SIMULATING DOWNTOWN SAN ANTONIO TRAFFIC WITH VANETS

Samantha Jackson
Department of Science
Texas A&M University-San Antonio
San Antonio, TX 78224
sjack012@jaguar.tamu.edu

Jonathan Davis
Department of Science
Texas A&M University-San Antonio
San Antonio, TX 78224
jdavi074@jaguar.tamu.edu

Nico Gibson
Department of Science
Texas A&M University-San Antonio
San Antonio, TX 78224
ngibs01@jaguar.tamu.edu

May 14, 2025

ABSTRACT

Traffic congestion in urban environments continues to pose significant challenges especially in cities with an over-reliance on vehicles. Vehicular Ad-Hoc Networks (VANETs) have emerged as a promising technological solution to traffic management and safety by enabling the communication between vehicles and infrastructure. This study presents a simulation of downtown San Antonio with VANETs to explore their potential for mitigating congestion and avoiding accidents. The simulation offers insights into VANETs performance in real-world scenarios. A comparative analysis of existing VANET simulators highlights the pros and cons of our simulation approach. Our findings underscore the potential of VANETs in creating more efficient urban transport.

1 Introduction

Traffic congestion in urban areas has increased in recent years, a trend exacerbated as employees return to the office during the transition away from COVID-era work from home policies. This is accompanied by a growing number and variety of autonomous vehicles operating in an increasing number of metropolitan areas. These two forces represent some of the most significant motivation spurring the development of vehicular traffic simulations and communication networks. The goals of such systems include mitigating congestion through the communication of road conditions and notification of accidents allowing the implementation of alternate routes when needed, accident avoidance through the use of distance measuring, alerts and autonomous intervention such as automatic braking, and a better understanding of how traffic conditions can impact the performance of such vehicular communication networks as measured by network throughput and packet delivery ratio.

The platform upon which such simulations and communication have been developed are called VANETs or vehicular ad-hoc networks. VANETs, based on and considered a subset of MANETs or mobile ad-hoc networks, are designed to both simulate and support vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) communication where vehicles communicate with road-side units (RSUs) using on-board unit (OBU) devices. Architectures of VANETs can be of either only one type, or can utilize a hybrid approach involving both V2V and V2I communication simultaneously.

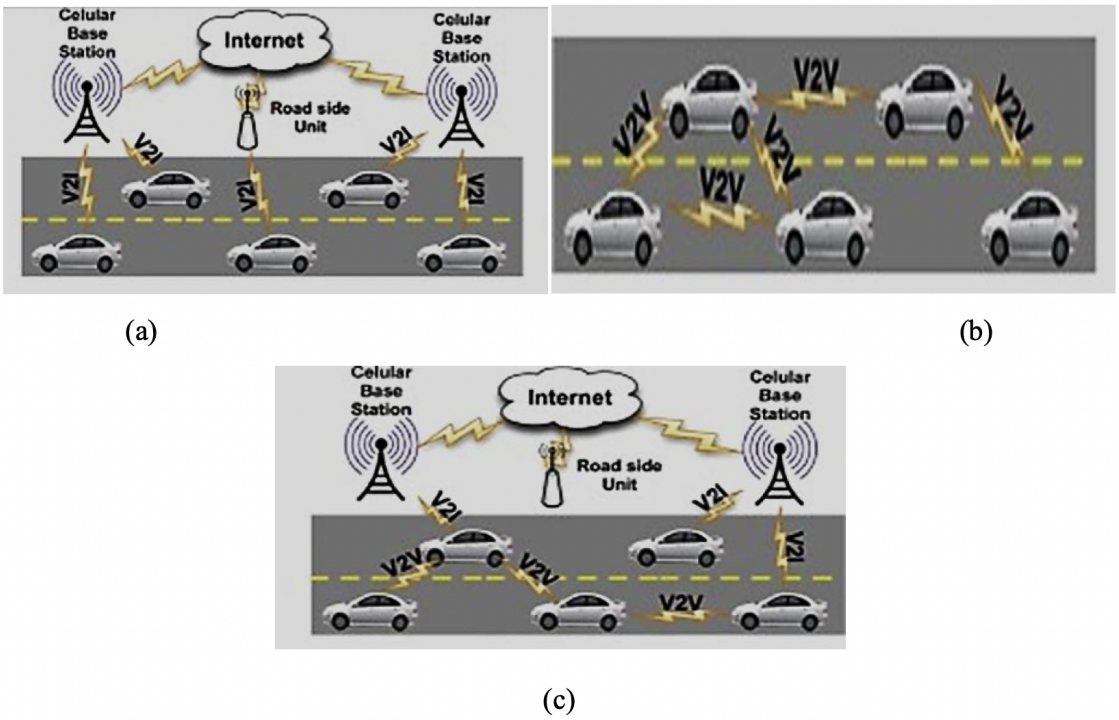

Figure 1: Network architecture of VANETs. (a) Infra-structured. (b) Ad hoc. (c) Hybrid [1]

VANET routing protocols are divided into two main categories. Topology-based protocols can be either unicast or multicast, and rely on the layout of connections in the network. Topology-based protocols are further divided into two types. Proactive topology-based routing protocols maintain information about the network topology and continuously update that information in routing tables contained in the nodes. Reactive topology-based routing protocols do not continuously maintain this topology information, and instead update routing information only when a source node needs to transmit data packets to a destination node. Geography-based protocols by contrast use the geographic location of nodes obtained using methods such as GPS coordinates to make routing decisions instead of relying on routing tables or route discovery processes.

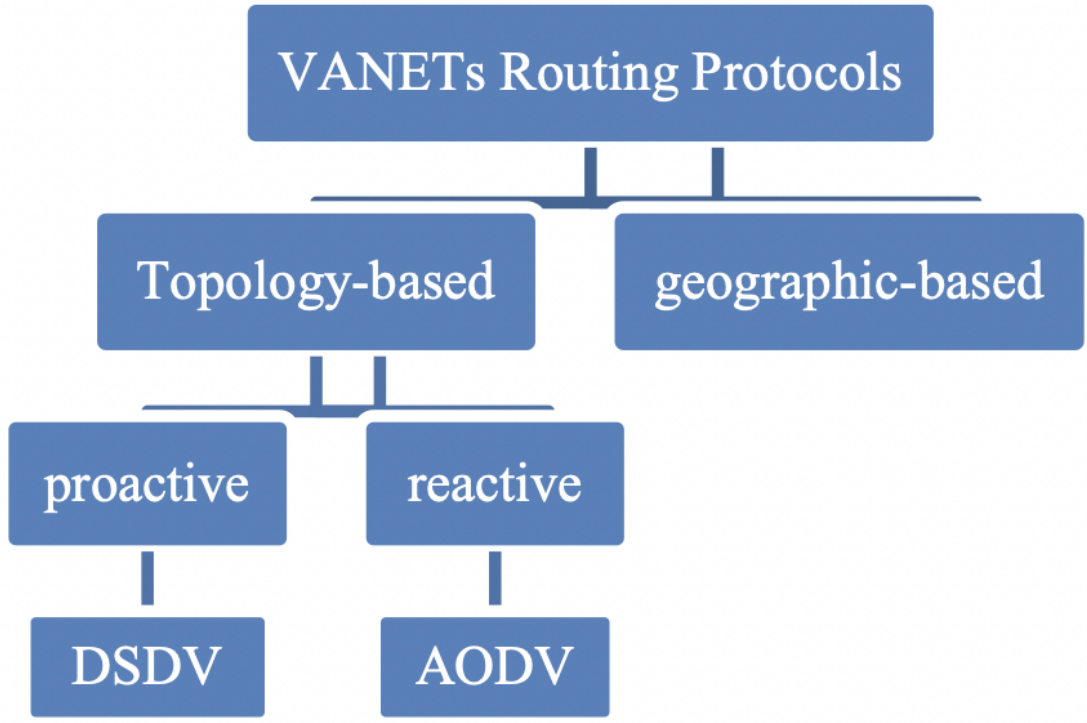

Figure 2: Routing protocols of VANETs [2]

The deployment and testing of VANETs in the real world is both incredibly expensive and labor intensive. Alternatively the use of simulation is a much more cost effective substitute that can be used to test this technology prior to vehicle testing and commercial release. Therefore it is very important that these simulation frameworks are accurate to current technological advancements. In this study we analyze the three components that make up VANET simulators, discuss three popular frameworks for VANET simulation, and explain the process required to convert an OpenStreetMap file of downtown San Antonio, Texas to a network simulation in OMNeT++ utilizing Veins and SUMO.

2 Mobility Simulators

There are a multitude of VANET simulators that are easily available for anyone to use. The performance of any given VANET can generally be determined via their mobility generator, network simulator, and VANET simulator. The mobility generator enhances the realism of vehicle movement based on the model for the road and the scenario parameters (speed limits, check-in, departure times, etc). Generally a good mobility model should include the following feature set: [3]

- **Accurate topological maps:** The model should consider the maps containing various types of streets with varying traffic density and different node speeds (nodes being cars).
- **Vehicle speed variation:** Nodes in the network should not immediately accelerate or decelerate they should gradually increase or decrease in speed.
- **Obstacles:** Barriers for wireless communication should implemented in order to accurately simulate V2I and V2V communication.
- **Points of attraction:** Points on the map that serve as final destinations for multiple drivers that could cause bottlenecks.
- **Weather and traffic conditions:** The effects of weather conditions and changes in traffic conditions like rush hour should be considered in the density and movement of vehicles.
- **Driving patterns:** Drivers should interact with obstacles like adjacent vehicles and pedestrians.

There are a multitude of different mobility models. However, we will only be covering Simulation of Urban Mobility (SUMO). SUMO has dominated the VANET simulation space because of its wide variety of input formats including

support for OpenStreetMap, NavTeq, MATsim, VISSIM, VISUM, and OpenDRIVE. Additionally SUMO uses C++ and Python making it compatible with many network simulators including OMNeT++, NS-2, and NS-3. SUMO is available for Windows, Linux, and macOS distributions. The major drawback for this mobility simulator is simply its difficulty. SUMO can be difficult to setup and use. Despite that it contains a massive feature set making it perfect for mobility simulation. [4]

3 Network Simulators

There are two types of simulators: commercial and open-source. Commercial software is not free and its development is supported by the company that developed the software. Open-source software is free of charge and has community driven development usually on GitHub [5]. You cant necessarily guarantee that open-source software will run whereas commercial software development bug fixes are swift. Network simulators are an important part of the simulation process that manages the ad-hoc network proposed by the mobility generator.

There are a multitude of network simulators, including NS-2, NS-3, and OMNeT++. However, we will only be covering OMNeT++. This network simulator is an open-source, modular, component based C++ simulation library and framework with a GUI. This allows users to add their own packages in addition to the already existing framework. For example, VANET simulations in OMNeT++ generally use the Veins and its dependency package INET. The GUI also allows users to clearly view the simulation while OMNeT++ is collecting and managing relevant network data. The simulation IDE runs on Windows, Linux, and macOS distributions [6].

4 VANET Simulators

Closing out the development stack VANET simulators manage driver reactions to network data. Say for example if a driver gets a collision warning through the network they may decide to take action by getting off the highway or braking. The combination of network simulators and mobility simulators through VANET simulators make for a very powerful tool that accurately simulates traffic movement.

There are a multitude of VANET simulators including TraNS, MobiREAL, and Veins. However, we will only be covering Vehicular In-Network Simulations (Veins). This simulator is based off OMNeT++ and SUMO. When combined this provides a sleuth of models for IVC simulation. Veins uses TCP links and Python scripts to that allow SUMO to act as a mobility model inside OMNeT++. This makes Veins an extremely powerful free tool [7].

5 Comparison of Simulation Environments

The available literature that reviews the VANET landscape identifies a collection of features which distinguish the capabilities of the various solutions available today. Map quality and accuracy can differ between simulation packages. Some do not support real-world maps while others are capable of importing real-world maps from various sources, and not all are capable of resolving different road traffic densities and different speeds. Only some solutions are capable of representing vehicular acceleration and deceleration while others handle changes in speed as discrete and instantaneous. There are varying degrees of capability of VANETs to model obstacles to signals, such as buildings that may interrupt wireless transmissions, or road-traffic obstacles which can be static like road closures, or complex such as nearby vehicles, and pedestrians. Another feature to consider is whether the VANET is capable of simulating weather, and traffic conditions such as rain, fog, rush hours, weekends and school zones.

VANET simulations typically employ mobility traces generated by real-world experiments or dedicated road traffic simulators which are typically created offline to aid network simulation performance. However, as cited in the Sommer et al paper, the major drawback to using such offline mobility traces is that they only model the effect of road traffic on network traffic, and cannot demonstrate any effect network traffic might have on road traffic [8]. In response to this limitation, more complex simulation techniques which include bidirectional coupling between network and road traffic simulations have been developed.

5.1 Veins

Vehicles in Network Simulation also known as Veins is a well established bidirectional simulator which combines two simulators into a convenient package. Which is considered easy to install and only moderately difficult to use according to the survey conducted by I. A. Aljabry and G. A. Al-Suhail [2]. Veins integrates the network traffic simulator OMNeT++ with the microscopic road traffic simulator SUMO (Simulation of Urban Mobility) and can support both geography-based and topology-based protocols. The SUMO integration enables the modeling of road

vehicles, public transport, and pedestrians. It supports a wide variety of map import formats such as OpenStreetMap, NavTeq MATsim, VISSIM, VISUM, and OpenDRIVE, and can handle tasks such as finding routes, visualizations, and emission calculations.

Applications for which Veins has been used include traffic optimization, safety applications, traffic infrastructure communication, security, and platooning where vehicles autonomously follow a lead car at reduced distance to improve fuel efficiency, and improve traffic flow and safety. One potential drawback to Veins is the increased potential for unreliable results as bugs in either the OMNeT++ or SUMO component can adversely impact data.

OMNeT++ integration makes for a somewhat difficult user experience. Not only does Veins require OMNeT++ but it also requires the INET dependency. Depending on your operating system or file locations this can directly impact whether or not your simulations can run. OMNeT++ has the capability to corrupt its own file systems. Excluding the time it takes to compile INET and Veins you can expect to take a long time to set up this development stack. This doesn't necessarily lend itself to usability, but it makes up for it with a broad, expandable feature set and the ability to program network projects in C++. This makes OMNeT++ particularly valuable to VANET developers this makes SUMO, Veins, INET, OMNeT++ integration one of the definitive ways to experiment with VANETs.

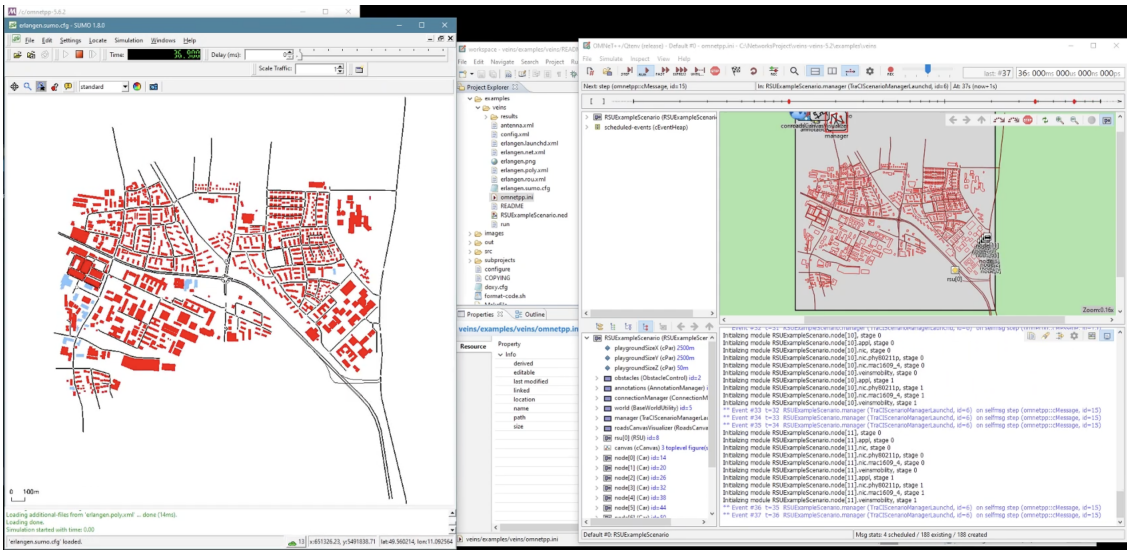

Figure 3: VEINS, RSUExampleScenario simulation. Left: SUMO GUI, Right: OMNeT++ GUI

5.2 Eclipse MOSAIC

Eclipse MOSAIC is an open-source, multi-domain simulation framework that provides users the flexibility of performing vehicle-to-everything (V2X) simulations with simulators of their choosing through the use of management modules which coordinate the simulators with Eclipse MOSAIC. Specifically it uses a federate manager to receive and send data between Eclipse MOSAIC's runtime infrastructure and other simulators, a time manager to synchronize the simulators and the federates and ensure events occur in the correct order, and an interaction manager which enables the exchange of data between federates through a publish-subscribe program. Another benefit to Eclipse MOSAIC is the capability of visualizing data in different ways using a variety of tools which include both 2-D and 3-D visualizers.

A shortcoming of Eclipse MOSAIC in comparison to more established VANETs like Veins is the lack of established research in several areas. According to the VANET simulator review by J. Weber, M. Neves and T. Ferreto, there is no significant research done in the field of software-defined networking (SDN) using Eclipse MOSAIC. SDN uses software applications to centrally control networks and is well-suited to dynamic network environments with a large number of connected devices such as VANETs. In contrast there are several examples cited where Veins is utilized in this field such as supporting SDN controllers working with RSUs to control an entire network, and in the rapid and reliable emergency message dissemination in areas with sparse RSU coverage. [9]

Eclipse MOSAIC is significantly more user friendly compared to Veins OMNeT++ integration. It also has significantly more developer support easily available to users. With its capability to easily create VANET simulations with web socket integration this an simple software to get up and running. Though it generally provides a less features compared to the OMNeT++ stack and its premium version requires a hefty user subscription. This subscription includes the 3-D features of Eclipse MOSAIC so there's very little reason to pick the non-premium version over other free, open-source simulators like the OMNeT++ stack.

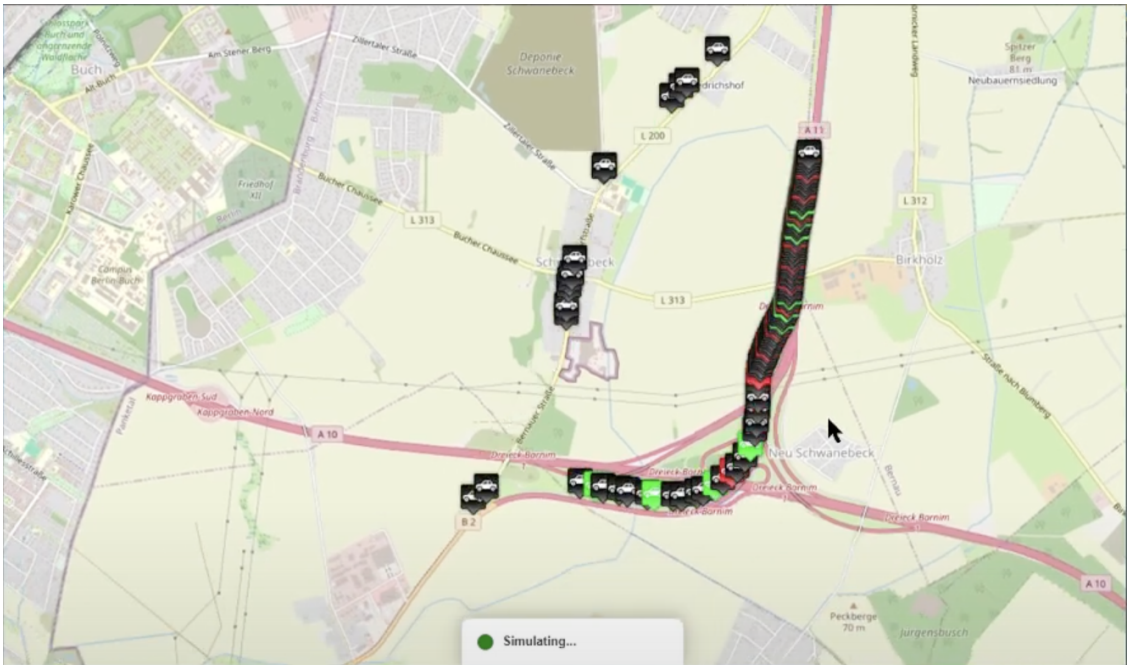

Figure 4: Eclipse MOSAIC, packaged Barnim websocket visualization

5.3 VANETsim

VANETsim is a security-focused microscopic VANET capable of utilizing real-world maps imported from OpenStreetMap, and provides detailed visualizations through a GUI interface. Unfortunately, VANETsim does not have the capability to support 3D environments. It is distributed with several security and privacy modules, and although no longer maintained, has well-documented features and an easy to follow guide according to J. Weber et al [9]. Like Eclipse MOSAIC, there is no available information indicating it can be used to explore SDN applications, and it would be difficult to extend the simulator to support new technology because all its components are directly integrated.

VANETsim has the one of the smallest feature sets compared to other VANET frameworks. Though it makes it relatively easy to take map data and start a simulation immediately. These simulations also automatically save network data to the VANETsim file location. Making it relatively easy to analyze data generated by VANETsim. Despite this there is little reason to use this software over the larger feature sets and continued support from more established projects like Eclipse MOSAIC and Veins.

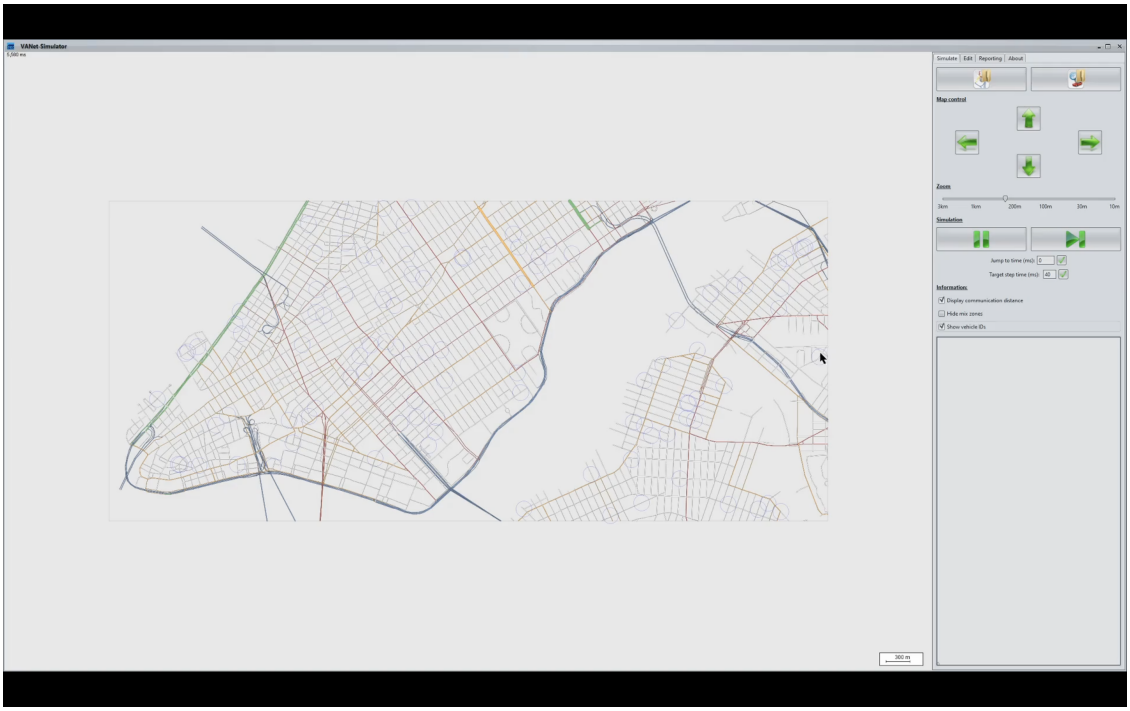

Figure 5: VANETsim, packaged New York simulation

The CIA triad is clearly an important aspect of VANETs, because they represent dynamic environments full of fast-moving vehicles capable of causing severe injury and death, and are of vital importance in urban environments. The three VANET solutions presented in this paper have varying capabilities regarding confidentiality, integrity and availability. None of the three are capable of performing encryption and decryption. However, Veins and Eclipse MOSAIC allow users to import cryptographic libraries such as Crypto++. VANETsim can only represent whether a message is encrypted through the use of a boolean flag and assumes that attacks cannot break such encryption.

As with encryption, Veins and Eclipse MOSAIC allow users to import integrity services through the use of third-party libraries. VANETsim by contrast uses an intrusion detection system (IDS) which monitors application layer data to detect malicious actors and associated false data in VANETs.

There is less research available concerning the security concept of availability in VANETs. However, both Veins and Eclipse MOSAIC are well suited to work in this area, as protocols that implement redundancy need to be implemented in their network simulators. Additionally, watchdog nodes that monitor the status of other nodes may also be implemented in their network simulators. There is no information available regarding VANETsim's capabilities in this area.

6 Network Description

To create any valuable contributions to these network simulators the choice of software is paramount in the development process. The VANET environment we considered for development is as follows:

- **OMNeT++ 5.6.2:** an open source event-based network simulator which has a large amount of pre-established projects and research. This makes it an ideal choice for VANET development.
- **Veins 5.2:** selected for multiple reasons. Veins is a free/open source project, has direct compatibility with OMNeT++, its feature set is extremely extensive and out-competes many existing VANET simulators on the market.
- **SUMO 1.8.0** a Veins dependent and its compatibility with OpenStreetMaps.
- **INET 4.2.1:** an open source Veins dependent.
- **Debian Linux and Windows 10:** Windows 10 handles the SUMO integration element and Debian Linux runs instant-veins-5.2-i1

This development stack comes with its own set of difficulties not limited to runtime errors on Windows 10/11 operating systems. However, due to its open source nature, it remains an ideal choice.

7 OpenStreetMap for VANETs

The OpenStreetMap (OSM) project was originally founded in 2004 by Steve Coast and has established itself as the most famous example of the Volunteered Geographic Information (VGI) and is dominate in the VGI landscape. The goal of this project was to obtain crowd-sourced data to obtain accurate, locally sourced data about roadways. The success of this VGI can be mostly attributed to its Web 2.0 format, the wide availability of highly-accurate Global Positioning System (GPS) through smartphones, and the low barrier to entry with users only needing to register to contribute to OSM. [10]

In the case of VANETs data from OSM is particularly valuable for its high levels of detail and accuracy, especially in heavily populated areas. This made simulating downtown San Antonio a particularly viable dataset. Data in this area of the map contains detail down to the levels of roadways, motorways, individual intersections, and speed limits. When implemented in SUMO the software is capable of handling the movement of a set amount of vehicles, the interactions between vehicles, and the length of the simulation. [11]

The following explanation of setting up VANETs will assume that the reader has a basic understanding of OMNeT++ and the required dependencies. This is not particularly easy to set-up by any means and usually will require you to understand how to properly debug OMNeT++. The following SUMO simulations were created in a Windows 10 distribution and the OMNeT++ simulations were run in a Debian Linux distribution using instant-veins-5.2-i1 on two separate pieces of computer hardware.

7.1 Exporting OSM Data

There are two ways to handle exporting OSM data. You can either extract it directly from the OpenStreetMap website or through the third-party tool Java OSM (JOSM). JOSM is an open-source, extensible editor created in 2006 for OSM files. It supports loading GPX tracks, background imagery, OSM data from local and online sources, and the editing of OSM nodes, ways and relations. JOSM is still receiving regular updates and bug-fixes to this date. [12]

Extracting map data directly from OpenStreetMap is a relatively straightforward process that doesn't require users to register to the website. The domain for OSM can be found here `https://www.openstreetmap.org/`. Once on the site you can navigate to the search bar and search for the area you wish to export. After navigating to the region you can see data about user contributions, the date of contribution, and the relevant XML data. Navigate to the export button and select a region to export, then click *Export*. This will download the relevant *.osm* file.

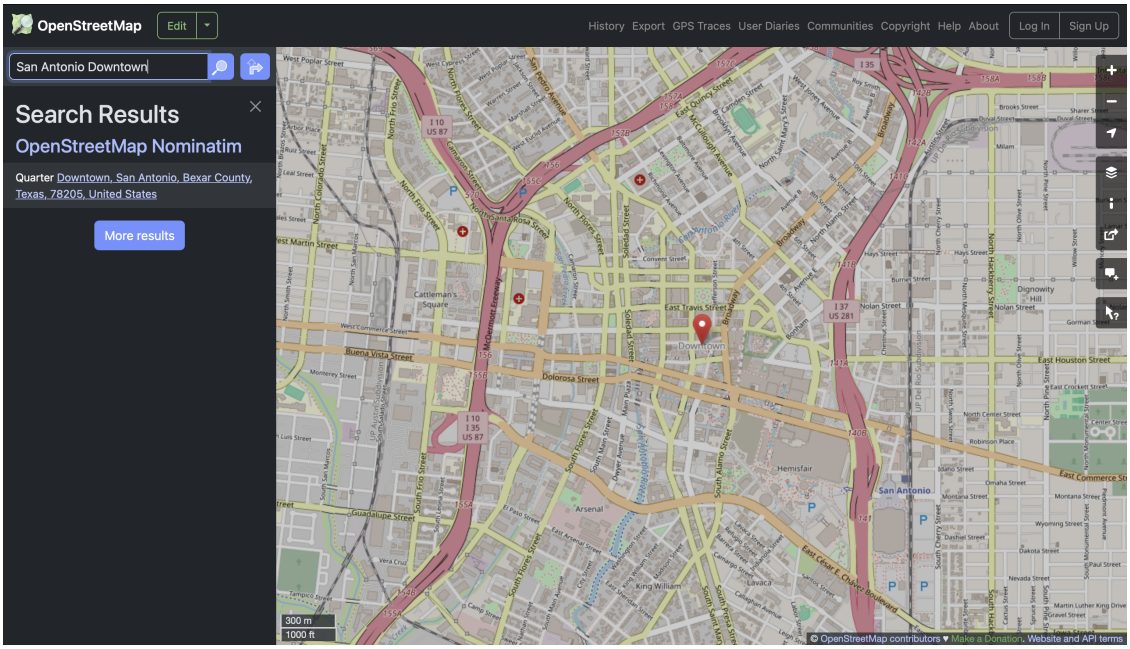

Figure 6: Search function in OpenStreetMap

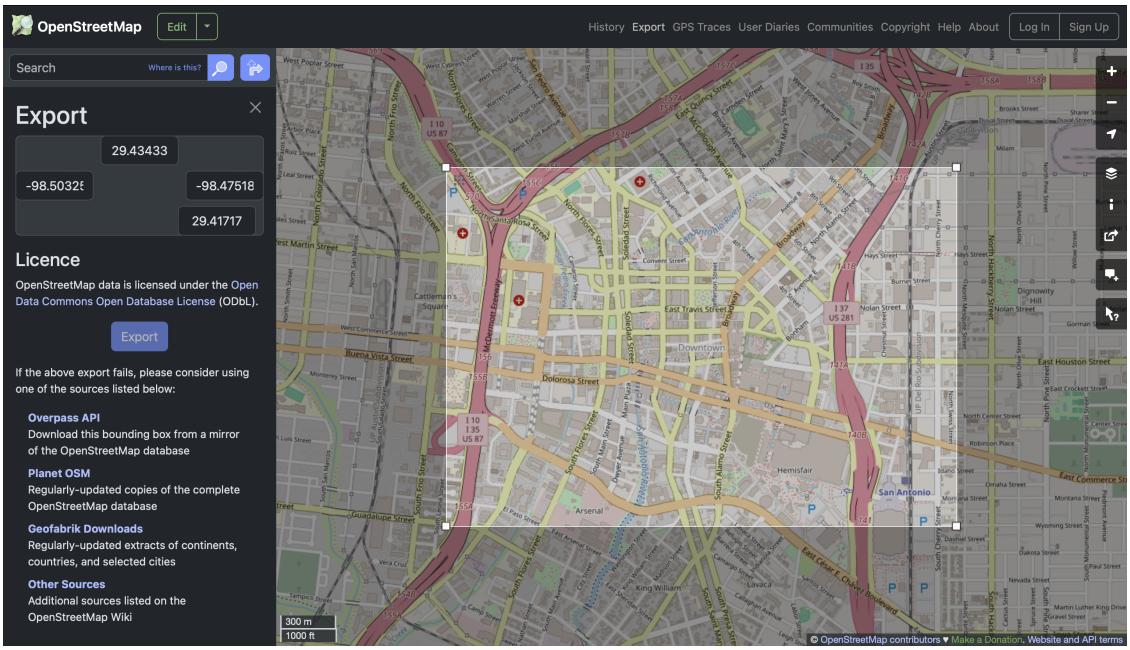

Figure 7: Exporting OpenStreetMap data of San Antonio downtown

7.2 Network Files and OSM Conversions

There are a multitude ways to handle OSM data. Due to the complexity of OSM data it's common for *.osm* to *.net.xml* conversions to create simulation inaccuracies for mobility simulators like SUMO. Typically this requires editing the OSM data prior to net conversions. OSM files can be edited either manually through Notepad++ or the more established JOSM software. It's generally recommended that JOSM is used for its graphical interface. This makes it relatively easy to see edits made to the OSM file in real time. Once all relevant edits are made this OSM file is ready for conversion.

The official SUMO documentation outlines the use of the netconvert command in order to convert *.osm* to *.net.xml* format `https://sumo.dlr.de/docs/netconvert.html`. This command line application is tested on Linux and Windows distributions. In the case of importing a network from OSM we'd use the *netconvert –osm my_osm_net.xml* command. netconvert also comes packaged with a plethora of options for processing conversions. We'll use add these options to the end of the command *netconvert –osm-files sadowntown.osm –output-file sadowntown.net.xml –geometry.remove –roundabouts.guess –ramps.guess –junctions.join –tls.guess-signals –tls.discard-simple –tls.join* [13]. From this file we are going to create a trips file with the command *randomTrips.py -n sadowntown.net.xml -e 1000 -o sadowntown.trips.xml*. Additionally we will create a route file with the command *duarouter -n sadowntown.net.xml –route-files sadowntown.trips.xml -o sadowntown.rou.xml –ignore-errors –write-trips* [14]. Typically these files will return errors but these are inconsequential. Once all of these steps are complete you will be able to use these files for SUMO integration.

7.3 SUMO Integration

In order to create a simulation file we will need a configuration file which we will create in command line using *type nul > sadowntown.sumo.cfg*. This file needs the following edits in a text editor of your choice. The input parameters will take the net-file input from *sadowntown.net.xml* and the route-files input from *sadowntown.rou.xml*. The time parameter will set the simulation time for 100 seconds.

To ensure that these files function properly in the OMNeT++ integration we will also need to make changes to the *sadowntown.rou.xml* file which handles the routing of vehicles. Typically you can simply ignore the errors when making the routing file if you're simply running the simulation in SUMO. However, OMNeT++ does not know how to handle when cars have incorrect routing. This file has 999 routes so we'll make a request that our output has the ability to write trips and will ignore routing errors. Simply ignoring errors isn't enough.

```
<?xml version="1.0" encoding="UTF-8"?>


<configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="http://sumo.dlr.de/xsd/netconvertConfiguration.xsd">

    <input>
        <net-file value="sadowntown.net.xml"/>
        <route-files value="sadowntown.rou.xml"/>
    </input>

    <time>
        <begin value="0"/>
        <end value="100000"/>
    </time>

</configuration>
```

Figure 8: sadowntown.sumo.cfg XML script.

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- generated on 12/04/24 22:09:08 by Eclipse SUMO duarouter Version 1.8.0
<configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="http://sumo.dlr.de/xsd/duarouterConfiguration.xsd">

    <input>
        <net-file value="sadowntown.net.xml"/>
        <route-files value="sadowntown.trips.xml"/>
    </input>

    <output>
        <output-file value="sadowntown.rou.xml"/>
        <alternatives-output value="sadowntown.rou.alt.xml"/>
        <write-trips value="true"/>
    </output>

    <report>
        <ignore-errors value="true"/>
        <ignore-route-errors="true"/>
    </report>

</configuration>
-->
```

Figure 9: Changes made to the sadowntown.rou XML script.

With the file obtained from conversion you can open the *.net.xml* in SUMO. This should load in a map generated from the *.osm* file with the presets we set when creating the *.net.xml* file. Running the simulation we can see the interaction between cars in major intersections and points of interest. You will also notice the large amount of warning messages related to teleporting vehicles which is typical with these large simulation files. This isn't necessarily a problem for our next steps.

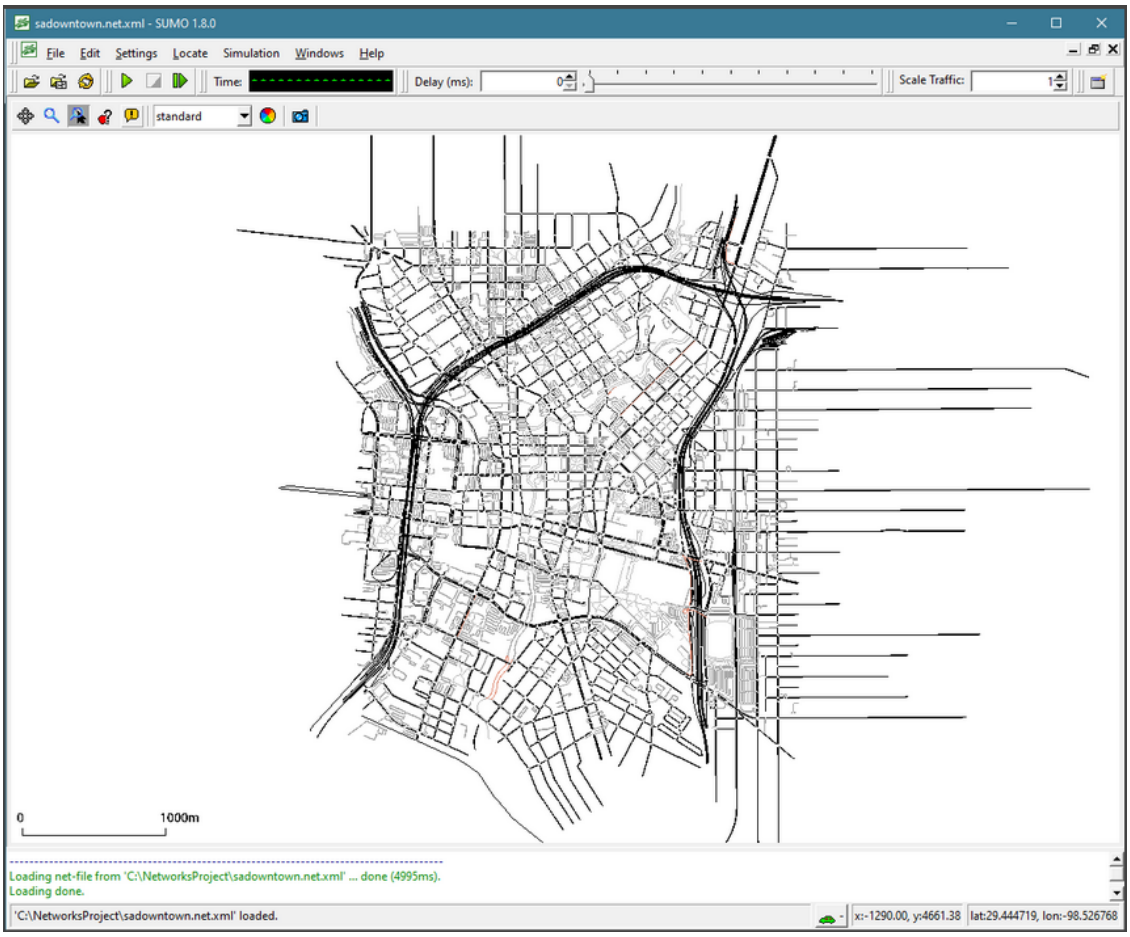

Figure 10: sadowntown.net.xml file in SUMO 1.8.0

```
Loading configuration ... done.
Loading net-file from 'C:\NetworksProject\sumo\simulation\sadowntown.net.xml' ... done (2986ms).
Loading done.
Simulation started with time: 0.00
Warning: Teleporting vehicle '172'; waited too long (yield), lane='104538933#0_0', time=549.00.
Warning: Vehicle '172' ends teleporting on edge '104538933#1', time 549.00.
Warning: Teleporting vehicle '166'; waited too long (yield), lane='104538933#0_1', time=551.00.
Warning: Vehicle '166' ends teleporting on edge '104538933#2', time 553.00.
Warning: Teleporting vehicle '285'; waited too long (jam), lane=':1175108912_3_0', time=653.00.
Warning: Vehicle '285' ends teleporting on edge '618087944#5', time 653.00.
Warning: Teleporting vehicle '211'; waited too long (jam), lane='-15184261#11_0', time=741.00.
Warning: Teleporting vehicle '433'; waited too long (yield), lane='-15165217#0_0', time=769.00.
Warning: Vehicle '433' ends teleporting on edge '1036561423', time 769.00.
Warning: Teleporting vehicle '470'; waited too long (jam), lane='1036561424_2', time=785.00.
Warning: Vehicle '470' ends teleporting on edge '121514385#1', time 787.00.
Warning: Vehicle '211' ends teleporting on edge '-15184261#6', time 811.00.
Warning: Teleporting vehicle '228'; waited too long (jam), lane='-15184261#10_0', time=824.00.
Warning: Teleporting vehicle '312'; waited too long (wrong lane), lane='155135494_0', time=845.00.
Warning: Vehicle '312' ends teleporting on edge '1034141251#0', time 847.00.
Warning: Teleporting vehicle '523'; waited too long (yield), lane='49426781_1', time=857.00.
Warning: Vehicle '523' ends teleporting on edge '150603593', time 857.00.
Warning: Vehicle '228' ends teleporting on edge '-15184261#6', time 872.00.
Warning: Teleporting vehicle '195'; waited too long (wrong lane), lane='154453843#1_0', time=937.00.
Warning: Vehicle '195' ends teleporting on edge '168952835', time 939.00.
Warning: Teleporting vehicle '414'; waited too long (jam), lane=':1175108912_3_0', time=1040.00.
Warning: Vehicle '414' ends teleporting on edge '618087944#5', time 1040.00.
Warning: Teleporting vehicle '247'; waited too long (yield), lane='104538933#4_1', time=1050.00.
Warning: Vehicle '247' ends teleporting on edge '104538933#5', time 1050.00.
Warning: Teleporting vehicle '342'; waited too long (jam), lane=':1175108823_7_1', time=1071.00.
Warning: Teleporting vehicle '463'; waited too long (jam), lane=':1175108823_1_0', time=1071.00.
Warning: Vehicle '342' ends teleporting on edge '121514385#1', time 1071.00.
Warning: Vehicle '463' ends teleporting on edge '121514385#1', time 1071.00.
Warning: Teleporting vehicle '695'; waited too long (yield), lane='104538933#4_0', time=1089.00.
Warning: Vehicle '695' ends teleporting on edge '104538933#5', time 1089.00.
Warning: Teleporting vehicle '505'; waited too long (jam), lane=':1669841818_0_1', time=1147.00.
Warning: Vehicle '505' ends teleporting on edge '155135481', time 1147.00.
```

Figure 11: sadowntown.net.xml warning messages in SUMO 1.8.0

7.4 OMNeT++ Integration

With the integration of SUMO we are finally ready for OMNeT++. Firstly we build the INET and Veins dependencies within OMNeT++. These required very specific versions in order to maintain compatibility. We'll also have to make some changes to makefile options. These can be found under OMNeT++ > Makemake > Makemake; Options > Custom. These changes are required for INET V.4+ because OMNeT++ doesn't immediately apply an improved message compiler nor does it link with the ws2_32 library. [15]

Once these packages are successfully built we are going to place our packages inside of the *veins-5.2/subprojects/veins_inet/examples/veins_inet* folder. When attempting to create a new project outside of this folder even after changing the package locations inside of the INET and Veins dependencies there were still several package errors. By running the simulation inside of the already working folder we side-step this issue. Initially this file runs the square.sumocfg simulation.

Under the *omnetpp.ini* we'll make the following change to line 44 changing our referenced xml document to *sadowntown.launchd.xml* this file will handle the files that are launched by the OMNeT++ network. Then we'll take the *sadowntown.net.xml, sadowntown.rou.xml,* and *sadowntown.cfg* file and place them into the current folder. Making sure to remove the existing square files and renaming the *square.launchd.xml* to *sadowntown.launchd.xml*. Navigating to the *sadowntown.launchd.xml* we'll remove the reference to the *square.poly.xml* file. Then change the existing reference to match the files we placed inside the current folder as shown in figure 14. Additionally we will make a change to the sim-time-limit changing it from 60s to 100s. This change will match the SUMO simulation time.

Once these changes are made we can run the *omnetpp.ini* file to start the simulation. However, in order to connect SUMO to Veins we will have to run the following command in *mingwenv.cmd /veins-veins-5.2/sumo-launchd.py -vv -c C:/SUMObin/sumo-gui.exe*. This command will call the *sumo-launchd.py* script and open the *sumo-gui.exe* file [16]. The following simulation was produced as shown in figure 15. The combination of these tools gives us a simulation where vehicles not only navigate roads realistically, but are capable of communicating relevant road information with hundreds of other vehicles throughout the network.

```
[ ]  # User-supplied makefile fragment(s)
     # >>>
     # inserted from file 'makefrag':
     #
     # Use the new message compiler introduced in OMNeT++ 5.3
     #
     MSGC:=$(MSGC) --msg6

     ifeq ($(PLATFORM),win32.x86_64)
       #
       # on windows we have to link with the ws2_32 (winsock2) library as it is no longer added
       # to the omnetpp system libraries by default (as of OMNeT++ 5.1)
       #
       LIBS += -lws2_32
       DEFINES += -DINET_EXPORT
       ENABLE_AUTO_IMPORT=-Wl,--enable-auto-import
       LDFLAGS := $(filter-out $(ENABLE_AUTO_IMPORT), $(LDFLAGS))

     endif

     # <<<
```

Figure 12: makefile options changes for INET.

Figure 13: Changes made to the omnetpp.ini file on line 44.

```xml
<?xml version="1.0"?>

<!--
// Copyright (C) 2018 Christoph Sommer <sommer@ccs-labs.org>
//
// Documentation for these modules is at http://veins.car2x.org/
//
// SPDX-License-Identifier: (GPL-2.0-or-later OR CC-BY-SA-4.0)
//
// This program is free software; you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation; either version 2 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
//
// -
//
// At your option, you can also redistribute and/or modify this file
// under a
// Creative Commons Attribution-ShareAlike 4.0 International License.
//
// You should have received a copy of the license along with this
// work.  If not, see <http://creativecommons.org/licenses/by-sa/4.0/>.
-->

<launch>
    <copy file="sadowntown.net.xml" />
    <copy file="sadowntown.rou.xml" />
    <copy file="sadowntown.sumo.cfg" type="config" />
</launch>

```

Figure 14: Changes made to the sadowntown.launchd.xml file.

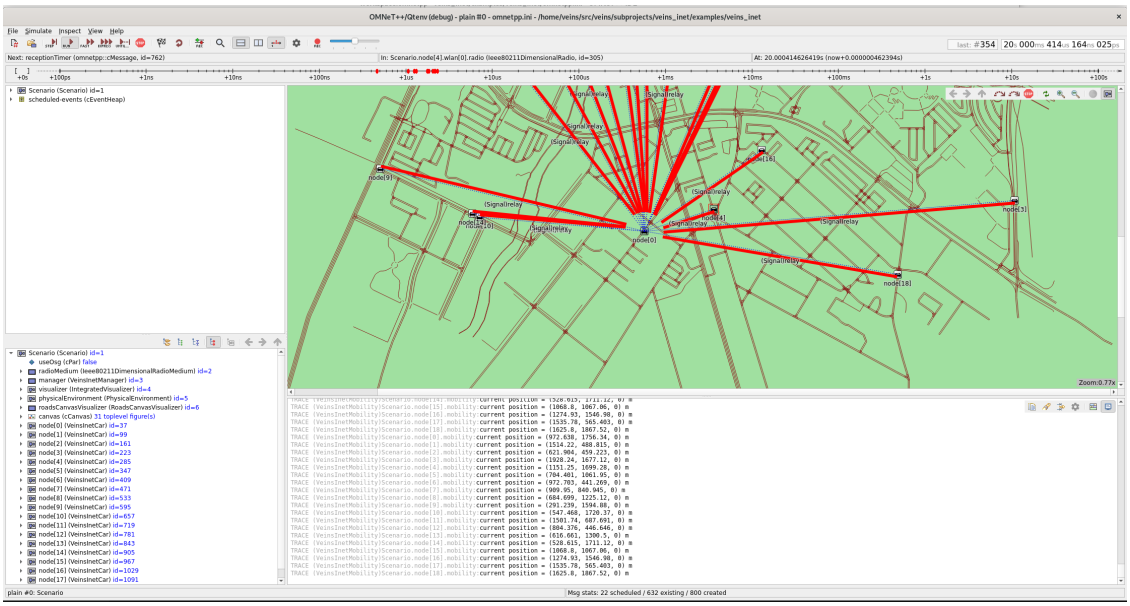

Figure 15: Simulation of downtown San Antonio running in OMNeT++.

8 Attempts at Network Analysis

After setting up this simulation we attempted to create a system that would generate *.sca* (Scalar Data) and *.vec* (Vector Data) files that would allow us to determine network throughput. However, our attempts were unsuccessful in generating anything but zero data points. Regardless I will outline our development process.

The creation of these files required several edits to multiple files including updates to *omnetpp.ini* and changes made to VeinsInetApplicationBase specifically in *VeinsInetApplicationBase.cc, VeinsInetApplicationBase.h*, and *VeinsInetApplicationBase.ned*. Firstly, we will address the changes made to the configuration settings file *omnetpp.ini*.

8.1 omnetpp.ini

The changes made to this file included the additions that allowed for scalar-recording and vector-recording as well as the creation of a throughput record.

```
# Misc
**.vector-recording = true
**.scalar-recording = true
**.node[*].app[0].throughput.record = true
```

Figure 16: Changes made to the MISC section of the omnetpp.ini file in the simulation directory.

8.2 VeinsInetApplicationBase

The changes made to the files in VeinsInetApplicationBase are intended to serve as a basis for our data collection. Firstly, *VeinsInetApplicationBase.h* was changed to define packet statistics, signals for vector recording and packet processing. Secondly, the changes made to *VeinsInetApplicationBase.cc* were intended to manage the parameters of data collection within the simulation. Thirdly, the changes made to *VeinsInetApplicationBase.ned* were changed to define network components. Changes to these files can be found in the documentation here `https://colab.research.google.com/drive/1h4uHLr-7xtSe7sRCCuHDYWvsBWt6hbtY?usp=sharing`.

8.3 .vec and .sca Results

With the changes made to the aforementioned files running the San Antonio downtown simulation will now produce two result files. Though these files contained the defined information they should contain they do not return any valuable data. If bug fixes is made that manages to fix that documentation any relevant information will be added to this research.

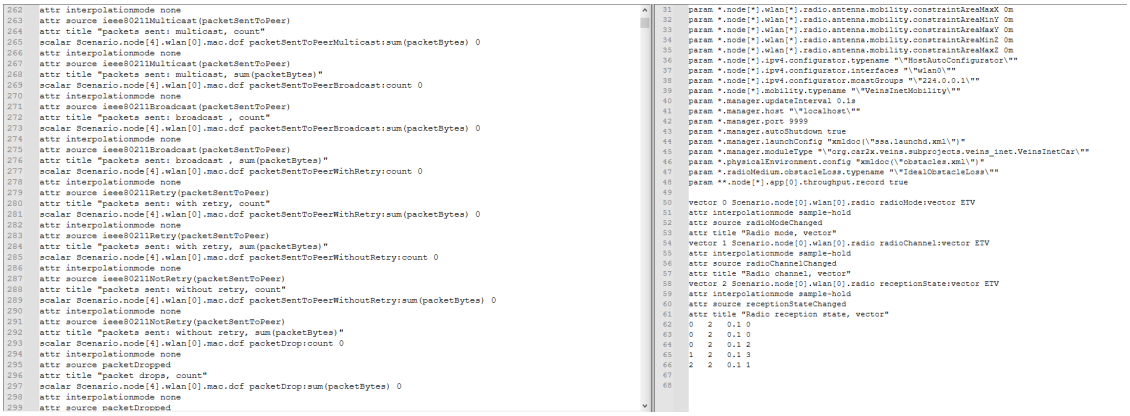

Figure 17: Raw data from the generated files; Right: .sca, Left: .vec

9 Conclusion and Future Work

This study demonstrates the potential of VANETs in solving urban traffic challenges specifically in busy, heavily populated areas like downtown San Antonio. The simulation of real-world traffic scenarios shows the effectiveness of VANETs in mitigating congestion and reducing accidents through enhanced communication in the form of V2V and V2I. Our network reveals that the integration of VANETs into urban traffic systems can improve overall efficiency and safety. This will continue to become more relevant with the prevalence of modern vehicles equipped with advanced sensor packages and communication hardware. It is likely roadways will continue to become safer with the implementation of VANETs.

Our comparisons with existing tools and research indicates the advantages of our approach to VANET simulation not only in its realism and scalability, but it's adaptability through original programmed scenarios. However, issues like growing infrastructure requirements and security concerns need to be addressed for these to be applicable in any real-world implementation. Exploration of potential bug-fixes and implementation of hard-coded network analysis within OMNeT++ simulations of VANETs would assist in creating more efficient simulations. Additionally, the creation of larger and more complex traffic scenarios including larger geographical areas and support for more complex vehicle types could help in creating more realistic and varied simulations. This study highlights the potential for VANETs to create more safe and efficient transportation systems in heavily-populated urban environments.

References

[1] F.D.Cunha, L. Villas, A. Boukerche, G. Maia, A. C. Viana, R. A. F.Mini, and A. A. F. Loureiro, "Data Communication in VANETs: Survey, Applications and Challenges", Ad Hoc Networks, pp.90-103, 2016. https://hal.science/hal-01369972/file/AdHocNetworks%20(1).pdf

[2] I. A. Aljabry and G. A. Al-Suhail, "A survey on network Simulators for Vehicular Ad-hoc Networks (VANETS)", International Journal of Computer Applications, 174(11), pp. 1-9, 2021. https://faculty.uobasrah.edu.iq/uploads/publications/1631546206.pdf

[3] M. A. Elgazzar, A. Alshareef, VANET Simulator: Full Design Architecture", International Journal of Engineering and Advanced Technology (IJEAT), vol. 9. no. 3, Feb 2020. https://www.ijeat.org/wp-content/uploads/papers/v9i3/C4784029320.pdf

[4] SUMO https://www.eclipse.org/sumo

[5] C. Tripp-Barba, A. Zaldívar-Colado, L. Urquiza-Aguiar, and J. A. Aguilar-Calderón, "Survey on Routing Protocols for Vehicular Ad-hoc Networks Based on Multimetrics", Electronics, vol. 8, no. 10, 2019. https://www.mdpi.com/2079-9292/8/10/1177

[6] OMNET https://omnetpp.org

[7] Veins https://veins.car2x.org/textbf

[8] Sommer C, Eckhoff D, Brummer A, Buse DS, Hagenauer F, Joerer S, Segata M, "Veins: The open source vehicular network simulation framework" Recent Advances in Network Simulation, pp.215–252. 2019. https://link.springer.com/chapter/10.1007/978-3-030-12842-5_6

[9] Weber, J., Neves, M. Ferreto, T. "VANET simulators: an updated review", J Braz Comput Soc 27, 8 (2021). https://journal-bcs.springeropen.com/articles/10.1186/s13173-021-00113-x

[10] P. Mooney and M. Minghini "A review of OpenStreetMap data" Mapping and the Citizen Sensor, pp.38. 2017. https://www.researchgate.net/profile/Marco-Minghini/publication/320878311_A_review_of_OpenStreetMap_data/links/5a0095150f7e9b62a151f9cf/A-review-of-OpenStreetMap-data.pdf

[11] OpenStreetMap https://www.openstreetmap.org/about

[12] "Introduction," JOSM https://josm.openstreetmap.de/wiki/Introduction

[13] "netconvert Documentation," SUMO https://sumo.dlr.de/docs/netconvert.html

[14] "duarouter Documentation," SUMO https://sumo.dlr.de/docs/duarouter.html

[15] "OMNeT++ 5.3 Release Notes," OMNeT+, Apr 2018. https://omnetpp.org/software/2018/04/12/omnet-5-3-released.html

[16] "The veins_launchd," Veins https://veins.car2x.org/documentation/sumo-launchd/

[17] "VANETTutorials," GitHub https://github.com/chaotictoejam/VANETTutorials