DEN: disseminated messages linger for some time in the proximity of the event that generated them. Moreover, Artery's DenService does not provide a parameter, like other services, to limit the frequency of message generation.

The first problem was solved by introducing a memory system in RsuStreamerService to avoid sending duplicates, akin to the one already featured in vehicles. By only sending messages with increasing IDs, it is possible to completely prevent repetition and save bandwidth. A simple extension of the DenService for vehicles called **CustomDenService** was then introduced to limit the DEN generation frequency by adding a parameter modifiable in the omnetpp.ini file.

To summarize, the new components introduced in the scenario are:

- **WebSocketServer**: made available to all services to send data through a WebSocket.
- **RsuStreamerService**: subscribes to signals emitted by RsuCaService to forward all received messages. It also acts as the DenService for the RSUs.
- **TraciSnooper**: gathers all movement data from vehicles after each SUMO step and sends it through the websocket.
- **CustomDenService**: simple extension to the existing DenService for vehicles, to limit DEN message frequency.

3.2.2 Scenario

The complete scenario outlined in Figure 3.3 has the folder structure shown in Figure 3.4. At the root level, the following configuration files are present:

- **omnetpp.ini**: the main configuration for the OMNeT++ simulation, as explained in Section 2.3.2.
- **services.xml**: definition of the services mounted on simulated vehicles.
- **rsuservices.xml**: definition of the services that are mounted on simulated RSUs. Both “services.xml” and “rsuservices.xml” are referenced in the main configuration file, omnetpp.ini.
- **WorldWithStreamer.ned**: the definition of the network. It extends the default “World” network with the WebSocket Server and the TraciSnooper component, both of which are thus registered at the highest level of the component hierarchy.
- **CMakeLists.txt**: the build configuration file for the scenario, which compiles all custom services and links them to the Artery libraries.

A service, to be correctly compiled and usable in the simulation, needs, alongside the C++ source and header, an .ned file to register it in OMNeT++. It then needs to be included in the CMakeLists.txt file.

```
custom_scenario/
├── rsu ........................ RSUs to be spawned in the network
│   └── RSU.xml
├── network ............................. Example SUMO scenario
│   ├── routes.rou.xml
│   ├── example.net.xml
│   ├── example.sumocfg
│   └── trips.xml
├── services.....................................Custom services
│   ├── websocketutility
│   │   ├── MessageStreamerWebSocket.cc .h .ned
│   │   └── MessageStreamerWebSocketServer.cc .h .ned
│   ├── CustomDenService.cc .h .ned
│   ├── RsuStreamerService.cc .h .ned
│   └── TraciSnooper.cc .h .ned
├── ................................. Artery Configuration files
├── omnetpp.ini
├── services.xml
├── rsuservices.xml
├── WorldWithStreamer.ned
└── CMakeLists.txt
```

Figure 3.4: Folder structure of the custom Artery scenario

3.3 *Vesuva*

The name "Vesuva" refers to the main component at the core of the whole Digital Twin, with the actual tracking on the virtual map done by an internal module. It was designed to be a reusable piece of software that could support other applications. To underline the modularity of the system, it defines some interfaces to describe how it interacts with storage, concurrent execution of tools, and a GUI. Vesuva then implements those interfaces to support the needed use cases.

Since receiving messages and saving data to the database are I/O procedures that, if not handled well, can introduce latency and even data loss, a significant effort was put into preserving the asynchronicity of all components.

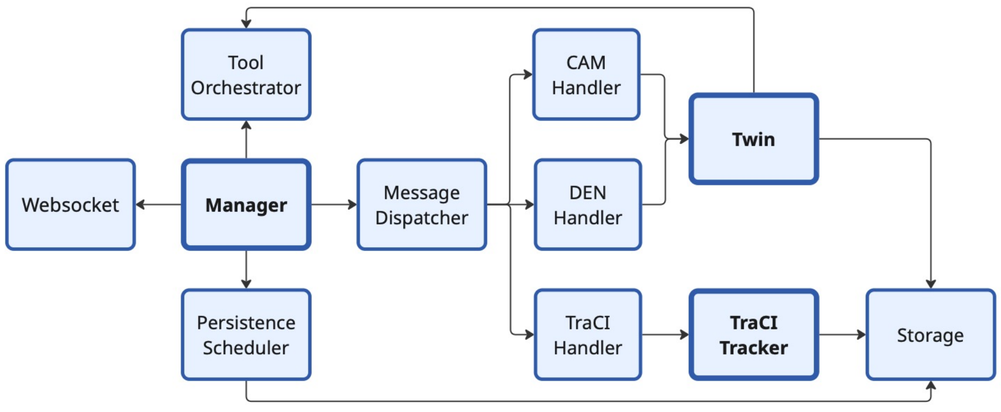


Figure 3.5: Vesuva architecture

As shown in Figure 3.5, the most important components are the **Twin** (the CAM tracking module), the **TraCITracker** (that handles the source of truth messages captured by the simulation) and the **Manager** (which starts all other components, manages their life cycle and constructs the communication hierarchy). Data from the WebSocket is automatically handled by the correct entity associated with their data type. A **Persistence Scheduler** component backs up data every minute to prevent losing data due to crashes. Lastly, the **Tool Orchestrator** makes it easy to launch other tools (even external scripts) in the background.

In the following sections, all components will be analyzed in-depth, in the order that best reflects the flow of data.

3.3.1 Messages

A **WebSocket** client is responsible for the communication with the data source. To avoid blocking the reception of messages when they need to be processed or other blocking operations have priority, instead of a callback approach in which the client itself notifies the Twin for new messages, a push/pull model was preferred. The WebSocket puts all messages it receives in a shared `asyncio` library's Queue. If the connection to the server is lost or canceled, the client periodically tries to reconnect.

Data is read from the Queue by the Manager and directly passed to the **MessageDispatcher** class. This simple component holds a collection of Handlers associated with a data type, along with a method that, given a message, calls the appropriate handler.

```python
class MessageDispatcher:
    def __init__(self):
        self.handlers = {}

    def register_handler(self, message_type: str, handler):
        """Register a handler for a specific message type"""
        self.handlers[message_type.upper()] = handler

    async def dispatch(self, message: Dict[str, Any]):
        """Route incoming messages to the appropriate
           ↪ handler"""
        message_type = message.get('type', 'missing').upper()

        if message_type in self.handlers:
            await self.handlers[message_type].handle(message)
```

Listing 3.1: src/vesuva/dispatcher.py

Handlers are specializations of the abstract class ***MessageHandler***. Four handlers are defined: **CAMHandler**, **RSUHandler**, **DENHandler**, **TraCIHandler**.

```python
class MessageHandler(ABC):

    @abstractmethod
    async def handle(self, message: Dict[str, Any]) -> None:
        """Process the message"""
```

Listing 3.2: src/vesuva/handlers.py

Each handler simply calls the correct method in either the Twin or TraCITracker component.

Separation of concerns is almost exasperated, and as the project stands, this implementation choice could be seen as overkill: messages are simply forwarded to the correct function. However, the philosophy of this work is to be as modular and as extensible as possible, and the combination of a MessageDispatcher class that manages some Message Handlers is a great forward-looking solution. It allows, for example, the easy introduction of a parser if messages change in structure.

3.3.2 Twin

The Twin is the most important component in the entirety of the Vesuva project, as it is responsible for tracking the vehicles on the virtual map. As already touched upon in Section 3.1, it was designed with reusability in mind. The requisites were:

- Be agnostic to the data source attached.
- Support Cooperative Awareness Messages.
- Support Decentralized Environmental Notification Messages.
- Map vehicles on a SUMO network.
- Launch external tools while the Twin is running.
- Interface with any storage solution and graphical application.

CAM

The Twin manages a virtual map, formed by a collection of “Features”. At the base, there is a SUMO network that includes the area of interest covered by the RSUs. Features are an extension of SUMO edges and junctions, preloaded and analyzed to improve vehicle tracking.

Cooperative Awareness Messages already include the position of the vehicle at each timestamp, but that information is not enough. This Thesis needs to analyze traffic in a way akin to how underground induction loops count vehicles. This means that each vehicle needs to be accurately tracked on the streets of the city.

Map Matching is a problem tackled by many researchers [47], but out of the scope of this work since the process, for the task at hand, can be facilitated by a small amount of “cheating” on the premise of the complete agnosticism of the Twin. Vehicle GPS information, in fact, is impossibly accurate in this scenario, since a simulated environment is supplying that data. SUMO already provides a method to locate the nearest edge in a network from a given position. The combination of these two facts means that by matching the network both in the data source and in the Twin, the process of reverse geocoding (that is, finding features from coordinates) is trivial.

The `libsumo` library provides the methods `getNeighboringEdges()` and `getNeighboringLanes()`. Tracking on the specific lanes that form an edge is an effort that would be hard in the event of a real scenario, so even if the task at hand benefits from high precision, it is better to settle for just edge mapping. However, the `getNeighboringEdge()` method searches in a given radius, ordering results by the distance from the middle segment of the edge; if a wide edge, with many lanes, is next to a narrow edge, vehicles in the outer-most lane of the wide edge would be incorrectly assigned to the narrow edge, since it is closer in distance. An example of this phenomenon is given in Figure 3.6.

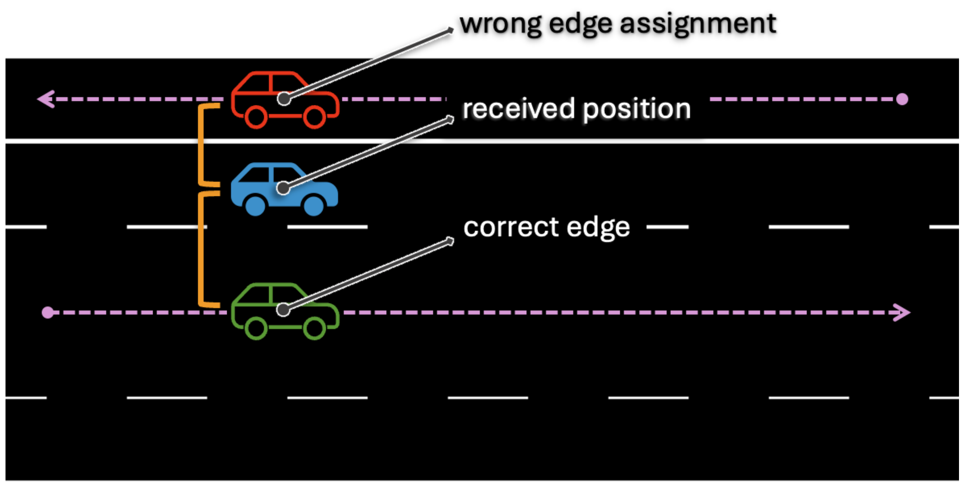


Figure 3.6: Since the vehicle is in the outermost lane of the wide edge, it is closer to the small edge than to the middle segment of the wide edge. This would lead to an incorrect assignment of the vehicle to the small edge

Searching by lane is thus much more precise, because lanes are generally the same size. The edge is then easy to find since all lanes include a reference to the one they belong to.

Vesuva tracks vehicles by assigning each CA message to a **LocationUpdate** object. This class holds all information assigned to the notification of movement, along with the ID of the RSU that received the message. The Twin ensures that duplicates are discarded and that messages forwarded by multiple Roadside Units are merged. This happens if the radio range of two RSUs overlap.

LocationUpdate searches for features in the vicinity of the sent point. First, coordinates need to be translated from longitude and latitude to the correct projection on the SUMO map. Then, lanes are searched with a radius that increases until matches are found. Vesuva correctly identifies vehicles on junctions by searching for internal edges too (Section 2.3.3). This process is sped up by the augmented virtual map briefly discussed at the beginning of this Section: internal edges are not directly available with `libsumo`, since some are actually computed at runtime if not defined in the network configuration file; so, on Twin initialization, a Map is created with all those features pre-computed.

Vesuva models the entities it tracks as **Vehicle** objects, which keep the list of all LocationUpdates in a “route”. Since there is no way to know when a vehicle leaves the network, either because it arrives at its destination or because it never crosses a RSU again, periodically, vehicle routes are flushed to storage. Keeping track of the whole route could be useful if heuristics were to be introduced for map matching, but as it stands, there is no real reason to keep the entire route, as the last Update is enough to detect duplicates.

RSU

Another duty of the Twin is to keep track of the passing of time. The moment a message is received can be greatly distant from when it was actually sent. Unfortunately, as discussed in Chapter 2.1, Cooperative Awareness does not provide a straightforward way to know the precise timestamp of its messages. Vesuva takes advantage of the fact that RSUs send CA messages at a fixed rate to track the passage of time. By doing so, RSUs become a sort of clock that tracks when an “overflow” happens (remembering that CA messages send timestamps in modulo 2^{16}). This has the added bonus of providing simple support for simulated environments, since there is no need to synchronize time between the Twin and the Simulation. SUMO and OMNeT++ can run at the pace that is needed for the complexity of the simulation, but timestamps will still be correctly tracked (also thanks to OMNeT++’s internal clock).

RSUs are assigned to LocationUpdates, but since CA messages from Roadside Units are periodic, on startup, some vehicles may already be streaming when no information is known about the RSUs in the network. Also, RSUs could be added while the simulation is running, if testing more dynamic scenarios. When a message arrives from an “unknown” RSU, it is saved in a list of pending messages to be resolved once information about the Unit arrives.

RSUs are the core of this Thesis, which aims to test their role as more than just a Cooperative Awareness augmentation entity. Using just CA messages is a novel approach in the field, which usually sees RSUs equipped with cameras [13] or LiDARs [14]. But relying solely on RSUs, of course, has some drawbacks. The problem of finding the best placement for this type of entity is an ongoing research topic [16], but it is realistic to assume that a complete coverage of a large area would be a rarity, especially when the technology is still being tested. In a setting in which 100% of the vehicles are equipped with the required communication stack, the area covered by a RSU radio range would yield 100% accurate results, and 0% about the immediate surroundings. This would make each of these areas a sort of separate, smaller map in which all information is available. This is the real source of data for traffic modeling, since in the rest of the network there is no certainty about the vehicles' movements. This remains true even in a more realistic scenario, with a much lower penetration of intelligent vehicles.

Each RSUs is not an independent entity in the Digital Twin, but rather a part of a larger system. RSUs with overlapping radio ranges could share information, since together they cover a larger area and could be considered as a single entity. With a sufficiently dense grid, the size and the quality of the information of the actual covered map results larger that the sum of the individual parts.

Vesuva models the areas that have information available as "Observed Zones". They are an abstraction of RSUs, simply portions of the network in which vehicles are sensed; they can be formed by more than one RSU, if they overlap.

The **ObservedZone** class keeps a list of all RSUs that are part of the zone, and provides the logic to test if new ones can be added. To understand if two RSUs overlap is a matter of querying (using `libsumo`) the radio range of each RSU, and checking if there are edges in common. A distinction is made between internal edges (which are fully covered) and fringe edges (those that lead in or out of the zone, only partially in the radio range). In Figure 3.7, fringe edges are orange and internal edges are green.

Vesuva recognizes internal edges by checking the connections (incoming and outgoing) of each edge, and checking if all connections of a given edge are also in the radio range of the RSU. If so, the edge is considered internal. A fringe edge has at least one connection that is not visible by the RSU. When an RSU is added to an ObservedZone, all fringe edges are computed again, while the list of internal edges can simply be merged. If two zones share a fringe edge (pointed at in Figure 3.7), that can be considered internal when the areas are joined, since even if not fully covered, all of its connections are now covered.

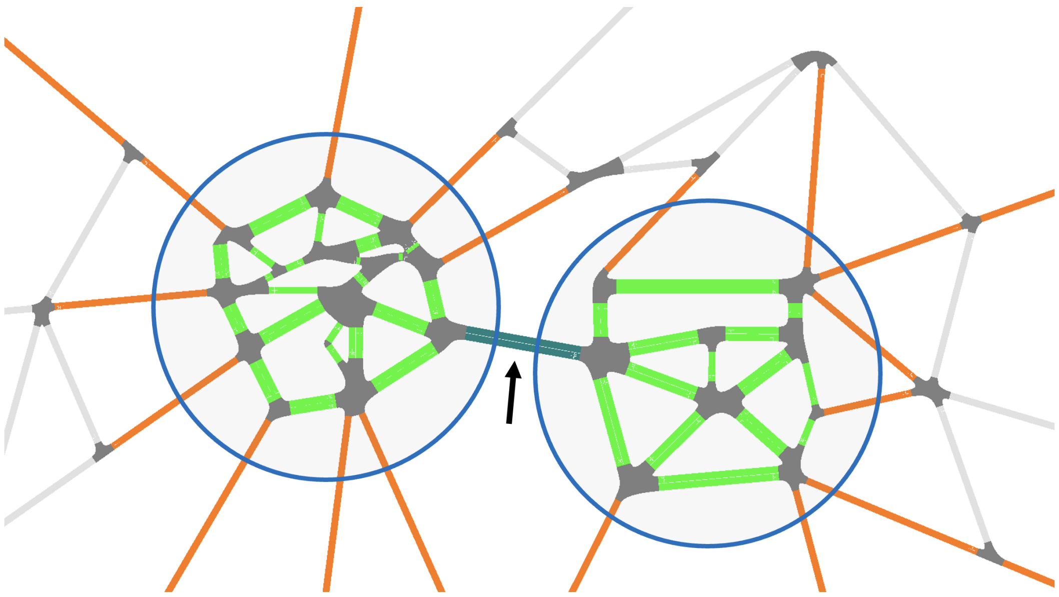


Figure 3.7: In green, the internal edges, in orange, the fringe edges. While merging the two RSUs into one observed zone, the shared fringe edge will be considered internal

DENM

The last task the Twin is responsible for is handling DEN messages. When an event is happening in the network, a large number of messages are generated, captured by the RSU, and sent to Vesuva. To showcase action in the Twin, that is, demonstrate that the application provides a framework for decision-making, Vesuva, after a certain amount of DEN messages, launches an external tool called Agamotto.

Agamotto (which is fully showcased in its own Section - 3.6 - at the end of this chapter) is a Python tool to test the impact of road closures by simulating in parallel many scenarios of traffic redirection. It launches a series of SUMO simulations, gathers data, and presents the results to find out, according to the chosen metric (traffic, pollution, noise, etc.), the best strategy to redirect vehicles when a road is closed. To work, it needs a complete SUMO configuration, together with some routes; to compute these, all data gathered by the running Vesuva instance can be used.

The tool of choice to pair with Agamotto was Jtrrouter, selected among those presented in Section 2.3.3. First, flows and turn probabilities are computed, then saved in temporary .xml files to be passed to the router. When the tool is done running, Agamotto is launched.

The sequence of

1. Creating flows.xml file,
2. Creating turns.xml file,
3. Generating routes with Jtrrouter,
4. Writing the SUMO config file,
5. Launching Agamotto

all happens in the background while Vesuva is running, thanks to the ToolOrchestrator class. Output from the external scripts is even redirected to the same display interface that Vesuva uses, so that it can be shown on screen if a GUI is attached.

3.3.3 Orchestrator

The Orchestrator class is a powerful tool for the execution of background activities. It is a fully modular and independent component that can be used in any project that needs to run external scripts or Python coroutines in the background. It was designed to handle either Python coroutines or external scripts, and even a mix of both in sequence.

All tasks are assigned an ID to track and control their life cycle independently. A Task object has an optional "dependencies" field that can be used to chain tasks together by providing the list of task IDs it depends on, so that a task is only executed after all its dependencies have been completed. This ensures that the order of execution is respected, without the need to manually manage it. Once tasks are registered, the Orchestrator oversees their execution in the background, while the main thread is Agamotto.

The external script task is the most complex one. In addition to the command to launch, it accepts the definition of the working directory and eventual environment variables. All process output is captured using an `asyncio` subprocess Pipe, which redirects standard input ("stdin") and standard output ("stdout") to a coroutine that reads all data and forwards it to any compatible display, according to an interface.

All tasks have a status to better monitor their life cycle:

- **Pending**: initialized but waiting to be started
- **Running**: started and currently executing.
- **Completed**: done running and exited with success.
- **Failed**: execution terminated due to an error.
- **Canceled**: execution halted by the user.

3.3.4 Storage

As briefly touched upon in Section 3.3.2, Vesuva periodically commits vehicle data to storage. This happens when vehicles have not received updates for some time, and according to a scheduled action. The PersistenceScheduler class hosts a task that periodically gets hold of the data lock, triggers a save, then lets execution resume. Both CAM and TraCI information is saved, to make sure that recent data is always available on the database.

In Section 3.1, it was mentioned that MongoDB is the database of choice for the project. The Twin presents a generic interface for storage, to allow compatibility with different solutions. Vesuva implements the Storage interface to support Mongo. Being a document-oriented database, it is a good match with data that partially comes from JSON messages.

All entries are associated with a unique ID representing the current "run". Since the outer layer of Vesuva is aware of the simulated environment, this makes sure that data from different runs is not mixed up. Both CAM and TraCI updates are saved with an `InsertMany` call. The advantage of saving periodically is that it can be done in batches, greatly reducing the amount of DB writes required. RSU details are also saved.

The collections and their fields in the database are:

Runs Run ID, Time of creation, and a description (scenario, CAM penetration, and optional text).

Updates Vehicle ID, Position (lat, lon), SUMO feature (edge or junction with ID), List of RSUs that received the update, Timestamp.

RSUs RSU ID, Run ID, Position (lat, lon), Range.

3.3.5 Manager

The Manager class is the entry point for Vesuva. It handles the instancing and composition of all other components and manages their life cycle. The outer layer of Vesuva is allowed to know that the data source is an Artery environment, hence it is responsible for creating both the Vesuva Twin and the TraciTracker instances. To forward messages to the correct entity, a series of Message Handlers is defined, each with its own callback function and recognized message type. They are registered in a MessageDispatcher class for easier access and greater abstraction.

The Manager handles both simple asynchronous tasks and fully fledged background processes, adding to the complexity of managing the life cycle of all pieces of the system. It is a requirement that some parts of the applications keep running while others wait for operations to complete, for example, receiving messages and saving data to the database are two expensive operations that: a) must not block execution and b) must not lose any data. Since data that is being written to the database cannot change while a save is in progress, analysis of incoming messages needs to be stopped every time a backup is requested. This is possible thanks to a shared lock, that only when released can then be taken control of by any component that needs it, pausing the others.

To summarize, the Manager:

1. Receives simulation settings
2. Initiates the MongoDB connection
3. Creates the Twin instance and the TraciTracker instance
4. Starts a Persistence Scheduler component to periodically back up data
5. Creates the Message Dispatcher and registers all Handlers
6. Creates a queue for incoming messages
7. Starts the WebSocket client
8. Starts a process to read messages from the queue
9. Correctly stops all processes on closure

3.4 *Dominaria*

Dominaria is the simulation manager. It launches and monitors the life cycle of Vesuva instances, the actual Digital Twins. It also provides an interface to configure the simulation environment, such as the SUMO scenario and the penetration rate of intelligent vehicles in the network. Having a GUI is not a requirement, but it greatly helps with the development and testing of the Twin.

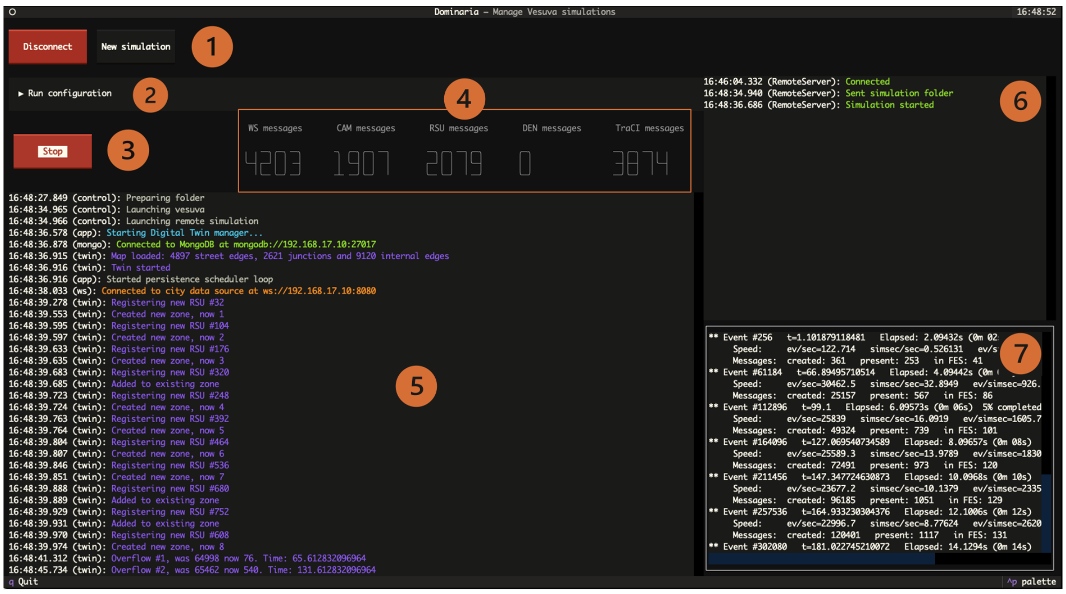
Dominaria implements the display interfaces defined in Vesuva to provide a simple way to visualize the data being processed by the Twin. A robust framework is needed to allow the user to start and stop simulations, configure the environment, and visualize the data being processed. Furthermore, Dominaria needs to display both Vesuva's and the remote Simulation's output to have a complete understanding of the state of the simulation.

To speed up the process of designing the GUI, the `Textual` [48] Python library was used. It allows to create Terminal User Interfaces (TUIs) with a rich set of widgets. It supports a set of advanced features such as asynchronous input and output, which is a requirement for the project since Vesuva needs to run in the background while the user interacts with the interface.

Thanks to the introduction of a series of custom widgets, Dominaria can display the state of the Twin, the current simulation, and all running tools. It also allows starting and stopping Vesuva instances and configuring the simulation environment. A counter for all message types incoming from the simulation is also displayed to provide a quick overview of the amount of data being processed.

The interface shown in Figure 3.8 has the following component:

- **(1)** Buttons for connecting to the remote server and adding a simulation
- **(2)** Simulation configuration form (collapsed)
- **(3)** Simulation control button
- **(4)** Received messages statistics
- **(5)** Vesuva debug
- **(6)** Remote simulation debug
- **(7)** Remote OMNeT++ program output



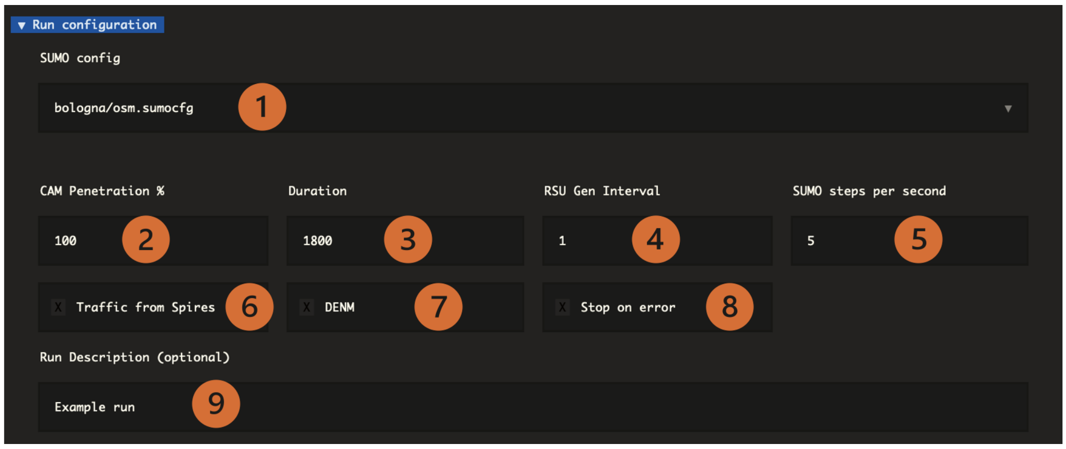
```
WS messages   CAM messages   RSU messages   DEN messages   TraCI messages
4203          1907           2079           0              3874
```

```
16:48:27.849 (control): Preparing folder
16:48:34.965 (control): Launching vesuva
16:48:34.966 (control): Launching remote simulation
16:48:36.578 (app): Starting Digital Twin manager...
16:48:36.878 (mongo): Connected to MongoDB at mongodb://192.168.17.10:27017
16:48:36.915 (twin): Map loaded: 4897 street edges, 2621 junctions and 9120 internal edges
16:48:36.916 (twin): Twin started
16:48:36.916 (app): Started persistence scheduler loop
16:48:38.033 (ws): Connected to city data source at ws://192.168.17.10:8080
16:48:39.278 (twin): Registering new RSU #32
16:48:39.553 (twin): Created new zone, now 1
16:48:39.595 (twin): Registering new RSU #104
16:48:39.597 (twin): Created new zone, now 2
16:48:39.633 (twin): Registering new RSU #176
16:48:39.635 (twin): Created new zone, now 3
16:48:39.683 (twin): Registering new RSU #320
16:48:39.685 (twin): Added to existing zone
16:48:39.723 (twin): Registering new RSU #248
16:48:39.724 (twin): Created new zone, now 4
16:48:39.763 (twin): Registering new RSU #392
16:48:39.764 (twin): Created new zone, now 5
16:48:39.804 (twin): Registering new RSU #464
16:48:39.807 (twin): Created new zone, now 6
16:48:39.846 (twin): Registering new RSU #536
16:48:39.851 (twin): Created new zone, now 7
16:48:39.888 (twin): Registering new RSU #680
16:48:39.889 (twin): Added to existing zone
16:48:39.929 (twin): Registering new RSU #752
16:48:39.931 (twin): Added to existing zone
16:48:39.970 (twin): Registering new RSU #608
16:48:39.974 (twin): Created new zone, now 8
16:48:41.312 (twin): Overflow #1, was 64998 now 76. Time: 65.612832096964
16:48:45.734 (twin): Overflow #2, was 65462 now 540. Time: 131.612832096964
```

```
16:46:04.332 (RemoteServer): Connected
16:48:34.940 (RemoteServer): Sent simulation folder
16:48:36.686 (RemoteServer): Simulation started
```

```
** Event #256   t=1.101879118481   Elapsed: 2.09432s (0m 02
   Speed:     ev/sec=122.714   simsec/sec=0.526131   ev/s
   Messages:  created: 361   present: 253   in FES: 41
** Event #61184   t=66.89495710514   Elapsed: 4.09442s (0m
   Speed:     ev/sec=30462.5   simsec/sec=32.8949   ev/simsec=926.
   Messages:  created: 25157   present: 567   in FES: 86
** Event #112896   t=99.1   Elapsed: 6.09573s (0m 06s)  5% completed
   Speed:     ev/sec=25839   simsec/sec=16.0919   ev/simsec=1605.7
   Messages:  created: 49324   present: 739   in FES: 101
** Event #164096   t=127.069540734589   Elapsed: 8.09657s (0m 08s)
   Speed:     ev/sec=25589.3   simsec/sec=13.9789   ev/simsec=1830
   Messages:  created: 72491   present: 973   in FES: 120
** Event #211456   t=147.347724630873   Elapsed: 10.0968s (0m 10s)
   Speed:     ev/sec=23677.2   simsec/sec=10.1379   ev/simsec=2335
   Messages:  created: 96185   present: 1051   in FES: 129
** Event #257536   t=164.933230304376   Elapsed: 12.1006s (0m 12s)
   Speed:     ev/sec=22996.7   simsec/sec=8.77624   ev/simsec=2620
   Messages:  created: 120401   present: 1117   in FES: 131
** Event #302080   t=181.022745210072   Elapsed: 14.1294s (0m 14s)
```

Figure 3.8: Dominaria's TUI while a simulation is running



SUMO config: bologna/osm.sumocfg

CAM Penetration %	Duration	RSU Gen Interval	SUMO steps per second
100	1800	1	5
X Traffic from Spires	X DENM	X Stop on error	

Run Description (optional): Example run

Figure 3.9: Simulation configuration form in Dominaria's TUI

The configuration form, shown in Figure 3.9, once open reveals the following options:

(1) SUMO config file selector

(2) Penetration of the CA Service in simulated vehicles

(3) Duration of the simulations in seconds

(4) Seconds between RSU CAM generation

(5) SUMO steps in a second

(6) Toggle to generate a scenario from Bologna's official induction loop data

(7) Toggle to enable DEN events
(8) Toggle to gracefully stop the simulation on any WebSocket error
(9) Optional text description for the run

Dominaria oversees all operations needed to launch both the Artery simulation and Vesuva.

Artery requires a scenario folder, with the required files explained in Section 2.3.4. Dominaria first generates this and then moves it to the correct location. The base structure of the scenario is the one showcased in Section 3.2.2, with some changes made according to the values in the configuration form of Figure 3.9:

- the SUMO scenario is the one chosen in the form dropdown.
- the penetration rate of the CA service is set by adding a filter to services.xml definition, as shown in Listing 3.3.
- the presence of the DEN service is governed by the same type of filter as the point above, modeling True and False values as penetration 100 and 0. To actually generate adverse conditions in the network, a "storyboard" is also copied.
- the omnetpp.ini file sees the simulation time and SUMO config path changed to the requested values. The generation interval for RSUs messages is also set here.
- if the "Traffic from Detectors" option is selected, the SUMO scenario's route file is generated from scratch instead of being copied over from the original scenario, as later explained in its own paragraph 3.4.
- in the sumo config, the step-length parameter is set as 1 over the number of steps-per-second inserted

```xml
<service type="artery.application.CaService" name="CA">
   <listener port="2001" />
   <filters>
      <penetration rate="[PENETRATION]" />
   </filters>
</service>
```

Listing 3.3: src/scenario/services.xml

Artery may be running on a remote server. To copy all required files from the local folder to the remote one, an SSH connection is opened with the `paramiko` Python library. Copying files over SSH/SCP works even if the target host is the same as the one from which the files are being sent, so every configuration is supported, even deployment on the same machine. After the complete folder is sent, Vesuva is launched. Once it is ready, a command is issued, via SSH, to the machine hosting Artery. Output is set to be automatically redirected to the display by continuously reading from the buffer and appending lines to a `Textual` widget (**(7)** in Figure 3.8). Vesuva needs to start before Artery so as not to lose any message.

While the program is running, the tally of all processed messages is displayed. There are five fields:

- Messages received by the WebSocket. This number increases even when Vesuva is not processing messages, thanks to the Queue mechanism explained in Section 3.3.1.
- Vehicle CAMs.
- RSU CAMs.
- Vehicle DENMs.
- TraCI location messages.

Traffic from Detectors While many tools exist to convert real urban networks to their digital counterpart, traffic modeling is arguably the most complex task when using SUMO, as huge amounts of data and adequate algorithms are needed. This Thesis aims to address this problem by utilizing real-time data from vehicles on the road, but alternative approaches are also viable. For instance, many cities have underground induction loops to count vehicles, and some make this data available to anyone, though most with some delay (e.g., updating it every week or month). For example, the OpenData portal of the city of Bologna [12] provides an extensive API to retrieve information about many aspects of the urban administration, including hourly counts of the detectors spread throughout the city, usually updating the portal on a monthly basis.

To run Vesuva with a realistic flow of traffic and test its accuracy against a real source, Dominaria is capable of generating routes from Bologna's induction loops' data. This is an operation that can be greatly facilitated by using a preexisting and tested program. SUMO is bundled with **flowrouter.py** [49], which takes as input the position of the detectors and their total, and outputs a "routes" file and a "flow" file that tries to match the requirements.

To retrieve the data from Bologna's API, generate the required input files, and call the program, the Orchestrator class comes in handy, as despite being blocking, all operations need to happen asynchronously to preserve the responsiveness of the TUI.

First, a series of calls are made to the API. Data is paged in rows of 100, so multiple fetches are needed to retrieve all information. Then, a file containing the position of the detectors is generated. To do so, the position is projected to the SUMO map of the chosen scenario, and then, since SUMO models induction loops lane-wise, for each detector in the API, multiple virtual loops are positioned. The vehicle count is distributed so that the total matches the real data. Detectors are saved in a detector.xml file, and the counts in a flows.csv file. These are passed as input to the `flowrouter` command, along with the time windows that need to be considered. Since Bologna's Open Data saves hourly sums of the detected vehicles, flowrouter is informed that the data should be organized in groups of 60 minutes.

The results are two files, ready to be saved in the scenario folder to be sent to Artery. One is a routes.xml file with the definition of a series of trips vehicles can take (as a list of edges). The other is a flows.xml file that defines the actual flow of vehicles that will inhabit the network (number of vehicles to be generated in a given time frame, taking a certain route).

3.5 *Tolaria*

The final element of the Digital Twin's architecture is Tolaria, the data analysis library. It is a Python package that pulls from the MongoDB database and provides a set of tools to analyze the data generated by Vesuva and Dominaria. It is the key to understanding the impact of the penetration rate of intelligent vehicles in the network, and to evaluating the quality of the information exchanged by RSUs. It is accompanied by Academy, a simple web application that provides a graphical interface to visualize the data processed by Tolaria.

Tolaria is thus divided into three parts: the analysis layer, the frontend layer, and a simple API to interface the two.

3.5.1 Analysis

Vesuva makes it possible to generate traffic models from the information on the number of vehicles counted. Tolaria can compute, from that data, two main metrics:

- The number of vehicles in the network, and their distribution.
- The turn probability for each junction.

Combining density information with the probability of a vehicle taking a certain turn can already yield a good model of traffic, since the hierarchy of roads is respected and simulated vehicles follow realistic patterns. Turn probability is, in any case, computed directly from the edge-counting data, and as such, it is as accurate as the other metric is. The peculiarity of the proposed Digital Twin solution is that tracking the specific entity is straightforward, as CA messages themselves already contain identification of the vehicle that is sending it. This feature greatly helps with the analysis. It would not be available using Radars while it would be very complex to obtain only relying on computer vision.

The operations of extracting density and turn probabilities can be performed at the same time, since they both require the same data. The first step is to gather all routes for all vehicles in the network, and to compute how many are in each edge. This is done by iterating over all LocationUpdate objects in the database and counting how many times each edge is visited by a vehicle. To help gather Update data from the database, a custom MongoDB query is used. Mongo has a useful feature to aggregate data into “views”. They allow the creation of a virtual collection that can be queried as if it were a normal collection, but with the data already aggregated. A route view is created that contains all LocationUpdates for each vehicle and the edges they visited. This allows for the quick computation of the number of cars on each edge.

Density is computed for all edges covered by each RSU’s radio range. The number of vehicles in each edge is then divided by the total number of cars in the area to obtain a percentage-based metric of density. This is done because it is the fairest way to compare the data from the two sources: CAM and TraCI. The former is available only for a selection of streets, while the latter covers the whole network. By providing a normalized metric, it is possible to compare the two sources and understand how much information is lost when the penetration rate of intelligent vehicles decreases.

The turn probabilities are computed by iterating over all routes and counting how many vehicles take each edge when at an intersection. This is also a percentage, but of just the vehicles that reach the junction.

Thanks to the data visualization web application Academy, showcased in the following Section (3.5.2), it is possible to visualize the routes of all vehicles. It is immediately noticeable how many edges, despite being traversed by many vehicles, cannot be counted using just the basic approach, for the sake of the above analysis. This is because, since location updates are sent periodically, if a vehicle is moving at a sufficient speed and edges are short enough, the vehicle may not send any update while traversing that edge, effectively skipping it.

Some heuristics can be applied to mitigate this problem, such as assuming that if a vehicle visited the edge before and after, it must have traversed the one in the middle too. This approach can be applied to junctions too (an edge between two junctions is always traversed) and many other combinations. The process of extending the amount of inferred data could introduce some errors, so for this Thesis, it was decided to limit the depth of research of skipped edges to just one level.

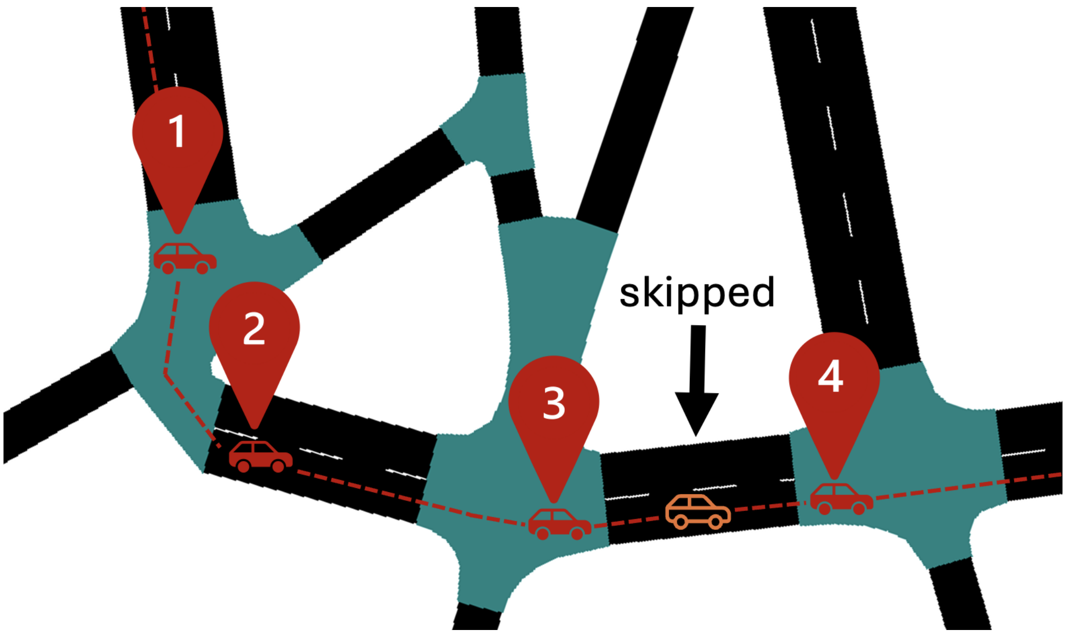


Figure 3.10: The edge between update #3 and #4 is skipped, since the vehicle did not send any update. It still can be inferred to have been traversed, since the vehicle was on the edge before and after

3.5.2 *Academy*

Academy is a simple web application that provides a graphical interface to visualize the data processed by Tolaria. It features an interactive map on which all data gathered by Vesuva can be visualized.

The app is built using the `deck.gl` library, which allows the creation of a stack of layers on top of a map. Each layer can be configured to display different data, such as the density of vehicles in each edge or the turn probability for each junction. A simple selector allows the user to choose the simulation run for which to display the data. The available data, fetched with the Tolaria API, is:

- List of all RSUs in the network, with their radio range.
- Routes of all vehicles, both from CAM and TraCI.
- Density of vehicles in each edge covered by RSUs.
- Turn probability for each junction covered by RSUs.

The RSUs are shown as circles on the map, with a radius accurate to their radio range (or better, the average range of vehicles that can reach it). Routes are shown as a collection of points (the Location Updates) connected by lines, with a random color for each vehicle. Density is shown as a column at the center of each edge, with a height proportional to the number or percentage of vehicles. The columns for CAM and TraCI are at a slight offset, so that they can be compared side by side. Turn probability is shown as an arc connecting pairs of edges at a junction, with a width proportional to the probability of taking that turn. The arcs are colored according to the probability, with a gradient from black (low probability) to orange (high probability).

Deck.gl allows a definition of a tooltip for each layer, so that when hovering over a feature, the user can see more information about it. For example, hovering over a RSU shows its ID, position and radio range; hovering over a vehicle update shows the ID, visited edge, coordinates and timestamp; hovering over a density column shows the number of vehicles and the percentage of the total; hovering over a turn probability arc shows the probability of taking that turn.

Vehicles can be filtered by their ID, so that only the updates for a specific vehicle are shown on the map. This is useful to track a specific vehicle and see its route, or to compare the data from CAM and TraCI for that vehicle.

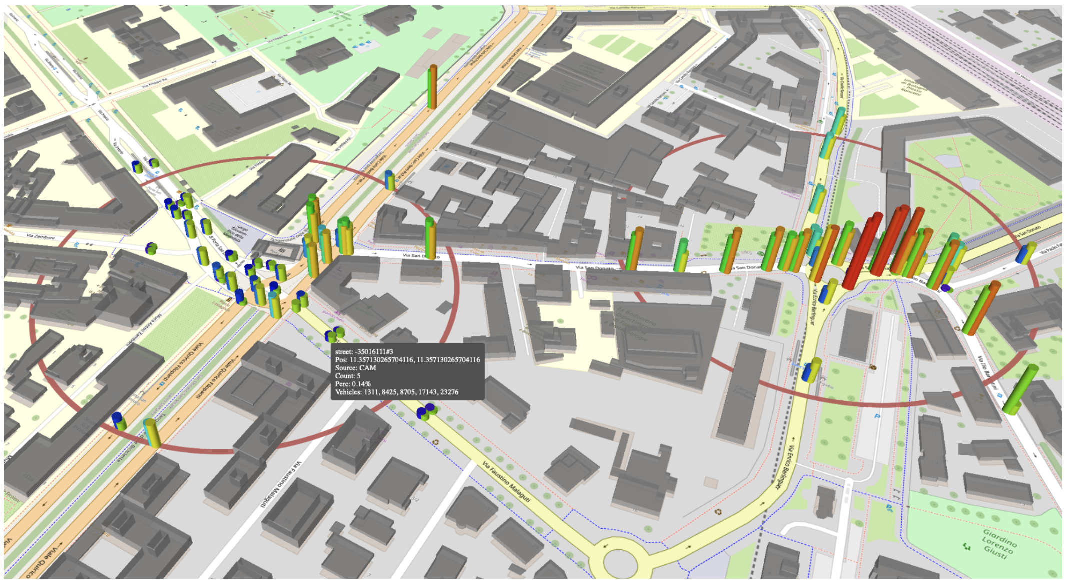

```
street: -35016111#3
Pos: 11.357130265704116, 11.357130265704116
Source: CAM
Count: 5
Perc: 0.14%
Vehicles: 1311, 8425, 8705, 17143, 23276
```

Figure 3.11: Screenshot of the Academy web application showing density data on each edge

3.6 *Agamotto*

Agamotto is a Python tool developed to assist with the simulation of urban environments with the purpose of better understanding how to manage traffic when one or more roads are closed, by generating and running multiple what-if scenarios of traffic redirection. Agamotto is a standalone tool that is not designed to be used solely in Vesuva; rather, it is an example of a Vesuva-compatible tool that can be launched in the background while the Twin is running. In the context of this Thesis, it showcases the potential for the DT to make decisions based on the data obtained.

SUMO is used to test several variations of the same scenario, with Agamotto acting as a middleware that automatically prepares and launches simulations and then gathers and presents all relevant data. Since the number of simulations needed to test all possible combinations of configurations of a scenario can increase rapidly depending on the settings, Agamotto distributes runs on available cores to parallelize execution. Runs are compared based on diverse statistics, including traffic density, vehicle emissions, or mean duration of routes, all metrics that can then be plotted as human-readable output like heatmaps.

The direct interaction with SUMO means that all its configuration options are supported. Additional settings are available to tune Agamotto's specific behavior and test strategies. For example, many roads can be closed at a time, and weights can be specified to control how likely a driver knows, before reaching a junction, that their route is affected.

Overview

The effect of different strategies of traffic redirection is evaluated by running many different environments and comparing them. Since the number of combinations to try can be very large, the first optimization strategy is to execute simulations in parallel. After deciding on the scenarios and distributing them on the available threads, simulations can start. They execute in batches to reduce the overhead of launching more SUMO instances than necessary. When all simulations have completed, the results are collected.

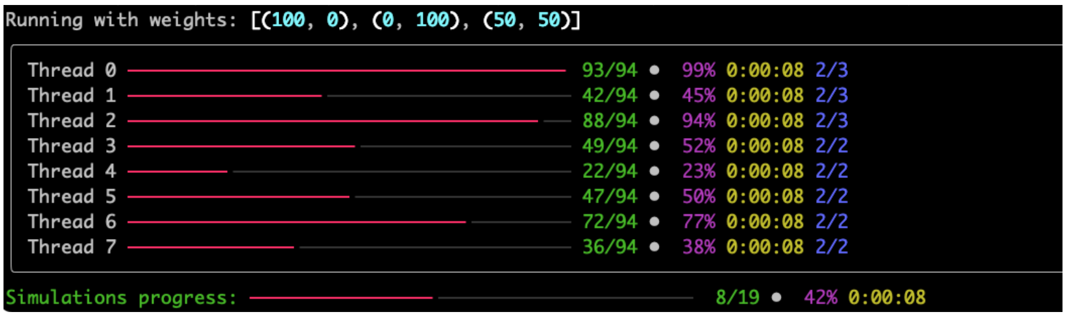
When roads are closed in everyday life, alternative routes are suggested (or imposed) to drivers. They include the roads that are immediately reachable from the junction that vehicles are currently at when they find the closure. In Agamotto, a list of edges can be marked as closed at the same time. Scenarios are then constructed according to that list by looking at all edges that are connected to them. To interact with the network, two tools, the usual suspects, are available: `traci`, to send commands while a simulation is running, and `sumolib`, to analyze the network offline.

To simulate traffic redirection, for each closed edge, alternatives need to be found for every road that is connected to it. Since this operation must be done before simulations are run, `sumolib` directly analyses the network from the corresponding file. For each closed edge, the function `get_options()` is called. This operation finds the list of the incoming connections (an edge leading into the closed edge) and for each of them:

- If there are alternatives to passing through a closed edge, all of them are marked as possible options (if they are not already).
- If only closed edges are available, the connection itself is marked as closed, and the function is recursively called with the updated list.

Once this simple algorithm has produced the complete list of options, all possible combinations of redirections are taken by computing their Cartesian product. Each of these combinations is put into a dictionary containing the "origin" node (a closed edge connection) and "destination" node (the redirection to take to avoid any closed edge).

Agamotto models two strategies that can be applied to each environment: when a road is closed in the real world, some drivers either know in advance (perhaps because they read it on the newspaper or are using a navigation app) or are made aware when they are in its proximity, thanks to a sign or a policeman. Virtual drivers are distributed between these two categories by a weight system that assigns a given percentage of vehicles to each strategy. A list of weight pairs can be provided, and a scenario is run for every one of them. Figure 3.12 shows Agamotto running and the progress of the different concurrent simulations.



```
Running with weights: [(100, 0), (0, 100), (50, 50)]

  Thread 0 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 93/94 •  99% 0:00:08 2/3
  Thread 1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 42/94 •  45% 0:00:08 2/3
  Thread 2 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 88/94 •  94% 0:00:08 2/3
  Thread 3 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 49/94 •  52% 0:00:08 2/2
  Thread 4 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 22/94 •  23% 0:00:08 2/2
  Thread 5 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 47/94 •  50% 0:00:08 2/2
  Thread 6 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 72/94 •  77% 0:00:08 2/2
  Thread 7 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 36/94 •  38% 0:00:08 2/2

Simulations progress: ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 8/19 •  42% 0:00:08
```

Figure 3.12: Agamotto running in the terminal

SUMO simulations in Python are launched and interacted with using the `traci` Python library. Since starting the application requires some overhead, instead of running a new instance for each simulation, TraCI can be used to switch scenarios on the fly, allowing a simulation to be restarted on an already running instance.

Batches share the SUMO config (thus the network) and simulation options. Additionally, each simulation requires the following parameters:

- IDs to recognize the thread and current run.
- List of closed edges (which are kept the same among all runs).
- Specific environment settings, like weights and combinations.
- The folder where to save all simulation output.

Output is either directly collected by SUMO and printed in files in the specified folder (granular data for all vehicles, such as emissions and routes) or collected by Agamotto itself. A directory for each scenario is created inside the run folder, with its index for the name.

Commands can be issued between `traci` requests of steps in the SUMO simulation. Before the first step happens, closed edges need to be marked as such. To do so, they are set as disallowed for vehicles with the 'custom1' class. SUMO handles vehicle access with a class system that, beyond the default ones, includes two custom classes that are available to developers. 'Custom1' is arbitrarily chosen as the class that indicates vehicles that cannot pass through a closed edge. Figure 3.13 shows an example of a scenario running in SUMO-GUI. Red edges are closed, green ones are "origin", while blue edges are "destination". We have already established that two strategies are available for "communicating" to drivers that a road closure is happening. At the beginning of each step, all vehicles currently in the simulation are queried using `traci` as a list of IDs. For each of them, their route is taken, and if it starts from a closed edge, they are removed from the simulation to simplify the scenario. If they are kept in the simulation, a strategy is randomly chosen between the two according to the scenarios' weights, then they are assigned the "custom1" class. Since vehicles have a predefined route when they enter the simulation, classes do not have an effect until a reroute is requested.

If the chosen strategy is *navigation* (drivers know a priori about the road closure), rerouting is performed until a correct path is found. It is an iterative process, as redirections need to be enforced directly by a `setVia()` command, and adding them one at a time can lead to loops. If there is no way to enforce the combination for a given vehicle, it is removed from the simulation. The number of removed vehicles is then available in the output of the program.

If the strategy is *sign* (drivers find out about the road closure by looking at a sign when in proximity), they are added to a list and acted upon at a later stage, to optimize the program, since they can all be checked at the same time. To know if a vehicle is near a closed edge, all "origin" edges in the scenario's combinations are checked for the current list of vehicles passing on them. If a match is found, they are rerouted through a "destination" edge (shown as green in Figure 3.13). Since some combinations may create loops if vehicles cannot reach their destination, each vehicle route is checked for duplicate edges and removed from the simulation if a loop is found.

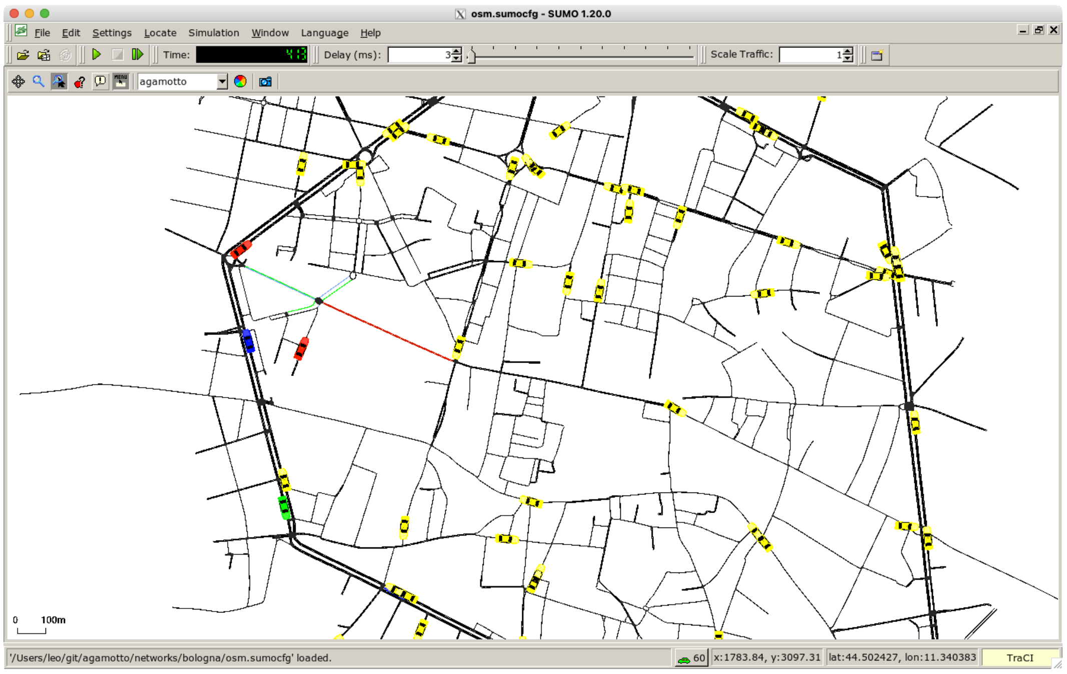

osm.sumocfg - SUMO 1.20.0
File Edit Settings Locate Simulation Window Language Help
Time:
Delay (ms):
Scale Traffic:
agamotto
0 100m
'/Users/leo/git/agamotto/networks/bologna/osm.sumocfg' loaded.
60
x:1783.84, y:3097.31
lat:44.502427, lon:11.340383
TraCI

Figure 3.13: Scenario rendered in SUMO-GUI

Agamotto makes available all output in the form of text files, generated by SUMO and containing details about the single simulation, and images generated by the program itself. The latter are heatmaps and comparison graphs that show the distribution of vehicles in the network or the impact of the redirection strategies according to many metrics (traffic, fuel consumption, emissions, etc.). They are all generated using the `matplotlib` library.

There is no mechanism in Vesuva to close roads in the running simulation automatically. While SUMO can mark streets as closed at run-time, in the real world, that would require human intervention. It is possible, however, to use Vesuva to gather data and then run Agamotto on the same network to test the impact of the redirection strategies. Furthermore, in a fully-fledged and integrated twin of a real city, control over traffic lights and other traffic management systems could be implemented, allowing for data sourced from Agamotto to apply changes to the network in real time.

Chapter 4

Results and findings

This Chapter presents the results obtained from the testing and evaluation of the proposed system, focusing on the accuracy of the Digital Twin in various conditions and highlighting the limitations encountered during density estimation.

4.1 Overview

The proposed system *Vesuva* and its accompanying tools were designed with the goal of creating accurate traffic models. Without delving into the field of Artificial Intelligence, a summary estimate of the characteristics of traffic in the city can be obtained by carefully analyzing the data provided by the application. There are many ways to test the accuracy of such models, and a series of tests can be conducted.

While using simulated data is convenient, it can also lead to wrong conclusions if the environment is not correctly set up. For the majority of tests, the same SUMO scenario was adopted. It is a portion of map just outside the center of the city of Bologna. Around the outermost ring of medieval walls, large roads of two to four lanes host the majority of the traffic on a typical day. It is the perfect testing ground for the system at hand for several reasons:

- The topology is more streamlined compared to the intricate mesh of narrow roads at the very center of the city, while complex enough to distance itself from simple formations, e.g., a highway.

- The administration provides real traffic data collected by induction loops under the roads.
- Those roads are realistic candidates for a real deployment of RSUs, due to their importance and flow of traffic.

The SUMO scenario was generated using the `OSMWebWizard` Python tool. As for the vehicles, despite the veracity of the data being of scarce interest, routes were generated from real induction loop data for the city of Bologna. The locations of the RSUs were chosen arbitrarily, ensuring a mix of highly congested zones, residential areas, groups of very close Units, and a combination of semaphore-controlled crossroads and roundabouts. The areas covered by these RSUs and their IDs can be seen in Figure 4.1.

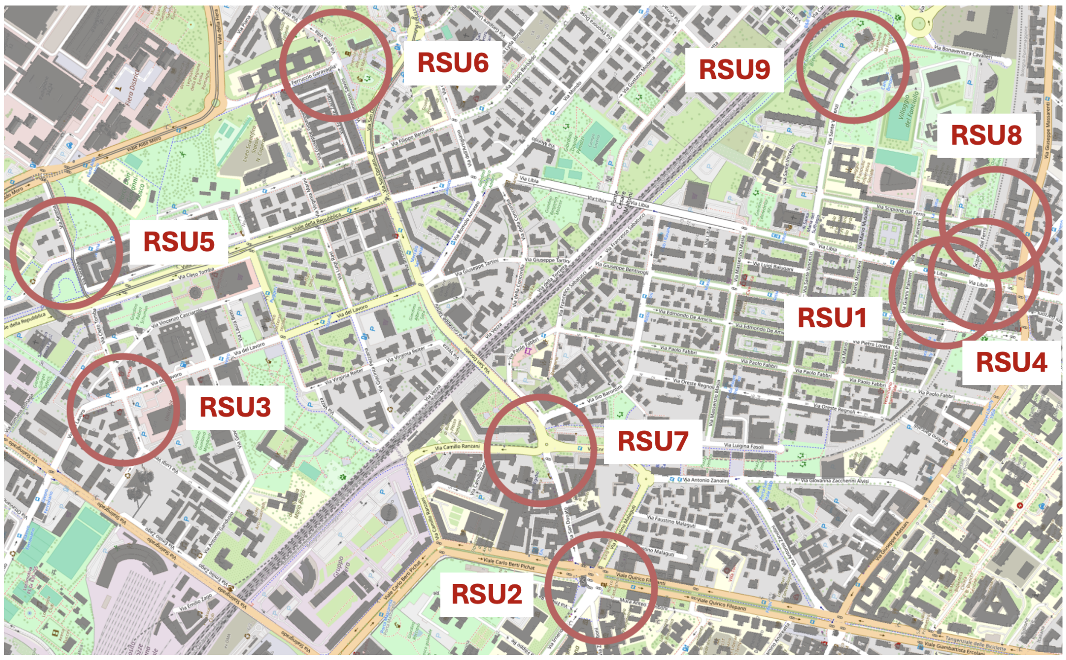


Figure 4.1: Screenshot from the *Academy* web app of some of the RSUs used for testing

Tests were conducted with fewer than two hundred vehicles in a time window of 30 minutes. There is no real need for more vehicles or greater time spans - short periods realistically convey the use case for this kind of Twin: punctual traffic information, as if an induction loop were counting cars passing above it. All simulations were run on a Debian virtual machine with 16GB of RAM and a 12-core CPU, while Vesuva was deployed on a Windows 10 machine with 16GB of RAM and a 4-core Intel i7-7700K CPU.

Slight differences from one run to the other, even with identical starting conditions, can arise from differences in probability-based processes or due to loss of data (either in CAM generation or their transmission through the WebSocket). To reduce variance, tests were conducted in "batches", with each run repeated a minimum of five times, and the average of all their results computed. Comparing runs with the same configuration, it is noticeable that a small percentage of them deviate from the baseline, as some CAM messages are missing compared to other (apparently) identical runs. The average-based approach is thus preferable to have no doubt about the robustness of the data source.

The best metric to understand the accuracy of the tracking, among those that are available given the proposed system, is the density, measured as the number of cars on an edge in a set time frame. Although it is not the sole indicator of a city's traffic situation, Vesuva's measurement, combined with real data from TraCI, makes it a relevant and reliable source of information. Tests aimed to compare how the performance of the Twin changes as the number of vehicles that participate in the communication decreases. The statistical expectation is that the lower the percentage (or "penetration") of cars equipped with the Cooperative Awareness service in the network is, the lower the number of correctly tracked vehicles will be. A baseline of the accuracy of the system is given by comparing the TraCI data with the CAM-sourced data at full penetration of the CA service. Once that is established, tests on a decreasing percentage of communicating vehicles give insight into the feasibility of the solution to create realistic traffic models.

4.2 Test results

In the following comparisons, all figures show three plots. On the x-axis are the IDs of the edges in the range of a given RSU. The first graph is CAM-sourced data, the result of the tracking provided by Vesuva, and the second is TraCI-sourced data, the ground truth taken directly from the SUMO simulation. To better present the differences between the two measurements, on the third plot, the delta between them is drawn, with a value close to zero meaning high fidelity.

Figure 4.2 shows how many vehicles traversed each edge of RSU “#7” in the given time frame. It is a Roadside Unit that oversees more edges than average because it contains a roundabout, which SUMO models as several small tracts connected by just as many junctions. It is also located in a congested area, and indeed, peaks of 60 vehicles are present. At 100% penetration, the expectation is for CAM and TraCI data to be very similar, and this is reflected in the plots.

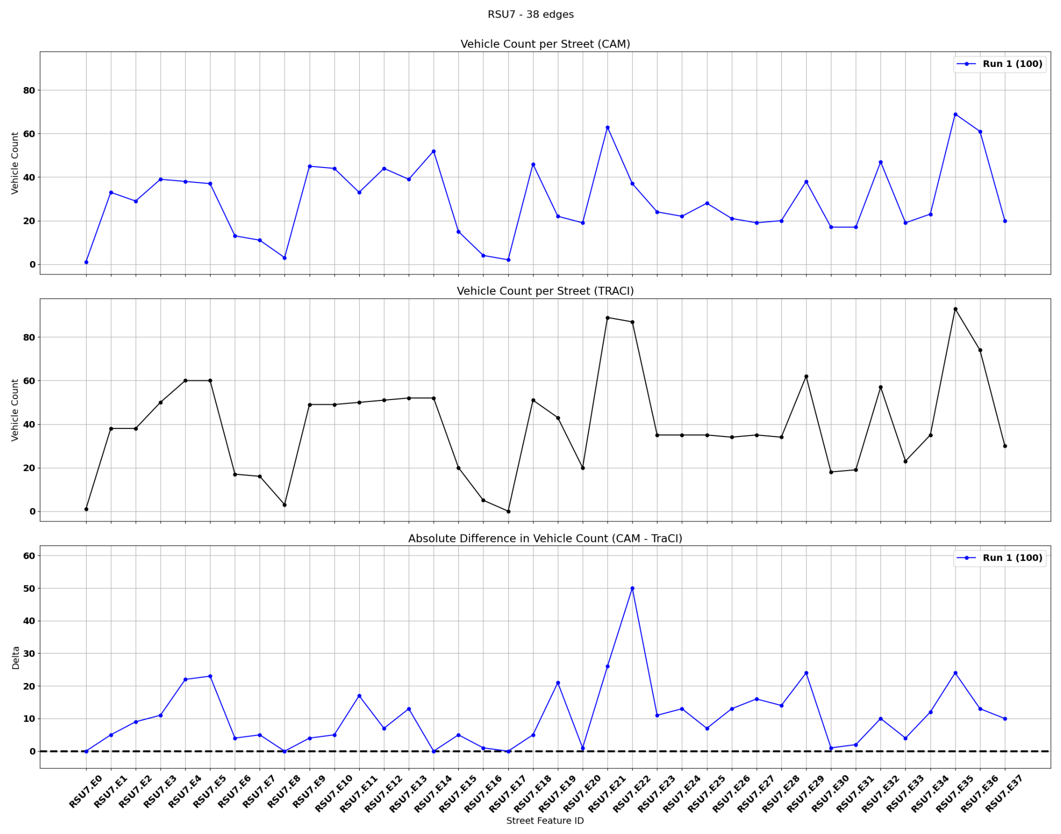


Figure 4.2: Vehicle count RSU#7. Penetration rate of 100% of the Cooperative Awareness service

At a glance, the curves in the first two plots look very similar. The two uppermost plots show on the y-axis the number of vehicles on each edge in the runtime of the simulation. The third chart shows that, on average, there is an absolute difference of 10 to 15 cars between the two sources, so they are not being counted by CAM (TraCI will always count at least the same number of vehicles or more). The impact of this loss of data can be better understood by analyzing the density. It is computed RSU-wise as a percentage of vehicles per edge divided by the total number of all vehicles counted by the RSU. This method provides a relative representation of data that suggests a hierarchy between the streets and can be interpreted as a sort of heatmap of the “most popular” edges in the area covered by the Roadside Unit.

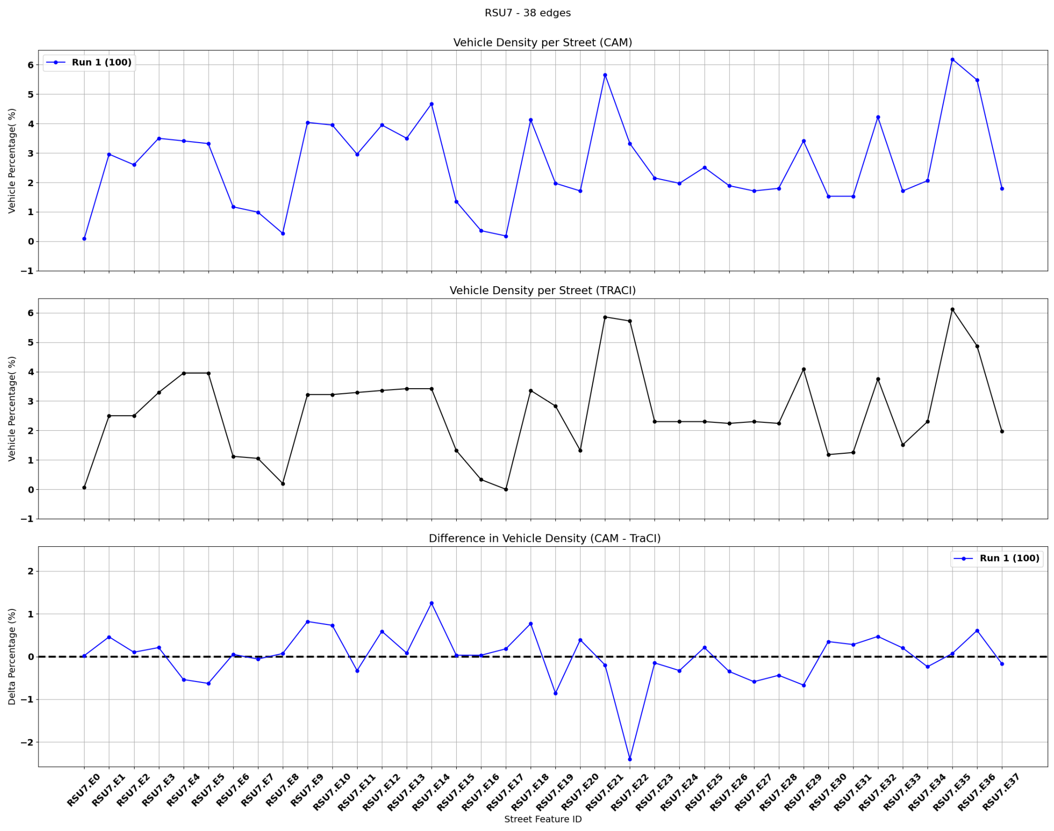
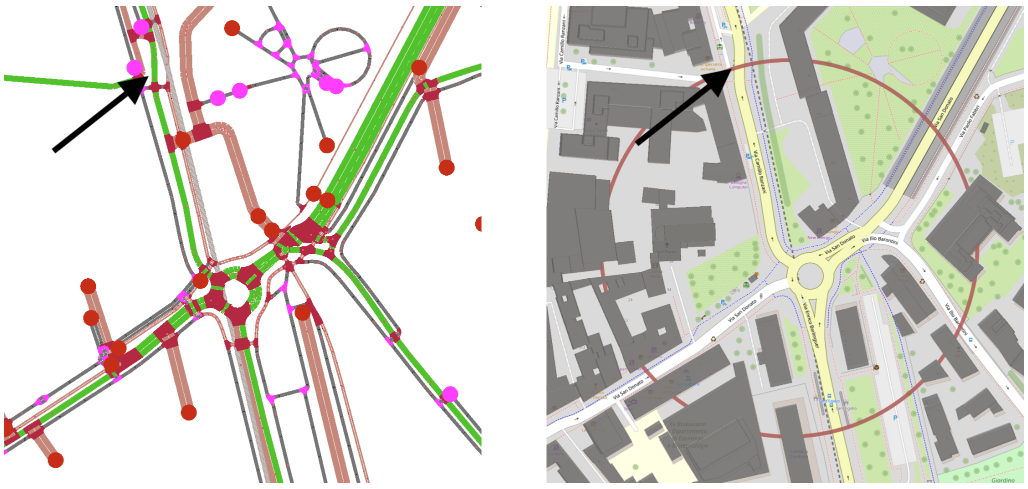


Figure 4.3: Density for RSU#7. Penetration rate of 100% of the Cooperative Awareness service

The comparison plot in Figure 4.3 shows a low difference in percentage for most edges. In all density comparison charts, the difference is not absolute: a negative value means that TraCI has a higher percentage than CAM. And there is one clear outlier, edge “RSU7.E22”. Figure 4.4 shows the RSU topology with a side-by-side view of Tolaria and SUMO, with the edge pointed at with the black arrow being the one with the worst accuracy. If a street is right at the border of the radio range, the likelihood of the RSU capturing less accurate data increases. A simple reason for this is that there is less information to base the inference of visited edges. This could be easily solved by ignoring all the outermost edges, even if partially covered by the RSU, or, a more sensible approach, defining, Unit by Unit, a list of edges to ignore, if they are deemed as not fit for the desired level of precision. Unfortunately, many tests on edge-quality classification proved that this classification would have to be conducted by hand, analyzing the average accuracy of the edges after many example runs. Ordering the results by distance from the RSU, by edge length, by permitted speed, and by number of connections did not delineate a clear trend. It is, however, confirmed that some characteristics of the road are more likely to provide less accurate results, and will be discussed later.



(a) SUMO-GUI (b) Tolaria

Figure 4.4: View of RSU#7's roads, with the least accurate edge indicated

Finally, Figure 4.5 shows the comparison between vehicle counts for four different levels of penetration of the Cooperative Awareness Service: 100%, 90%, 80%, and 70%. The similarities between the plots are a welcome sign - it means that despite having less data to work with, the Digital Twin still has a good view of the situation in the network. Still, some clear differences can be made out. For instance, the line representing 70% penetration in the "delta" plot is clearly above the rest, demonstrating poor accuracy.

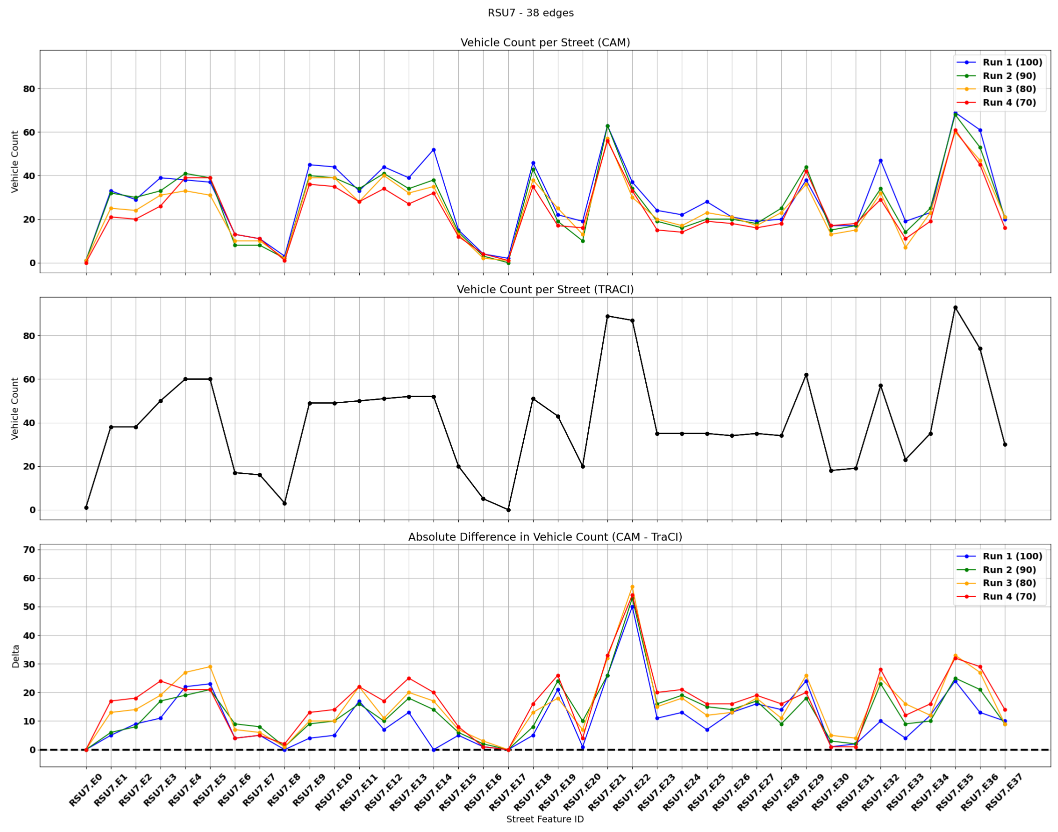


Figure 4.5: Vehicle count for RSU#7. Comparison of decreasing penetration rates of the Cooperative Awareness service

To exclude that an error is causing this outcome and the penetration of the service is correctly reflected in the simulation, tests were conducted with low penetration rates. Figure 4.6 shows rates of 40%, 25%, and 10% compared to 100%, and results are in line with the forecast: low penetration leads to poor accuracy. Nevertheless, the curves still follow the overall trend, and this speaks volumes about the effectiveness of the system.

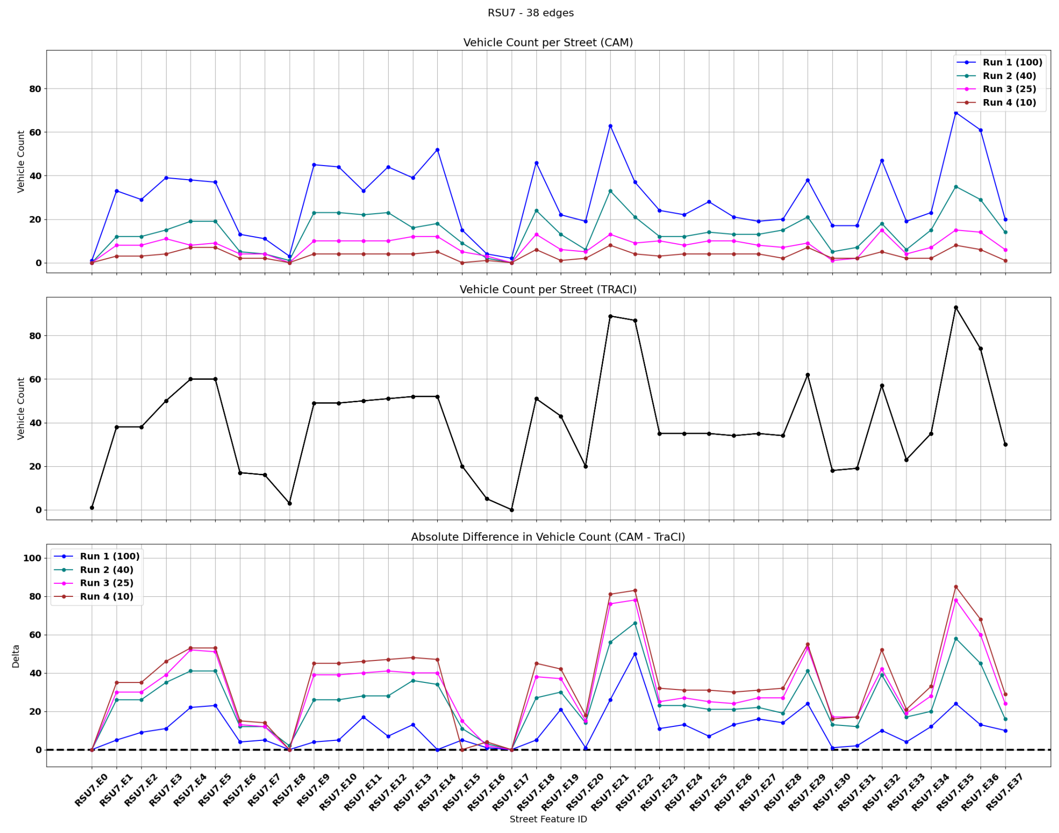


Figure 4.6: Comparison of low penetration rates of the Cooperative Awareness service for RSU#7

The next figures show two more example RSUs. The first, “#6”, is near a simple junction. Figure 4.7 shows the count and 4.8 the density. The other, “#2”, overlooks the most congested junction of the network taken into consideration. Figure 4.9 shows the count and 4.10 the density.

Both demonstrate that while a penetration of 70% is less accurate than higher rates, it still manages to at least capture the trends. Especially Figure 4.7 makes it clear, thanks to the low number of edges, that the expected statistical behavior is reflected.

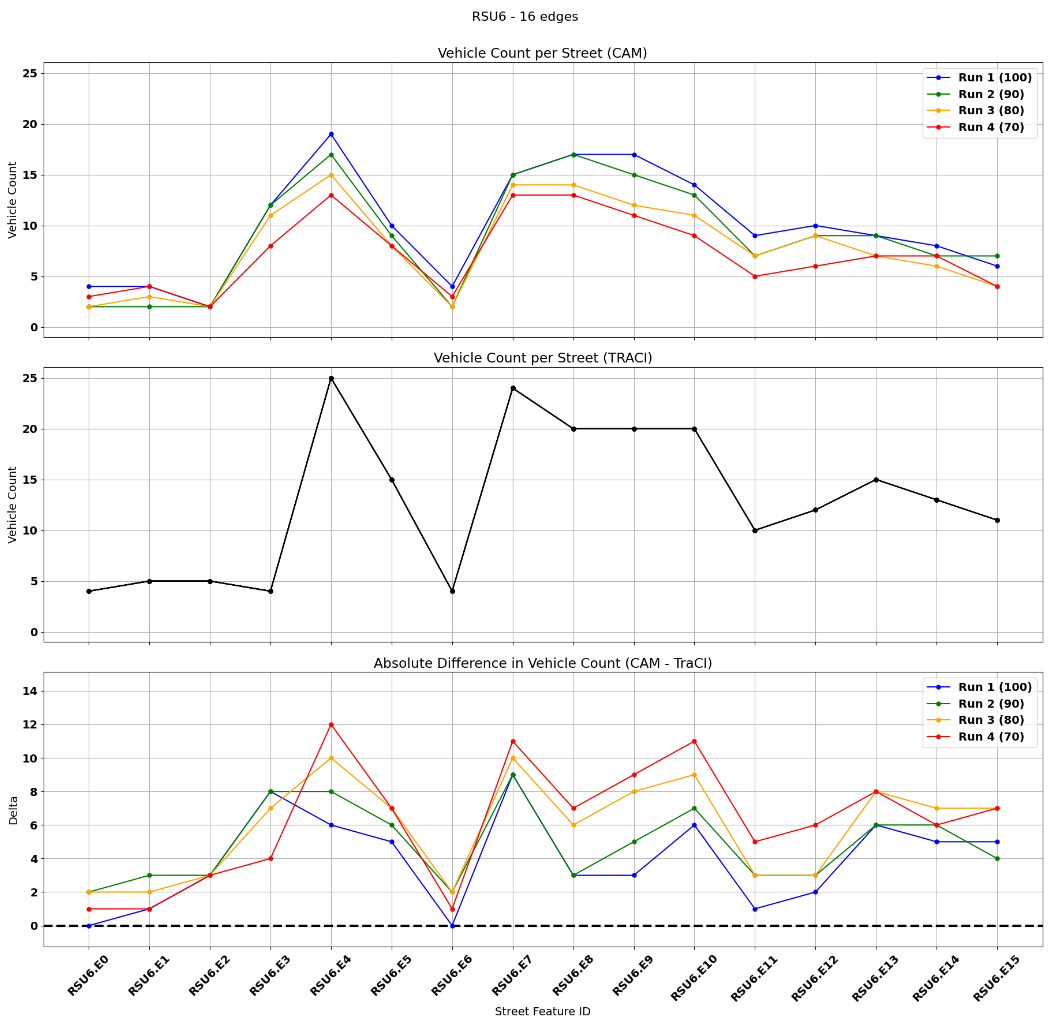


Figure 4.7: Vehicle count for RSU#6. Comparison of decreasing penetration rates of the Cooperative Awareness service

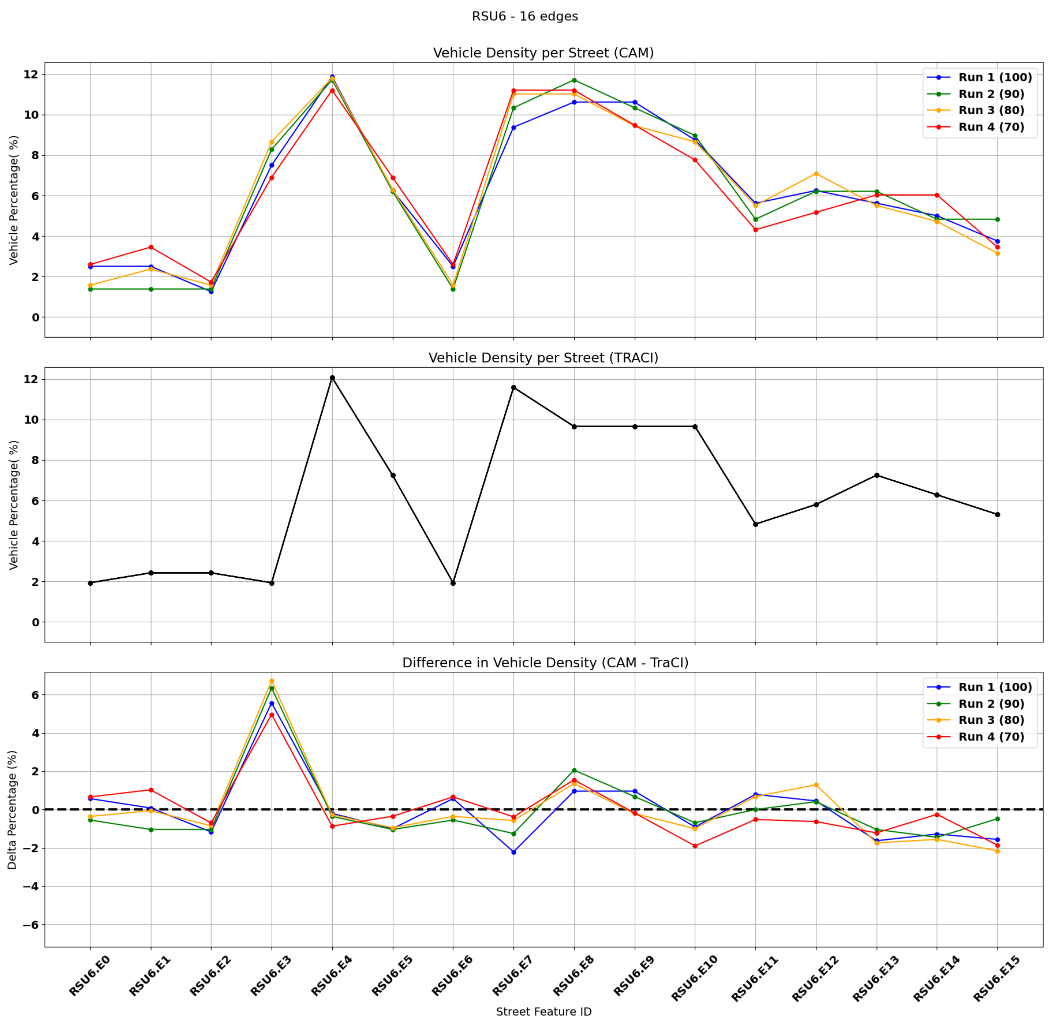


Figure 4.8: Density for RSU#6. Comparison of decreasing penetration rates of the Cooperative Awareness service

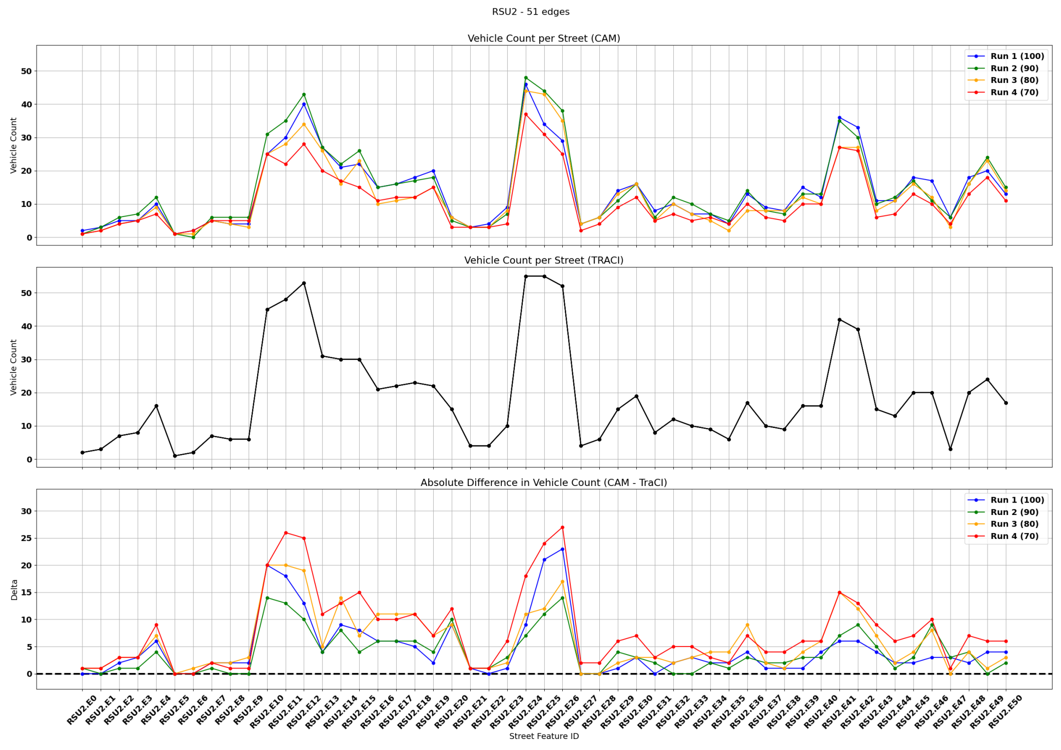


Figure 4.9: Vehicle count for RSU#2. Comparison of decreasing penetration rates of the Cooperative Awareness service

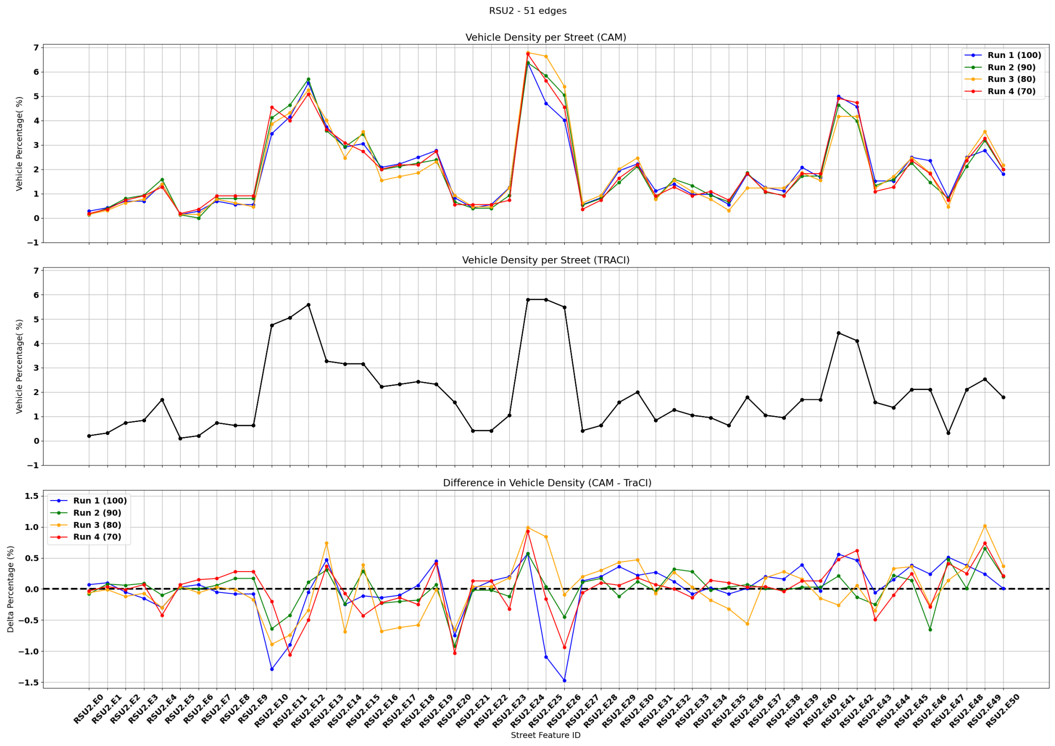
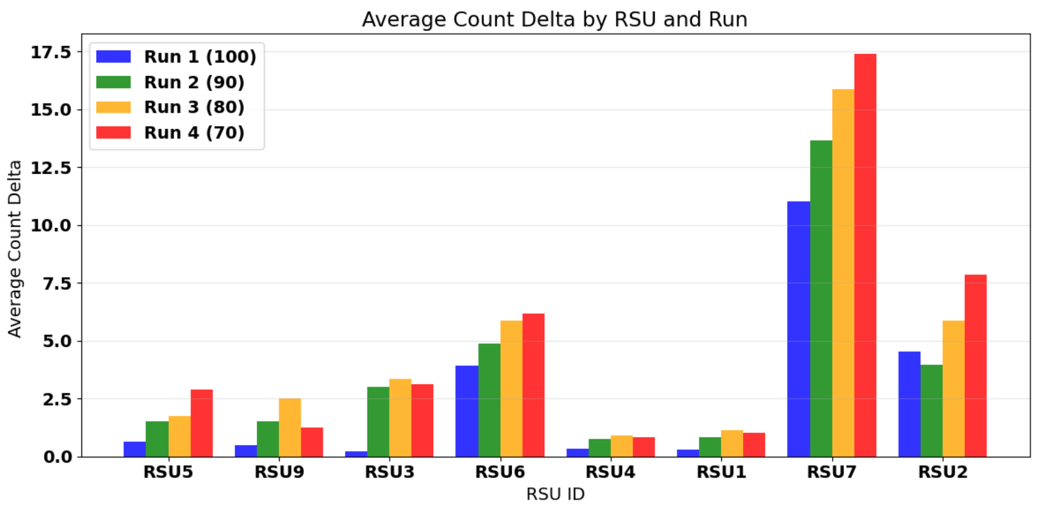
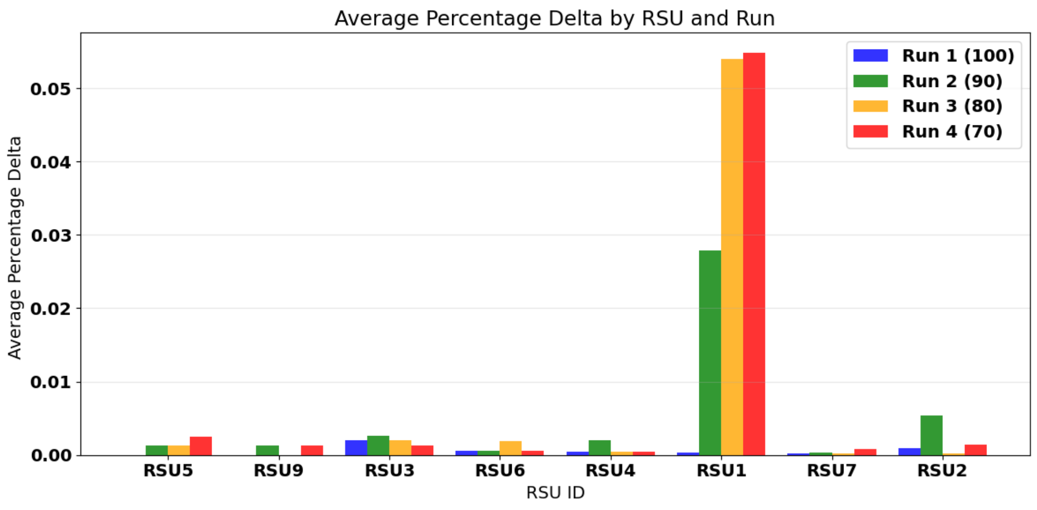


Figure 4.10: Density for RSU#2. Comparison of decreasing penetration rates of the Cooperative Awareness service

To investigate how the characteristics of the roads and the position of the RSUs affect accuracy, Figure 4.11 shows the mean absolute error of all edges for each Unit, ordered on the x-axis by the number of roads they oversee. Plot a) shows the average number of vehicles that are not getting counted by Vesuva, while plot b) shows the average delta between the sensed density and the real one. It is interesting to notice how, despite some RSUs, like #7, showing poor accuracy in the absolute count, density is almost perfect across the board.



(a) Count



(b) Percentage

Figure 4.11: Comparison of average deltas between runs for each RSU

4.3 Discussions

Despite promising results, especially in the comparison of sensed density with real data, there are still some key areas where simple precautions could yield more accurate outcomes. Before discussing those, it is best to consider some peculiarities of the problem at hand.

The first matter is one of network characteristics. This thesis has focused on dense, urban networks - those usually found in city centers. Compared to simpler topologies like highways, city streets are smaller and more interconnected. Buildings surround most roads, deteriorating radio signals both for communication (on which the Digital Twin fully depends) and positioning purposes (GPS information is the main tool used for map matching in this design).

Another problem is that the source of data for the tests was simulated. It is not just a matter of risking having used unrealistic data, as all analysis of the resulting traffic models remains valid since a relative comparison was being drawn, not an evaluation of the correlation with real data. In Section 3.3.2, it was claimed that the accuracy of the data provided to the Twin is unrealistic. It is true that Artery can correctly model, if supplied with sufficient data, the impact of buildings and other causes of transmission degradation. However, since SUMO governs the vehicle simulation itself, the precision obtained is far greater than what can be expected in a real scenario. This contrasts with the first problem listed, rendering it a non-issue in the case of the simulation. However, it is important to keep in mind that all results and conclusions were not based on the real environment.

Even in the presence of a perfect communication environment and a very accurate virtual map, the limitations of the chosen technologies undoubtedly hinder the accuracy that can be reached. In Section 3.5.1, the problem of skipped edges and junctions was explained and shown in Figure 3.10. The way the standard is constructed, there is no feasible way to avoid this at the source, only strategies to mitigate the problem, inferring the missing data from what is available. Despite all these considerations, the value of the results is not to be minimized. The last problem discussed is precisely the focus of the work, which wanted to ask the question of how limited the technology actually is.

The suspicions in actuality were proven to be true: the chosen technology is limited and requires careful use to provide good results. Simple GPS tracking is not sufficient for accurate analysis of traffic. Heuristics to extrapolate more information from what RSUs provide can greatly increase the quality of the data, going beyond simple inference like the one presented in this work. By combining the areas of RSUs close to each other, as detailed in Section 3.3.2, the system has more information to work with to deduce skipped edges. Cooperative Awareness Messages also contain additional useful information to deduce traffic that goes beyond position: for example, a lower mean vehicle speed compared to the norm can indicate that a road is congested.

In the areas where the network topology was more complex, the system showed poorer accuracy in testing. The quality of the virtual map can help improve all operations of map matching, but it is clear that a simple road structure can lead to better results. The radio range did not prove to be a limitation, but, as discussed at the beginning of this Chapter, in the real world, GPS is inevitably less accurate.

The source of data investigated in this Thesis, CA messages, from a Digital Twin standpoint, is better suited for traffic model creation for testing purposes, rather than precise traffic analysis of the city. To accurately count vehicles, an approach based on induction loops or cameras is preferable; tests on low penetration rates showed that the overall trend of traffic is followed even when few vehicles participate in the communication, but it is a metric that, unsurprisingly, is far from accurate from an absolute point of view. However, the ability to also receive event messages is very powerful. Chapter 5 will look at some use cases that better convey the usefulness of the solution in this regard.

Chapter 5

Use cases

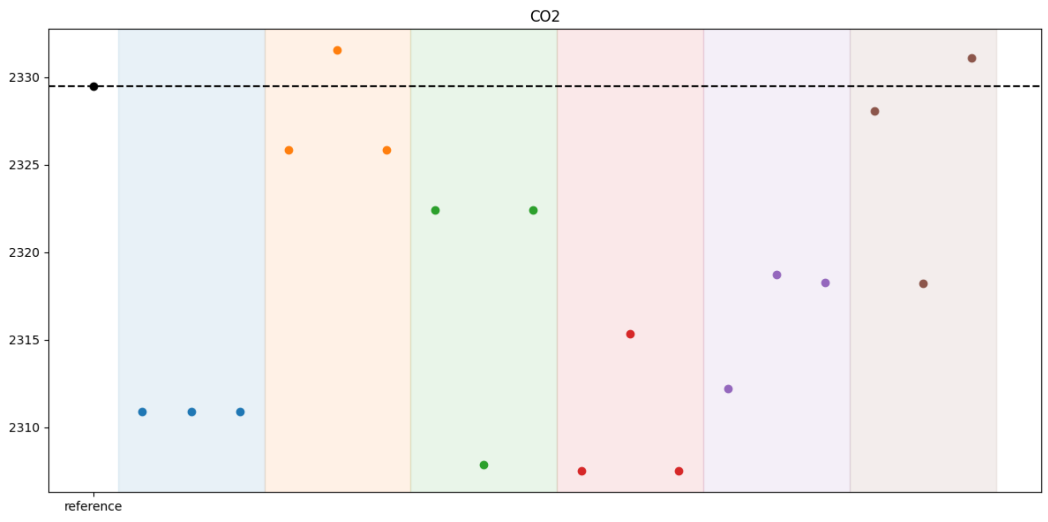
The potentialities of the proposed system can be better grasped by looking at some realistic use cases.

5.1 Assisting in traffic redirection decisions

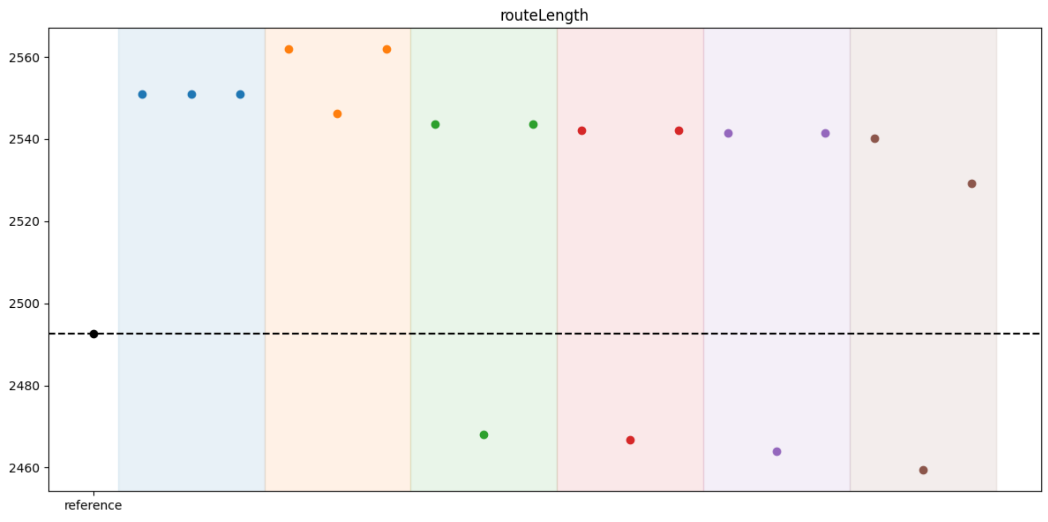
In the general context of Digital Twins, a simulation is only as useful as the features it makes available. Even a perfect virtual copy of a city has little value if the administration cannot use it to test real-world decisions. Vesuva addresses this need by supporting Agamotto, a tool that can act directly on the data that Vesuva gathers. This makes it possible to move from simple monitoring to actively experimenting with scenarios. The most basic use case is to take traffic models generated via Vesuva/Tolaria and use them to test different traffic situations inside Agamotto.

For example, an urban administrator might want to test the effects of closing a street before deciding how to redirect traffic. This is a common need: cities deal with this almost every day, whether due to accidents that require the presence of police, emergency infrastructure repairs, or construction work that changes road layouts for days or weeks.

Agamotto enables decision makers to simulate various strategies and select the one that best aligns with their goals, which can vary depending on the situation. In some cases, the most crucial factor might be reducing the average travel time. In others, it might be minimizing emissions, avoiding congestion near key areas, or keeping public transport on time. Agamotto supports this kind of what-if analysis using only real, up-to-date data collected from the city.



(a) CO2 emissions



(b) Average route length

Figure 5.1: Example result plots for Agamotto

By helping administrations explore different options and compare outcomes (like in Figure 5.1) before acting, this system could make it easier to make informed and thoughtful decisions, not based on guesswork, but on data that accurately reflects the current state of the city.

5.2 Automatic accident detection

The ability of Vesuva to correctly receive and manage Decentralized Environmental Notification Messages brings forth some use cases that showcase the potential of a Digital Twin that can leverage this kind of data. The previous use case asserted that Agamotto is a tool that could prove to be valuable in the day-to-day administration of a city. Its combination with real-time data is powerful, and it is worth being discussed further.

By participating in DENM communication, the DT gains access not only to the usual data, like vehicle positions and speeds, but also to reports of what is actually happening on the road network. This includes events such as accidents, sudden hazards, road work, or traffic congestion. Having this layer of information improves the reliability and relevance of any analysis the Twin performs. It allows the system to confirm events that might otherwise only be guessed from indirect information, or to discover problems that would go entirely unnoticed.

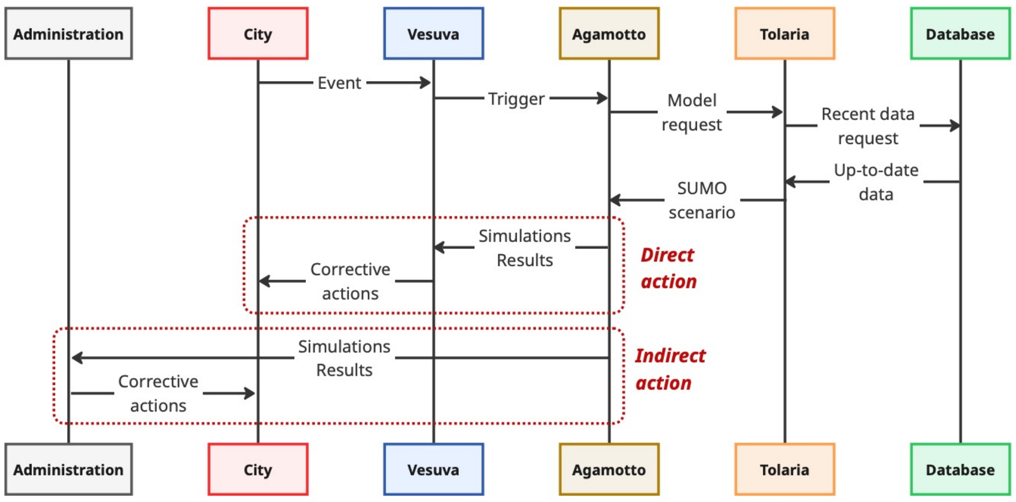


Figure 5.2:]
Example sequence for DT action based on DENM reception

Being quickly informed of emergency situations becomes even more potent if the Digital Twin can act, directly or indirectly, in response. For example, as shown in Figure 5.2, Agamotto could be triggered as soon as the Twin receives a relevant DENM, treating the affected area as a set of temporarily closed roads. This would allow it to automatically simulate redirection strategies and prepare possible responses without delay.

In systems where the Twin is also connected to infrastructure (like traffic lights), it could go a step further and take immediate action, changing signal patterns to ease traffic flow or redirect vehicles. Otherwise, it can still provide a detailed report of the best available strategies to the city administration, allowing them to make the final decision but significantly reducing the time between detecting the issue and responding to it.

5.3 Long-term urban planning

Beyond reacting to day-to-day changes in the city, a Digital Twin like Vesuva also holds potential as a planning tool for long-term infrastructure projects. By simulating proposed changes to the road network, Vesuva and Agamotto can assist administration in making informed decisions before construction even begins.

Typical examples include the addition of a new road, the removal of an intersection, or the creation of limited traffic zones. These are not simple changes, and their effects can propagate across the entire city. Predicting those effects in advance, especially in a realistic and data-driven way, is one of the most valuable contributions a Digital Twin can offer to urban planning, and one of the usual use cases for existing examples in the real world.

In this context, Vesuva provides the baseline by collecting and aggregating real traffic data. Combined with Tolaria's ability to analyze data over time, a detailed picture of the current situation emerges. This picture then becomes the foundation for Agamotto to simulate hypothetical changes, taking advantage of its ability to test many scenarios in parallel. City planners can use this approach to evaluate multiple strategies before committing to one. Each alternative can be tested against all the metrics that Agamotto provides, such as average travel time, congestion levels in key areas, and environmental impact. Over time, this allows for better planning decisions that are backed by measurable outcomes.

Conclusions

This thesis has focused on Intelligent Transport Systems (ITS) and the state of the art of Digital Twins, asking how to bridge the gap between the current adoption of smart vehicles and the level of data quality needed to feed the twins themselves.

An unexplored approach was employed to collect data: instead of mounting sensors and cameras on Roadside Units (RSUs), the solution takes advantage of their role in the communication stack, streaming all messages directly to the twin. By analyzing the obtained Cooperative Awareness Messages (CAMs) and Decentralized Environmental Notification Messages (DENMs)—both part of the European standard for WLAN-based Vehicle-to-Everything (V2X) communication "ITS-G5"—it is possible to correctly track Connected and Automated Vehicles (CAVs).

The design and implementation details for a proof of concept of this novel kind of Digital Twin were presented. The devised system closely resembles the structure of a classic Twin and is composed of the following components: a WebSocket client for receiving messages, a Tracker component for map matching, an asynchronous Storage solution to save all parsed data, and a Scheduler for tools running in the background. The data source was a traffic simulation run in Artery, an application that bundles the widespread frameworks SUMO and OMNeT++. This allowed the collection of actual vehicle movement data, which is convenient for testing the effectiveness of the solution. A component inside the Twin manages that information, which served as the ground truth for all tests.

A data analysis library and a graphical interface for data visualization complete the Digital Twin picture. Simulations are managed by a custom Terminal User Interface, which can control instances running on different hosts.

To showcase the Twin's ability to act on the collected data—and the value of being notified of hazardous events through DENMs—a tool was developed that simulates different scenarios of traffic redirection when a road is closed. Some use cases for the tool in conjunction with the Twin were illustrated.

The vehicle tracking accuracy was tested against decreasing penetration rates of intelligent vehicles in the network, to understand the impact that such a solution would have under low but realistic adoption—ultimately answering the question of how far current technology is from being viable in our cities.

Tests confirmed the statistical expectation: accuracy inevitably decreases when fewer vehicles participate in the communication. However, all results showed that the overall trend is still captured, even with low penetration of the service in the network. A clear hierarchy of the roads in an area covered by an RSU can be delineated by computing a density metric as the number of sensed vehicles on each street over the total. Such information is useful to create local traffic models that can either be employed for simulations or to visualize traffic trends. Some observations on the challenges faced and how to improve accuracy concluded the analysis of the results, asserting that many corrective actions would be needed to reliably use the solution for real traffic measurements, which as it stands is only fit for high-level evaluations.

Acronyms

3GPP 3rd Generation Partnership Project 6

ABS Anti-Lock Braking System 5

ADAS advanced driver assistance system 10

BTP Basic Transport Protocol 20

C-V2X Cellular V2X 7

CA Cooperative Awareness . i, 3, 4, 10, 34, 35, 44, 51, 52, 55, 67, 70, 79

CAM Cooperative Awareness Message i, 3, 9–12, 19, 31, 32, 34–36, 39, 41, 42, 48, 51, 53, 56–58, 67–69, 79, 85

CAV Connected and Automated Vehicle 2, 3, 19, 85

CPM Collective Perception Message 9

CSMA Carrier Sense Multiple Access 7

DEN Decentralized Environmental Notification 4, 12–14, 32, 36, 37, 46, 52

DENM Decentralized Environmental Notification Message i, 3, 4, 9, 12–14, 31, 32, 36, 41, 53, 83, 85, 86

DSRC Dedicated Short-Range Communications 6

DT Digital Twin 2–4, 16, 18, 19, 31, 33, 59, 83

ETSI European Telecommunications Standards Institute 8, 20

GDPR General Data Protection Regulation 17

IoT Internet of Things 14, 16, 17

ITS Intelligent Transport System . . . . i, 2, 3, 5, 8–10, 13, 85

ITS-S ITS-Station 9–14

ITSC Intelligent Transport System Communications 8

LoA Level of Automation 7

MAC Medium Access Control 7

MANET Mobile Ad Hoc Mobile Network 5

NED Network Description 23

RSU Roadside Unit . i, 3, 6, 8–10, 31, 32, 34–37, 42–46, 48, 51–53, 55–58, 66–70, 72, 76, 78, 85, 86

TUI Terminal User Interface 50, 54

URLLC ultra-reliable low-latency communication 7

V2I Vehicle-to-Infrastructure 2, 5, 6

V2N Vehicle-to-Network 2, 6

V2P Vehicle-to-Pedestrian 2, 6

V2V Vehicle-to-Vehicle 2, 5, 6

V2X Vehicle-to-Everything 2, 3, 5–8, 22, 29, 85

VANET Vehicular Ad Hoc Network 2, 5, 7

WAVE Wireless Access in Vehicular Environments 20

List of Figures

List of Tables

Listings

Bibliography

[1] Kathleen Frazer Oswald. “A Brief History of Smart Transportation Infrastructure”. In: *Transfers* 6.3 (2016), pp. 123–129. DOI: 10.3167/TRANS.2016.060310. URL: https://www.berghahnjournals.com/view/journals/transfers/6/3/trans060310.xml.

[2] Lola Woetzel et al. *Smart Cities: Digital Solutions for a More Livable Future*. McKinsey Global Institute. 2018. URL: https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/smart-cities-digital-solutions-for-a-more-livable-future.

[3] Mahwish Memon and Claudio Rossi. “A Review of EV Adoption, Charging Standards, and Charging Infrastructure Growth in Europe and Italy”. In: *Batteries* 11.6 (2025). ISSN: 2313-0105. DOI: 10.3390/batteries11060229. URL: https://www.mdpi.com/2313-0105/11/6/229.

[4] Imane Argui, Maxime Gueriau, and Samia Ainouz. “Advancements in Mixed Reality for Autonomous Vehicle Testing and Advanced Driver Assistance Systems: A Survey”. In: *IEEE Transactions on Intelligent Transportation Systems* 25.12 (2024), pp. 19276–19294. DOI: 10.1109/TITS.2024.3473740.

[5] NVIDIA. *NVIDIA DRIVE Thor: The Superchip for the Future of Autonomous Vehicles*. NVIDIA. 2022. URL: https://blogs.nvidia.com/blog/drive-thor/.

[6] Kai Zhang and Stuart Batterman. “Air pollution and health risks due to vehicle traffic”. In: *Science of The Total Environment* 450-451 (2013), pp. 307–316. ISSN: 0048-9697. DOI: https://doi.org/10.1016/j.scitotenv.2013.01.074. URL: https://www.sciencedirect.com/science/article/pii/S0048969713001290.

[7] ETSI. *Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Release 2*. Technical Report TR 102 638 V2.1.1. Sophia Antipolis, France: European Telecommunications Standards Institute, Apr. 1, 2024. URL: https://www.etsi.org/deliver/etsi_tr/102600_102699/102638/02.01.01_60/tr_102638v020101p.pdf.

[8] Kan Zheng et al. "Heterogeneous Vehicular Networking: A Survey on Architecture, Challenges, and Solutions". In: *IEEE Communications Surveys and Tutorials* 17.4 (2015), pp. 2377–2396. DOI: 10.1109/COMST.2015.2440103.

[9] Saleh Yousefi, Mahmoud Siadat Mousavi, and Mahmood Fathy. "Vehicular Ad Hoc Networks (VANETs): Challenges and Perspectives". In: *2006 6th International Conference on ITS Telecommunications*. 2006, pp. 761–766. DOI: 10.1109/ITST.2006.289012.

[10] Wasim A. Ali, Michèle Roccotelli, and Maria Pia Fanti. "Digital Twin in Intelligent Transportation Systems: a Review". In: *2022 8th International Conference on Control, Decision and Information Technologies (CoDIT)*. Vol. 1. 2022, pp. 576–581. DOI: 10.1109/CoDIT55151.2022.9804017.

[11] Muhammad Sami Irfan, Sagar Dasgupta, and Mizanur Rahman. "Toward Transportation Digital Twin Systems for Traffic Safety and Mobility: A Review". In: *IEEE Internet of Things Journal* 11.14 (2024), pp. 24581–24603. DOI: 10.1109/JIOT.2024.3395186.

[12] *Open Data Bologna*. URL: https://opendata.comune.bologna.it/pages/home/.

[13] Zilin Huang et al. "Toward C-V2X Enabled Connected Transportation System: RSU-Based Cooperative Localization Framework for Autonomous Vehicles". In: *IEEE Transactions on Intelligent Transportation Systems* 25.10 (2024), pp. 13417–13431. DOI: 10.1109/TITS.2024.3410185.

[14] Yunpeng Guo et al. "3D Digital Twin of Intelligent Transportation System based on Road-Side Sensing". In: *Journal of Physics: Conference Series* 2083.3 (Nov. 2021), p. 032022. DOI: 10.1088/1742-6596/2083/3/032022. URL: https://dx.doi.org/10.1088/1742-6596/2083/3/032022.

[15] Binbin Lu et al. “Cooperative Perception Aided Digital Twin Model Update and Migration in Mixed Vehicular Networks”. In: *IEEE Transactions on Intelligent Transportation Systems* 26.2 (2025), pp. 2293–2308. DOI: 10.1109/TITS.2024.3496121.

[16] Lejun Jiang, Tamás G. Molnár, and Gábor Orosz. “On the deployment of V2X roadside units for traffic prediction”. In: *Transportation Research Part C: Emerging Technologies* 129 (2021), p. 103238. ISSN: 0968-090X. DOI: https://doi.org/10.1016/j.trc.2021.103238. URL: https://www.sciencedirect.com/science/article/pii/S0968090X21002515.

[17] Morteza Mohammadi Zanjireh and Hadi Larijani. “A Survey on Centralised and Distributed Clustering Routing Algorithms for WSNs”. In: vol. 2015. May 2015. DOI: 10.1109/VTCSpring.2015.7145650.

[18] Fangchun Yang et al. “An overview of Internet of Vehicles”. In: *China Communications* 11.10 (2014), pp. 1–15. DOI: 10.1109/CC.2014.6969789.

[19] H. Hartenstein and L.P. Laberteaux. “A tutorial survey on vehicular ad hoc networks”. In: *IEEE Communications Magazine* 46.6 (2008), pp. 164–171. DOI: 10.1109/MCOM.2008.4539481.

[20] Daniel Jiang and Luca Delgrossi. “IEEE 802.11p: Towards an International Standard for Wireless Access in Vehicular Environments”. In: *VTC Spring 2008 - IEEE Vehicular Technology Conference*. 2008, pp. 2036–2040. DOI: 10.1109/VETECS.2008.458.

[21] Yunxin Li. “An overview of the DSRC/WAVE technology”. In: *International conference on heterogeneous networking for quality, reliability, security and robustness*. Springer. 2010, pp. 544–558.

[22] Toyota Motor North America. *Toyota and Lexus to Launch Technology to Connect Vehicles and Infrastructure in the U.S. in 2021*. Toyota Pressroom. Apr. 16, 2018. URL: https://pressroom.toyota.com/toyota-and-lexus-to-launch-technology-connect-vehicles-infrastructure-in-u-s-2021/.

[23] 3GPP. *Technical Specification Group Services and System Aspects; Study on Enhancement of 3GPP Support for 5G V2X Services (Release 16)*. Technical Report TR 22.886. 3rd Generation Partnership Project (3GPP), Dec. 2018. URL: https://www.3gpp.org/ftp/Specs/archive/22_series/22.886/.

[24] Ahmad Alalewi, Iyad Dayoub, and Soumaya Cherkaoui. “On 5G-V2X Use Cases and Enabling Technologies: A Comprehensive Survey”. In: *IEEE Access* 9 (2021), pp. 107710–107737. DOI: 10.1109/ACCESS.2021.3100472.

[25] Valerian Mannoni et al. “A Comparison of the V2X Communication Systems: ITS-G5 and C-V2X”. In: *2019 IEEE 89th Vehicular Technology Conference (VTC2019-Spring)*. 2019, pp. 1–5. DOI: 10.1109/VTCSpring.2019.8746562.

[26] David Eckhoff, Nikoletta Sofra, and Reinhard German. “A performance study of cooperative awareness in ETSI ITS G5 and IEEE WAVE”. In: *2013 10th Annual Conference on Wireless On-demand Network Systems and Services (WONS)*. 2013, pp. 196–200. DOI: 10.1109/WONS.2013.6578347.

[27] European Telecommunications Standards Institute. *Intelligent Transport Systems (ITS); Communications Architecture*. Standard EN 302 665 V1.1.1. Sophia Antipolis, France: ETSI, Sept. 2010. URL: https://www.etsi.org/deliver/etsi_en/302600_302699/302665/01.01.01_60/en_302665v010101p.pdf.

[28] European Telecommunications Standards Institute. *Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 2: Specification of Cooperative Awareness Basic Service*. Standard EN 302 637-2 V1.3.1. Sophia Antipolis, France: ETSI, Sept. 2014. URL: https://www.etsi.org/deliver/etsi_en/302600_302699/30263702/01.03.01_60/en_30263702v010301p.pdf.

[29] ETSI. *Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Local Dynamic Map (LDM); Rationale for and guidance on standardization*. Technical Report TR 102 863 V1.1.1. Sophia Antipolis, France: European Telecommunications Standards Institute, June 2011. URL: https://www.etsi.org/deliver/etsi_tr/102800_102899/102863/01.01.01_60/tr_102863v010101p.pdf.

[30] European Telecommunications Standards Institute. *Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 3: Specification of Decentralized Environmental Notification Basic Service*. Standard EN 302 637-3 V1.3.1. Sophia Antipolis, France: ETSI, Apr. 2019. URL: https://www.etsi.org/

deliver/etsi_en/302600_302699/30263703/01.03.01_60/en_30263703v010301p.pdf.

[31] Aidan Fuller et al. “Digital Twin: Enabling Technologies, Challenges and Open Research”. In: *IEEE Access* 8 (2020), pp. 108952–108971. DOI: 10.1109/ACCESS.2020.2998358.

[32] Michael Grieves. *Digital Twin: Manufacturing Excellence through Virtual Factory Replication*. Mar. 2015. URL: https://www.researchgate.net/publication/275211047_Digital_Twin_Manufacturing_Excellence_through_Virtual_Factory_Replication.

[33] Azad M. Madni, Carla C. Madni, and Scott D. Lucero. “Leveraging Digital Twin Technology in Model-Based Systems Engineering”. In: *Systems* 7.1 (2019). ISSN: 2079-8954. DOI: 10.3390/systems7010007. URL: https://www.mdpi.com/2079-8954/7/1/7.

[34] Werner Kritzinger et al. “Digital Twin in manufacturing: A categorical literature review and classification”. In: *IFAC-PapersOnLine* 51.11 (2018). 16th IFAC Symposium on Information Control Problems in Manufacturing INCOM 2018, pp. 1016–1022. ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2018.08.474. URL: https://www.sciencedirect.com/science/article/pii/S2405896318316021.

[35] Qinglin Qi and Fei Tao. “Digital Twin and Big Data Towards Smart Manufacturing and Industry 4.0: 360 Degree Comparison”. In: *IEEE Access* 6 (2018), pp. 3585–3593. DOI: 10.1109/ACCESS.2018.2793265.

[36] Wahib Saif. *Coding Reality: Decoding Digital Twin Architecture*. Okana. Mar. 28, 2024. URL: https://www.okana.global/insights/coding-reality-decoding-digital-twin-architecture/.

[37] Paolo Testolina et al. “Boston Twin: the Boston Digital Twin for Ray-Tracing in 6G Networks”. In: *Proceedings of the 15th ACM Multimedia Systems Conference*. MMSys ’24. New York, NY, USA: Association for Computing Machinery, 2024, pp. 441–447. ISBN: 9798400704123. DOI: 10.1145/3625468.3652190. URL: https://doi.org/10.1145/3625468.3652190.

[38] Mark Coates and Andrew Foster. *Singapore’s Digital Twin – From Science Fiction to Hi-Tech Reality*. Infrastructure Global. May 4, 2023. URL:

https://infra.global/singapores-digital-twin-from-science-fiction-to-hi-tech-reality/.

[39] City of Helsinki. *Helsinki 3D*. City of Helsinki. 2025. URL: https://www.hel.fi/en/decision-making/information-on-helsinki/maps-and-geospatial-data/helsinki-3d.

[40] David Weir-McCall. *51World creates digital twin of the entire city of Shanghai*. Epic Games / Unreal Engine. Sept. 15, 2020. URL: https://www.unrealengine.com/en-US/spotlights/51world-creates-digital-twin-of-the-entire-city-of-shanghai.

[41] Andras Varga. *OMNeT++*. Ed. by Klaus Wehrle, Mesut Güneş, and James Gross. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 35–59. ISBN: 978-3-642-12331-3. DOI: 10.1007/978-3-642-12331-3_3. URL: https://doi.org/10.1007/978-3-642-12331-3_3.

[42] Pablo Alvarez Lopez et al. "Microscopic Traffic Simulation using SUMO". In: *The 21st IEEE International Conference on Intelligent Transportation Systems*. IEEE, 2018. URL: https://elib.dlr.de/124092/.

[43] INET Framework Developers. *IEEE 802.11 (Wi-Fi)*. INET Framework for OMNeT++. 2024. URL: https://inet.omnetpp.org/docs/users-guide/ch-80211.html.

[44] *OMNeT website - documentation*. URL: https://omnetpp.org/documentation/.

[45] *SUMO website - documentation*. URL: https://sumo.dlr.de/docs/index.html.

[46] OpenStreetMap Foundation. *OpenStreetMap*. OpenStreetMap Foundation. 2025. URL: https://www.openstreetmap.org/.

[47] Pingfu Chao et al. "A Survey on Map-Matching Algorithms". In: *Databases Theory and Applications*. Ed. by Renata Borovica-Gajic, Jianzhong Qi, and Weiqing Wang. Cham: Springer International Publishing, 2020, pp. 121–133. ISBN: 978-3-030-39469-1.

[48] *Textual website*. URL: https://textual.textualize.io.

[49] SUMO Team. *Flowrouter.py Tool — SUMO Documentation*. Eclipse SUMO. Sept. 10, 2024. URL: https://sumo.dlr.de/docs/Tools/Detector.html.