



UNIVERSITÀ DEGLI STUDI DI CATANIA  
DIPARTIMENTO DI MATEMATICA E INFORMATICA  
CORSO DI LAUREA MAGISTRALE IN INFORMATICA

**Analisi dei rischi e simulazioni con  
VEINS su reti VANET**

**- Tesi Sperimentale -**

Luca Steccanella

RELATORE

Chiar.mo. Prof. Giampaolo Bella

CORRELATORE

Dott. Pietro Biondi

Anno Accademico 2019/2020

---

# Indice

<b>1</b>	<b>Introduzione</b>	<b>2</b>
1.1	Le reti veicolari . . . . .	2
1.2	Tipologie di VANET . . . . .	3
1.2.1	V2V: Vehicle-to-Vehicle Communication . . . . .	3
1.2.2	V2I: Vehicle-to-Infrastructure Communication . . . . .	4
1.2.3	V2X: Vehicle-to-Everything Communication . . . . .	4
1.3	Obiettivi . . . . .	5
1.4	La scelta del simulatore . . . . .	5
<b>2</b>	<b>Lo standard IEEE 1609.4</b>	<b>7</b>
2.1	Panoramica . . . . .	7
2.2	Lo stack WAVE . . . . .	8
2.3	Operazioni crittografiche e di validazione . . . . .	10
2.4	Distribuzione dei certificati . . . . .	11
2.5	I messaggi WAVE . . . . .	11
<b>3</b>	<b>VEINS</b>	<b>13</b>
3.1	Panoramica . . . . .	13
3.2	OMNET++ . . . . .	15
3.2.1	C++ . . . . .	16
3.2.2	File NED . . . . .	19
3.2.3	FileINI e la simulazione risultante . . . . .	20
3.3	SUMO e TraCI . . . . .	22
3.4	I moduli di VEINS . . . . .	23

3.5	Installazione e configurazione . . . . .	23
3.5.1	Installazione di OMNET++ . . . . .	23
3.5.2	Installazione di SUMO . . . . .	25
3.5.3	Installazione di VEINS . . . . .	25
3.5.4	Avvio del server SUMO e utilizzo . . . . .	26
<b>4</b>	<b>Scenari di attacco</b>	<b>27</b>
4.1	Panoramica . . . . .	27
4.2	Falla zero day su ECU o automobile . . . . .	28
4.2.1	Veicoli malevoli con guidatori ignari . . . . .	28
4.2.2	Veicoli malevoli controllati dall'attaccante . . . . .	29
4.3	Invio di messaggi WAVE inaffidabili da parte dei nodi . . . . .	29
4.4	Jamming delle comunicazioni . . . . .	30
4.5	Attacco diretto alla RSU . . . . .	30
4.6	Attaccante esterno con ECU compromessa . . . . .	31
<b>5</b>	<b>Analisi e simulazioni</b>	<b>32</b>
5.1	Lo scenario scelto . . . . .	32
5.2	Analisi dell'esempio di Erlangen . . . . .	34
5.2.1	L'esempio . . . . .	34
5.2.2	I file presenti all'interno della simulazione . . . . .	34
5.2.3	Come strutturare il progetto . . . . .	36
5.3	La realizzazione della simulazione a Catania . . . . .	37
5.3.1	La mappa . . . . .	37
5.3.2	Configurazione di Sumo . . . . .	41
5.3.2.1	catania.rou.xml . . . . .	41
5.3.2.2	catania.sumo.cfg . . . . .	42
5.3.3	Configurazione di OMNET++ e VEINS . . . . .	43
5.3.3.1	omnetpp.ini . . . . .	43

---

5.3.3.2	catania.launchd.xml . . . . .	48
5.3.3.3	TraCICT11p.ned . . . . .	48
5.3.3.4	TraCICT11p.h . . . . .	49
5.3.3.5	TraCICT11p.cc . . . . .	50
5.4	Simulazioni . . . . .	56
5.4.1	Simulazione di Catania senza VANET . . . . .	57
5.4.2	Simulazione di Catania con VANET funzionante . . . . .	58
5.4.3	Simulazione di Catania con VANET compromessa . . . . .	60
<b>6</b>	<b>Conclusioni</b>	<b>63</b>
6.1	Risultati ottenuti . . . . .	63
6.2	Limitazioni dei simulatori . . . . .	64
6.3	Sviluppi futuri . . . . .	64
	<b>Bibliografia</b>	<b>66</b>
	<b>Ringraziamenti</b>	<b>68</b>

---

# Prefazione

Le automobili moderne stanno attraversando in questi anni una fase di digitalizzazione senza precedenti che continuerà negli anni futuri a farsi ancor più prominente. La prossima fase sarà quella dei veicoli interconnessi fra di loro portando alla realizzazione di scenari che ancora oggi sembrano irrealizzabili. Tali scenari presentano notevoli sfide di sicurezza, molte sono le proposte ed i protocolli, ma vi è ancora tanto lavoro da svolgere per gettare delle basi più solide. È obiettivo degli studiosi in sicurezza informatica far sì che le nuove tecnologie siano basate sull'idea di security-by-design, tutto ciò richiede un lavoro di simulazione, design, threat modeling e implementazione di soluzioni nuove ed efficaci. Lo scopo di questa tesi è gettare le basi per la realizzazione di modelli di attacco e di studiare scenari non ancora presi in considerazione dal mondo accademico. Tali basi e sfide a cui si sta preparando il mondo della cybersecurity realizzeranno le fondamenta di un futuro più sicuro per noi e chi ci sta intorno.

---

# 1

## Introduzione

### 1.1 Le reti veicolari

Le reti veicolari, o più precisamente VANET, acronimo che indica Vehicular Ad Hoc NETwork; derivano dalle MANET (Mobile Ad Hoc NETwork) [1]. Le VANET sono delle reti in cui i nodi oltre ad essere mobili possono muoversi a velocità sostenuta, poiché, seppure i dispositivi mobili siano appunto “mobili” la velocità di spostamento è diversa, realizzando di conseguenza una differenza sostanziale fra le due tipologie di rete. Le applicazioni delle VANET al momento sono quasi del tutto teoriche e fra le più disparate: semafori intelligenti, segnalazione traffico, segnalazione incidenti, segnalazione guasti, prevenzione traffico, prevenzione incidenti, guida autonoma e semi autonoma come i protocolli di platooning. Le VANET sono state standardizzate in vari protocolli come i messaggi CAM e i protocolli standard IEEE 801.11p e la sua evoluzione IEEE 1609.4; proprio su quest’ultimo sono stati basati lo studio e la simulazione, oggetto di questa tesi. Il protocollo IEEE 1609.4 è lo standard europeo corrente per lo sviluppo delle VANET future.

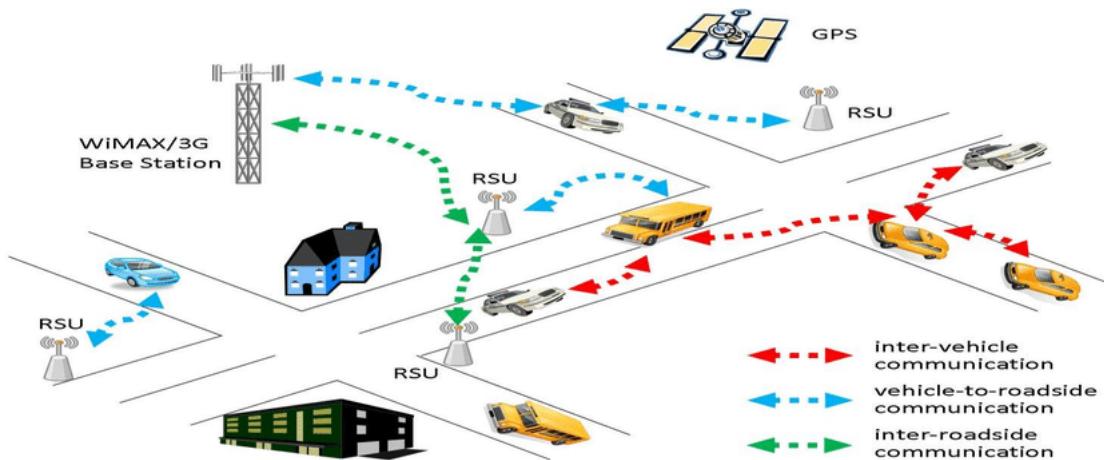


Figura 1.1: Rete VANET

## 1.2 Tipologie di VANET

Le VANET vengono categorizzate in tre tipi: V2V, V2I, V2X [2].

### 1.2.1 V2V: Vehicle-to-Vehicle Communication

In questa forma di comunicazione i veicoli comunicano direttamente fra loro senza alcuna terza parte che si frapponga fra gli stessi. È stata la prima forma di VANET, si basa sull'idea che i veicoli vicini possano comunicare fra loro al fine di far funzionare al meglio funzioni quali ABS, ESP e frenata di emergenza. Questo tipo di modello, sebbene abbia un approccio semplicistico può risultare efficace in quanto non dipende dall'esistenza di un'infrastruttura preesistente, d'altro canto presenta molte problematiche. Può funzionare soltanto quando i mezzi sono vicini fra di loro e supporta una serie limitata di funzioni, inoltre, la mancanza di un'infrastruttura può rendere le comunicazioni inaffidabili e complicate; si pensi alle difficoltà di individuare i veicoli vicini e stabilire delle comunicazioni veloci in tempi utili all'utente.

Su questo modello sono stati effettuati già molti studi, in quanto è il primo tipo di VANET ad essere stata teorizzata e sviluppata.

### 1.2.2 V2I: Vehicle-to-Infrastructure Communication

In questa modalità i veicoli anziché comunicare fra di loro comunicano con dispositivi ai bordi della strada. Tali dispositivi possono essere antenne che coordinano il traffico o semplicemente dispositivi quali semafori intelligenti. Questo sistema ha un approccio molto più tradizionale, sebbene sia necessario occuparsi anche di dispositivi che non siano i veicoli stessi, risulta più semplice implementare molte funzioni allorché difficili da supportare. Inoltre, registrare e tenere traccia dei veicoli in area è molto più facile, in quanto questo compito può essere imputato ad una antenna. Il modello V2I risulta essere più recente, ed inoltre, ma che sta rapidamente prendendo piede per i sopracitati motivi. Su questo si basano gli studi presenti in questa tesi.

### 1.2.3 V2X: Vehicle-to-Everything Communication

Questa metodologia è un ibrido delle due precedenti; in questo caso i veicoli comunicano sia con l'infrastruttura che con gli altri veicoli. Può servire per implementare soluzioni disparate e risolvere il problema dell'autenticazione e identificazione dei veicoli in zona. Può però diventare estremamente confusionario e difficile da implementare poiché questa forma di comunicazione a tre vie forma un'infrastruttura di rete poco chiara da definire in quanto il V2V si può ricondurre al P2P (Peer-To-Peer) e il V2I alle classiche reti a stella; il V2X invece è un ibrido fra le due cose che rischia di presentare più difetti di progettazione che vantaggi.

## 1.3 Obiettivi

L'obiettivo di questa tesi è gettare le basi del threat modeling relativo alle reti veicolari. Al giorno d'oggi lo stato degli studi applicativi sulla sicurezza di queste ipotetiche reti è sparso, incompleto e disorganizzato. La realizzazione di un'analisi dei rischi è il passo base per comprendere al meglio qualunque forma di studio successiva. La tipologia di VANET scelta per tale studio è la V2I poiché è quella su cui è disponibile meno letteratura ma che, altresì, sta rapidamente prendendo piede come valida opzione per la realizzazione delle VANET del futuro. Al fine di studiare al meglio tali rischi l'opzione migliore è quella di utilizzare un simulatore che permetta di comprendere la reazione di una VANET a determinati tipi di azioni, malevoli o meno che siano.

## 1.4 La scelta del simulatore

Il primo problema che si presenta in questo genere di tesi è la scelta del simulatore. È possibile accedere ad una gran quantità di simulatori di rete, anche gratuiti e molto noti, come NS2 e NS3. D'altro canto, data la delicatezza della nostra rete, che non è affatto comune, è necessario un simulatore che supporti le VANET, che sia configurabile in una macchina moderna, che non comporti costi eccessivi e che permetta di realizzare scenari in V2I/V2X dando un output chiaro e comprensibile. Sono state vagilate varie opzioni; innanzitutto i famosi NS2 e NS3 si sono rivelati troppo generici e troppo complessi per il compito scelto, dunque si è subito pensato all'opzione di utilizzare VEINS, software che sembra più votato allo scopo. Il grande difetto di VEINS sta nella documentazione quasi nulla e dalla flessibilità limitata nella realizzazione delle simulazioni, ho dunque valutato temporaneamente un suo fork chiamato Artery e un prodotto commerciale Tectos NetSim. NetSim sarebbe

stata una soluzione ottima, presenta un supporto pieno alle VANET, un’interfaccia semplice da usare e molta documentazione; questo prodotto avrebbe permesso una facile realizzazione di una moltitudine di scenari. Sfortunatamente, i costi di NetSim sono proibitivi e per questo motivo abbiamo deciso di migrare verso un altro simulatore. Artery, nasce dapprima come un plugin per Veins e successivamente è diventato un prodotto a sé stante. Nasce da un’iniziativa di Volkswagen AG per la realizzazione di un simulatore che supporti le VANET V2X. All’inizio i paper allegati e le funzionalità promesse da Artery sembravano promettenti, ma il progetto sembra aver raggiunto un punto morto, la documentazione è ancora più esigua e le simulazioni risultanti non danno un output chiaro, inoltre la realizzazione dello scenario desiderato appare complessa. Sono dunque ritornato a VEINS, che, nonostante le difficoltà iniziali ha permesso di realizzare qualche scenario esaustivo e di trarre risultati utili sulle mie supposizioni.

---

# 2

## Lo standard IEEE 1609.4

### 2.1 Panoramica

Lo Standard IEEE 1609.4 definisce come rendere sicuri i messaggi WAVE (Wireless Access in Vehicular Environments) facendo uso combinato di crittografia ed utilizzo di certificati; in maniera simile a SSL/TLS. Questo protocollo è un’evoluzione dello standard IEEE 801.11p [3]. Lo standard si prepone di fornire le linee guida per l’invio di messaggi che siano protetti da attacchi di intercettazione, manomissione, alterazione e replay. Fornendo dunque le primitive di autenticazione, identità e segretezza. Lo standard non fornisce istruzioni dirette su come implementare il protocollo ma fornisce altresì linee guida precise su come rendere sicure le comunicazioni, il formato dei dati e l’uso di tecniche che al momento sono riconosciute come stato dell’arte. Lo standard spiega dunque come va formato un messaggio, cosa cifrare, come usare i certificati e come condividerli. Di conseguenza, qualunque protocollo che implementi lo IEEE 1609.4 sarebbe sicuro dagli attacchi elencati precedentemente.

## 2.2 Lo stack WAVE

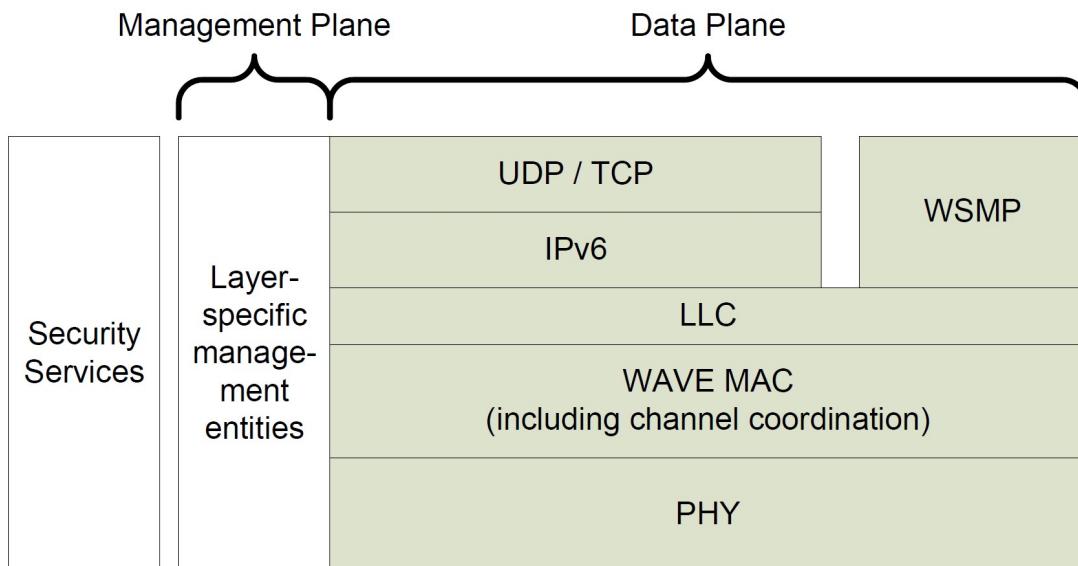


Figura 2.1: Stack Wave

Nello stack del protocollo viene definita la struttura del layer di comunicazione che di fatto si basa su protocolli già esistenti come il protocollo 801.11p, IP, TCP/UDP. Pertanto, non specifica con quale metodo tali dati vanno trasmessi ma altresì definisce la presenza dei WAVE security services che forniranno tutte le primitive di sicurezza descritte in precedenza. Lo schema successivo, in figura 2.2, può aiutare a chiarire ulteriormente questo concetto.

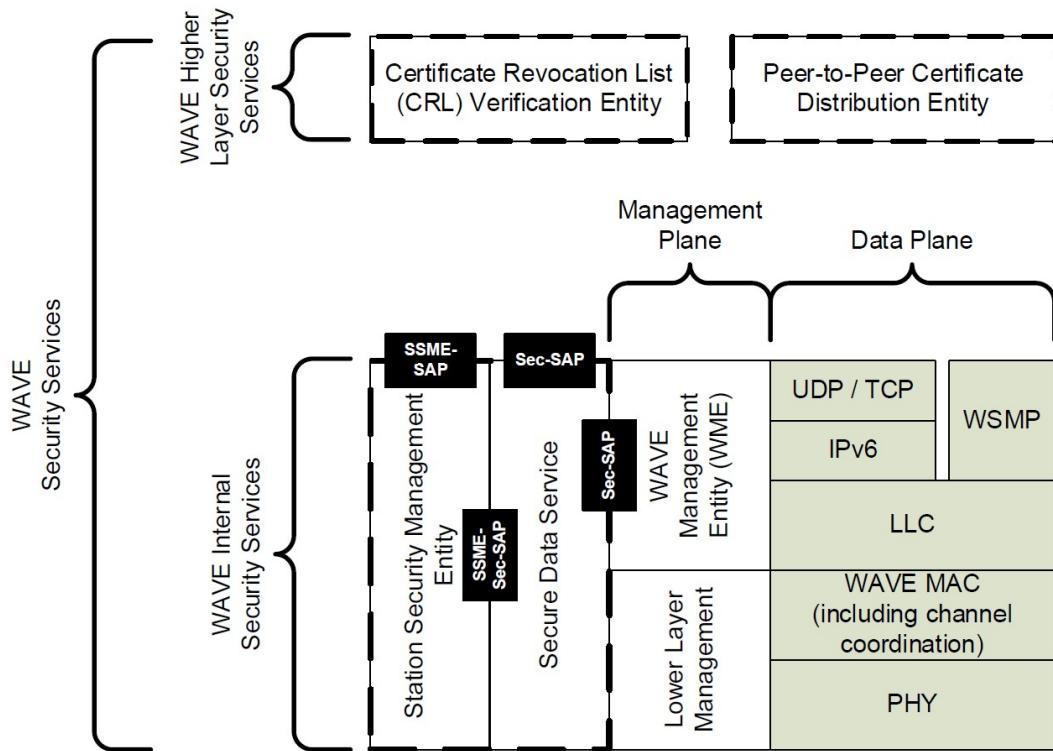


Figura 2.2: Wave security services

I servizi di sicurezza si dividono in quelli interni e quelli di alto livello. I servizi di sicurezza interni si occupano di rendere sicure le informazioni presenti nella sezione dati; in particolare si occuperanno di cifrare queste informazioni e decifrarle nel momento più opportuno al renderle fruibili al destinatario, in sicurezza. Inoltre, i servizi interni si occupano di gestire le informazioni ottenute dai certificati. I servizi di sicurezza di alto livello invece si occupano di gestire i certificati in sé; ossia prendere parte alla verifica e alla distribuzione degli stessi.

## 2.3 Operazioni crittografiche e di validazione

Lo IEEE 1609.4 definisce il certificato in modo tale da contenere [4]:

- Una chiave pubblica per effettuare le verifiche della firma digitale;
- I permessi ivi associati;
- Un'altra eventuale chiave pubblica per cifrare i dati;
- Identificativo di chi ha emesso il certificato;
- Informazioni che permettano di determinare se il certificato sia stato revocato o meno;
- Una dimostrazione crittografica che chi ha emesso il certificato abbia autorizzato il collegamento fra la chiave pubblica e le autorizzazioni.

I permessi che possono essere collegati consistono in:

- Validità geografica;
- Validità temporale;
- Permessi applicativi, ossia quali attività è autorizzato a fare chi possiede il certificato;
- Se chi possiede il certificato ne può emettere altri e quali;
- Se chi possiede il certificato ne può richiedere altri e quali.

Le operazioni crittografiche viene specificato l'uso degli Elliptic Curve Digital Signature Algorithm (ECDSA) e ricorda che al momento i due algoritmi di questo tipo che sono stati ufficialmente specificati sono il NIST P-256 e il brainpoolP256r1 [4].

## 2.4 Distribuzione dei certificati

Lo standard definisce la presenza di una Certificate Revocation List (CRL) e un'entità che verifichi quest'ultima. Tali componenti non vengono ulteriormente definite in quanto andavano fuori dallo scopo dello standard. Inoltre, viene specificato che il protocollo deve supportare la distribuzione Peer-To-Peer dei certificati, P2PCD (Peer To Peer Certificate Distribution). Tale metodologia permette di distribuire i certificati senza che un'unica parte si debba prendere carico dell'intero processo di distribuzione, ma altresì fare partecipare tutti i dispositivi WAVE al processo. Questa funzione si rivela molto utile, sia perché nel caso di una VANET V2V o V2X i veicoli comunicano direttamente fra loro, come in una rete P2P; ma anche perché nel caso di una rete V2I l'infrastruttura non ha una potenza di calcolo sufficiente ad effettuare tutte le operazioni da sola e ad ogni caso ogni singolo componente della stessa deve essere facilmente aggiornabile sullo stato dei certificati.

## 2.5 I messaggi WAVE

Oltre ai tecnicismi dettati dal protocollo è bene definire il contenuto dei messaggi WAVE. Le comunicazioni che avvengono utilizzando messaggi di tipo BSM (IEEE 801.11p) o WAVE (IEEE 1609.4) non inviano messaggi con una comunicazione diretta ma soltanto un set di informazioni standard.

Ad esempio, ogni nodo (veicolo) non invia comunicazione di incidente o altri problemi, ma invia in maniera costante agli altri elementi della rete aggiornamenti costanti sulla propria condizione, includendo dunque informazione quali velocità, posizione, attivazione di dispositivi di sicurezza, freni, etc... Tali messaggi dunque non sono modificabili e la responsabilità di segnalare un problema vero e proprio non ricade sul singolo nodo; questo ricevendo le informazioni dai veicoli in zona valuterà

cosa sta avvenendo intorno a sé. Se ad esempio riceverà segnalazione di più veicoli fermi in un punto, il nodo deciderà di segnalare all'utente di evitare la zona onde evitare la creazione di un ingorgo. Oppure, se troppi mezzi attivano l'ABS in un dato percorso, lo segnalerà come potenzialmente pericoloso. Questo tipo di metodologia è stata scelta per semplificare il formato dei messaggi e anche per evitare che un veicolo soltanto potesse segnalare per tutti un problema; responsabilità troppo alta che potrebbe dare ad un eventuale attaccante molto spazio di azione se riuscisse a intervenire sulla propria ECU. In questo modo invece, si ha un sistema che si avvicina al consenso, sono necessari più nodi per realizzare che ci sia un problema.

---

# 3

## VEINS

### 3.1 Panoramica

VEINS, acronimo di VEichle In Network Simulator, è un framework che contiene vari tool e software di simulazione che permettono di simulare delle VANET. Il framework è composto dai seguenti componenti:

- SUMO;
- OMNET++;
- TraCI;
- Moduli VEINS.

Dunque, è importante chiarire che Veins non è un simulatore a sé stante, ma un framework che si basa sì più simulatori e altre componenti per riuscire a simulare le VANET.

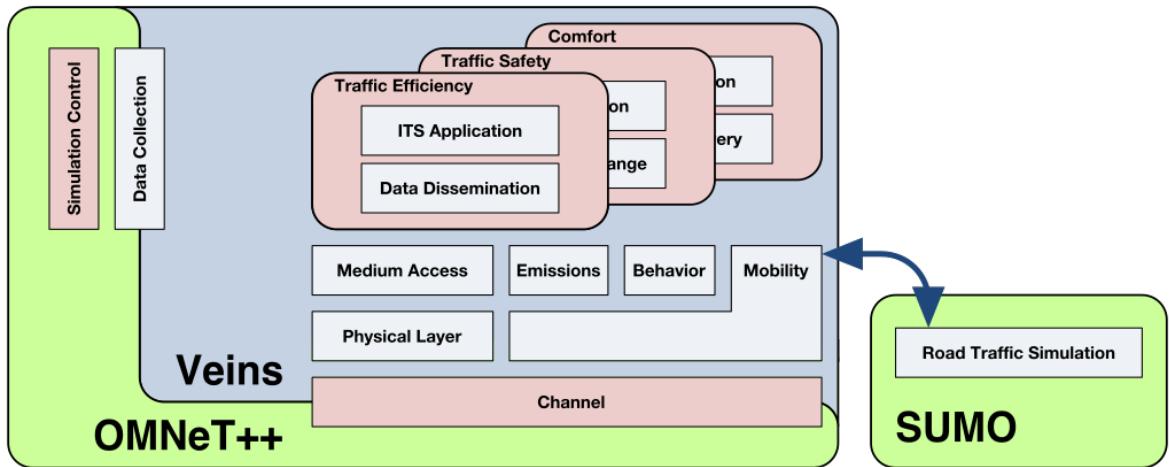


Figura 3.1: I componenti del VEINS Framework

Nello schema in figura 3.1 possiamo comprendere come questi componenti interagiscano fra di loro [5].

Si noti che il simulatore su cui si basa tutto il funzionamento è OMNET++, questo simulatore di eventi discreti permette di simulare reti, dunque viene posto alla base. Successivamente, i moduli aggiuntivi che vengono inseriti in OMNET++ forniscono i mezzi per simulare le VANET. Si noti come OMNET++ fornisca anche una IDE (Integrated Development Environment). TraCI viene utilizzato per far comunicare il blocco composto da OMNET++ con SUMO; quest'ultimo è un simulatore di traffico che, ricevendo l'input da OMNET++ avvia la simulazione del traffico risultante dando un output grafico e comprensibile all'utente finale. Le simulazioni di OMNET++ e SUMO, con questo sistema avvengono in tempo reale ed in contemporanea, sono da considerarsi dunque bidirezionali. VEINS presenta delle limitazioni, tra cui la documentazione, che è assente o di dubbia utilità nel caso delle poche indicazioni sparse in rete; dunque è stato necessario studiare in maniera singola ogni componente al fine di riuscire a configurare ed utilizzare il configuratore in maniera soddisfacente.

## 3.2 OMNET++

OMNET++ è un simulatore discreto di eventi modulare, il cui focus principale è la realizzazione di simulazioni di rete [6]. OMNET++ di per sé non è un simulatore di reti a sé stante ma fornisce un kernel di simulazione base ed un supporto estensivo alle interfacce grafiche. Questo lo rende estremamente modulare, difatti sul kernel è possibile caricare dei moduli scritti in C++ che poi verranno messi insieme da script scritti in linguaggio NED. Questi script fanno operare i moduli fra di loro e definiscono gli scenari di rete. Infine, la definizione di tutto lo scenario, di cosa va compilato e quali sono i file a cui va fatto riferimento viene inserito in un file di configurazione .ini. VEINS ad esempio porta con sé tanti moduli, che vengono compilati in OMNET++ e permettono la realizzazione di simulazioni in ambito VANET. Si noti che spesso, alla base di ogni simulazione si fa largo uso dei moduli INET, questi moduli infatti contengono i maggiori protocolli di rete e basandosi su questi ultimi è possibile realizzarne altri. INET infatti, come per altri progetti, è alla base di VEINS. Di seguito, in figura 3.2, l’interfaccia dell’IDE di OMNET++, basata su eclipse.

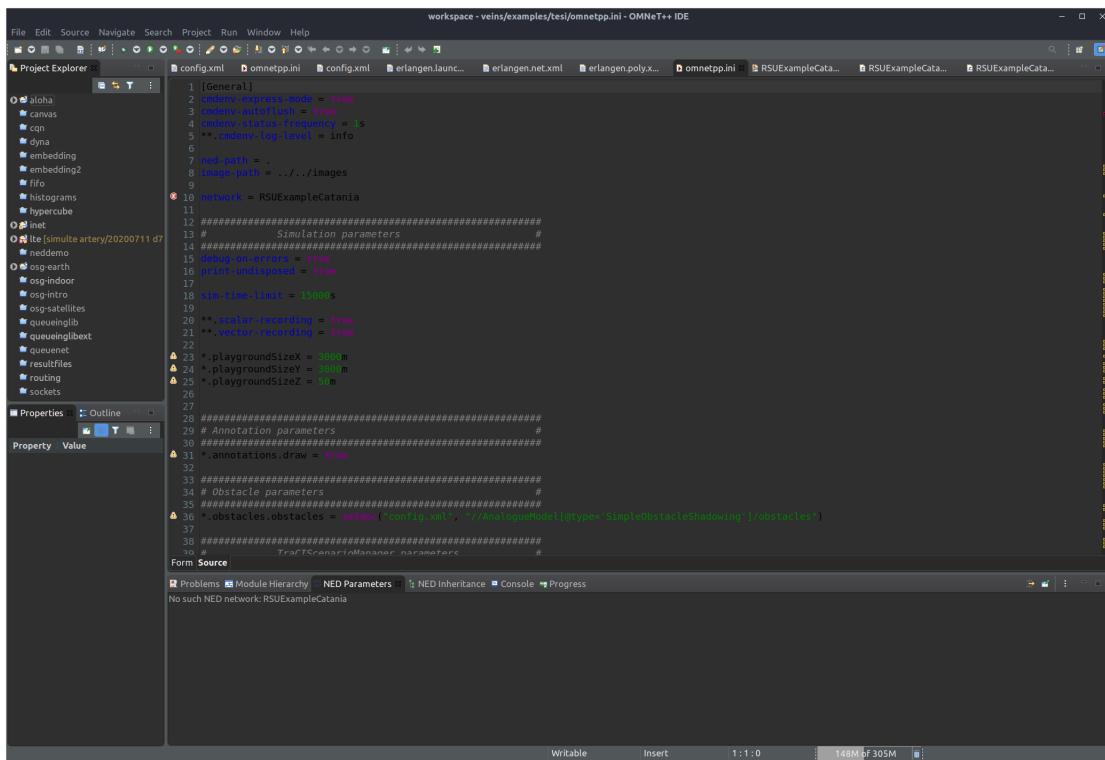


Figura 3.2: IDE di OMNET++

Al fine di comprendere meglio il funzionamento di OMNET++ di seguito illustrerò nel dettaglio dei file NED, CPP e INI di esempio; al fine di simulare due nodi che comunicano.

### 3.2.1 C++

Per cominciare dobbiamo definire i nodi, il formato del messaggio ed il comportamento del nodo relativo a quest'ultimo.

Di seguito il codice:

```

1 #include <string.h>
2 #include <omnetpp.h>
3
4 using namespace omnetpp;
```

```
5
6     class Txc3 : public cSimpleModule
7     {
8         private:
9             int i;
10
11         protected:
12             // The following redefined virtual function holds the
13             // algorithm.
14             virtual void initialize() override;
15             virtual void handleMessage(cMessage *msg) override;
16     };
17
18     // The module class needs to be registered with OMNeT++
19     Define_Module(Txc3);
20
21     void Txc3::initialize()
22     {
23         // Initialize is called at the beginning of the simulation.
24         // To bootstrap the tic-toc-tic-toc process, one of the modules
25         // needs
26         // to send the first message. Let this be 'tic'.
27         i=10;
28         WATCH(i);
29         // Am I Tic or Toc?
30         if (strcmp("tic", getName()) == 0) {
31             // create and send first message on gate "out". "tictocMsg" is
32             // an
33             // arbitrary string which will be the name of the message
34             // object.
35             EV << "Sending initial message\n";
36             cMessage *msg = new cMessage("tictocMsg");
37     }
```

```

33         send (msg, "out");
34     }
35 }
36
37 void Txc3::handleMessage(cMessage *msg)
38 {
39     i--;
40     if (i == 0){
41         EV << msg->getName();
42         EV << "'s counter depleted; deleting msg\n";
43     }
44     else{
45         EV << msg->getName() << "'s counter reached ";
46         EV << i << " sending another msg.";
47         send (msg, "out"); // send out the message
48     }
49 }
```

Listing 3.1: Esempio di OMNET C++

Nel file C++ è necessario innanzitutto includere l'header ometpp.h che permette di importare tutte le funzionalità di OMNET++. Successivamente creiamo la classe Txc3 che eredita da cSimpleModule, quest'ultima classe, inclusa in OMNET++, fornisce lo scheletro per la creazione di un modulo.

Nel nostro caso vogliamo creare un tipo di nodo che invia messaggi in un dato intervallo di tempo e che sia anche in grado di riceverli, inviando una notifica all'utente. Inoltre, vogliamo che tenga un contatore di messaggi, in modo che ogni dieci messaggi ricevuti smetta di rinviare altri messaggi e cancelli quello ricevuto.

Tale comportamento viene definito nei metodi initialize() e handleMessage(...). Tali metodi sono degli override di metodi nella classe madre, che come ricordiamo fornisce già lo scheletro per un modulo OMNET++.

Il metodo `initialize()` viene chiamato in automatico all'avvio della simulazione, quest'ultimo inizializzerà il contatore, creerà un messaggio iniziale e lo invierà all'altro nodo. Si noti anche che in questo metodo il nodo capirà se è il nodo `tic` o il nodo `toc`. Nel metodo dedicato alla gestione del messaggio, che viene ricevuto come argomento, innanzitutto viene scalato il contatore, se poi questo raggiunge lo zero il messaggio verrà eliminato; altrimenti verrà rimandato segnalando anche lo stato del contatore.

### 3.2.2 File NED

Il file NED serve a definire i parametri del modulo all'interno della simulazione, definire il network, i sotto moduli che ne faranno parte ed il loro comportamento.

Di seguito il codice:

```

1 simple Txc3
2 {
3     parameters:
4         @display("i=block/routing"); // add a default icon
5     gates:
6         input in;
7         output out;
8     }
9
10 //
11 // Two instances (tic and toc) of Txc1 connected both ways.
12 // Tic and toc will pass messages to one another.
13 //
14 network Tictoc3
15 {
16     submodules:
17         tic: Txc3 {
```

```

18         parameters:
19             @display(" i=gold");
20         }
21     toc: Txc3 {
22         parameters:
23             @display(" i=cyan");
24     }
25     connections:
26         tic.out --> { delay = 100ms; } --> toc.in;
27         tic.in <-- { delay = 100ms; } <-- toc.out;
28 }
```

Listing 3.2: Esempio di linguaggio NED

Con la parola chiave *simple* definiamo le caratteristiche di base del modulo Txc3, in questo caso specifichiamo quale deve essere l'icona di default e i gates, ossia i punti di input e output del modulo; nel nostro caso ne specifichiamo uno per l'input e uno per l'output.

Con la parola chiave *network* definiamo la nostra rete Tictoc3, in cui inseriamo due moduli Txc3 dandogli un colore diverso (ciano e oro) e le connessioni in uscita e in entrata. Si può notare come usando i gate creati in precedenza specifichiamo la direzione della connessione, includendo un ritardo di 100ms.

### 3.2.3 File INI e la simulazione risultante

Una volta completato il lavoro sui file NED e CPP definiamo l'INI.

```

1 [General]
2 network = Tictoc3
```

In questo caso il file INI è molto semplice, in quanto è necessario definire soltanto la rete che vogliamo simulare, che Tictoc3, ossia la rete che abbiamo definito nel file NED. Una volta avviata la compilazione, OMNET++ leggerà nell'ini la rete da

simulare, che verrà cercata nel file NED e da quest'ultimo verrà cercato e compilato il codice C++ dei moduli specificati.

Una volta completata la compilazione la simulazione verrà avviata all'interno della GUI, come è possibile vedere in figura 3.3.

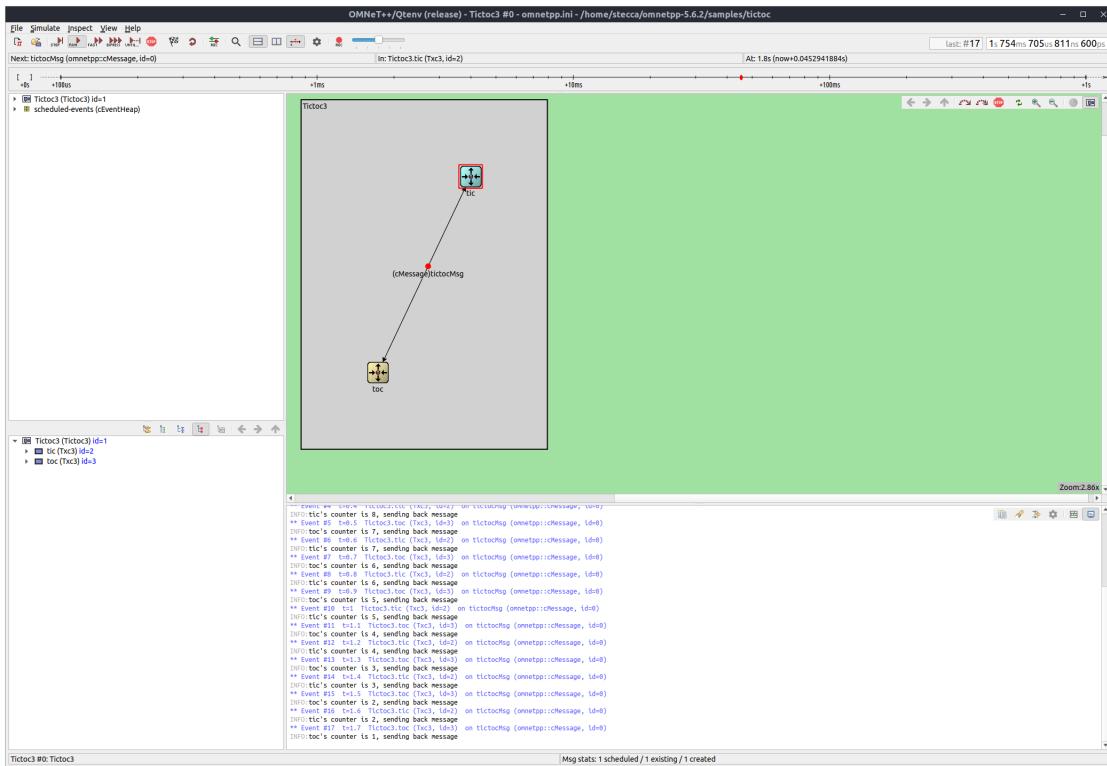


Figura 3.3: Esempio di simulazione nella GUI di OMNET++

### 3.3 SUMO e TraCI

SUMO, acronimo di Simulation of Urban MObility [7], è un simulatore di traffico opensource. Permette di simulare in maniera completa e realistica il comportamento del traffico in qualunque mappa reale o meno.

Tramite dei documenti XML è possibile creare strade, aggiungere elementi e veicoli, anche diversi fra di loro. Inoltre, le mappe possono essere importate da openStreetMap, automatizzando il processo.

Permette di simulare non solo il movimento dei veicoli, ma anche eventi come traffico e incidenti, inoltre, simula bene componenti comuni come i semafori. Tutti questi elementi possono essere aggiunti manualmente nella configurazione XML, ma possono essere importati insieme alla mappa.

Queste simulazioni avvengono in un'interfaccia grafica che permette di visualizzare il comportamento di ogni veicolo, la segnaletica e via dicendo. La simulazione può anche essere registrata, messa in pausa o velocizzata.

Inoltre, un componente molto importante è TraCI, ossia TRAffic Control Interface [8]; questo componente permette di ottenere i valori della simulazione e soprattutto di manipolare il comportamento di quest'ultima da fuori. Questo componente si basa su un'architettura TCP e permette a qualunque applicativo di accedere a SUMO.

Nel nostro caso, VEINS si connette a SUMO tramite TraCI; quest'ultimo gli permette di avviare la simulazione in SUMO e inviare il comportamento dei veicoli. Si noti dunque che nel nostro caso SUMO è abbastanza passivo, e si limita a visualizzare il risultato di una simulazione creata tramite un altro software.

## 3.4 I moduli di VEINS

Come già detto in precedenza VEINS contiene vari moduli che si trovano in **/src/veins/modules**; nel nostro caso le sorgenti di maggiore interesse si trovano in **/src/veins/modules/application**, qui infatti troviamo il codice per i moduli base 802.11p (sui quali si basano le simulazioni anche per 1609.4) e nella cartella **/traci** il codice sorgente che permette la realizzazione di una simulazione VANET, viene infatti fornito anche il codice barebone su quale deve essere basata ed è qui che va inserito il codice sorgente di una eventuale nuova simulazione. Inoltre, sempre nella cartella **/src/veins/modules/**, troviamo in **/messages** troviamo le frame di 801.11p e 1609.4, **/mac** i mac, in **/nic** le interfacce di rete e in **/phy** il livello fisico.

## 3.5 Installazione e configurazione

L’installazione e la configurazione dell’intero framework presenta alcune difficoltà dovuta sia alla documentazione che lascia davvero molto a desiderare sia ad alcune incompatibilità con i sistemi operativi più recenti. Per l’uso di Veins, nonostante sia multipiattaforma, viene consigliata l’installazione su un sistema Linux. Si noti che è disponibile al download una macchina virtuale pronta all’uso, quest’ultima è stata però scartata per fattori di praticità e prestazioni. È stato dunque scelto di installare VEINS sul sistema operativo Ubuntu 20.04; per farlo è stato necessario elaborare un metodo di installazione chiaro e funzionante.

### 3.5.1 Installazione di OMNET++

Innanzitutto, è necessario installare i prerequisiti con il comando:

```
1 $ sudo apt-get install build-essential gcc g++ bison flex perl python
      python3 qt5-default libqt5opengl5-dev tcl-dev tk-dev libxml2-dev
      zlib1g-dev default-jre doxygen graphviz libwebkitgtk-3.0-0
```

Sfortunatamente molti pacchetti non sono più disponibili nelle repo ufficiali di Ubuntu 20.04, dunque è stato necessario importare le vecchie repo ed installare alcuni deb manualmente, risolvendo dunque le dipendenze. È necessario inoltre aggiungere altre repo e installare altri pacchetti.

```
1 $ sudo add-apt-repository ppa:ubuntugis/ppa  
2 $ sudo apt-get update  
3 $ sudo apt-get install openscenegraph-plugin-osgearth libosgearth-dev
```

E anche i seguenti per supportare meglio le simulazioni in parallelo

```
1 $ sudo apt-get install openmpi-bin libopenmpi-dev
```

Successivamente è necessario scaricare OMNET++ dal sito ufficiale [9] e scomprimere l'archivio targz dove si desidera con il comando tar.

```
1 $ tar xvfz omnetpp-5.6.1-src.tgz
```

Dunque, è necessario spostarsi nella cartella di OMNET++ e fare il sourcing dello script setenv, che è necessario al fine di inserire alcune sottocartelle importanti nella path di sistema.

```
1 $ cd omnetpp-5.6.1  
2 $ . setenv
```

Successivamente è necessario aggiungere la cartella dei bin di OMNET++ nella path dell'utente modificando il file .bashrc (se si usa bash).

```
1 $ nano ~/.bashrc
```

Aggiungendo alla fine del file

```
1 export PATH=$HOME/omnetpp-5.6.1/bin:$PATH
```

Dopo possiamo finalmente configurare e installare OMNET++

```
1 $ ./configure  
2 $ make
```

Una volta completata la compilazione possiamo avviare OMNET++

```
1 $ omnetpp
```

Infine, possiamo anche creare le icone nel desktop e nel launcher con i comandi

```
1 $ make install-menu-item  
2 $ make install-desktop-icon
```

### 3.5.2 Installazione di SUMO

Fortunatamente l'installazione di sumo è molto più rapida, in quanto è disponibile un ppa sempre aggiornato.

```
1 $ sudo add-apt-repository ppa:sumo/stable  
2 $ sudo apt-get update  
3 $ sudo apt-get install sumo sumo-tools sumo-doc
```

Può essere dunque eseguito con il comando *sumo*, o *sumo-gui* per avviare la gui.

### 3.5.3 Installazione di VEINS

Per installare Veins è necessario scaricarlo dal sito ufficiale o dalla repo ufficiale su GitHub [10] ed estrarre l'archivio a piacimento. Avviare dunque di nuovo OMNET++ creare una cartella workspace con il wizard iniziale ed importare Veins cliccando su *File > Import > General: Existing Projects into Workspace* e selezionando la cartella di Veins.

Successivamente effettuare la build di veins cliccando in *Project > Build All*.

### 3.5.4 Avvio del server SUMO e utilizzo

Per utilizzare Veins adesso sarà sufficiente sfruttare i nuovi moduli come già illustrato in precedenza tramite i file CPP, NED edINI. Una volta effettuata la build, se questa avrà successo, verrà avviata la simulazione sia su OMNET++ che su SUMO. È necessario però avviare il server SUMO con alcuni parametri particolari, è dunque è necessario sfruttare uno script in python già incluso in Veins specificando però la posizione del binario di Veins. Tale server resterà continuamente in ascolto, pronto ad avviare una simulazione qualora Veins lo richieda. Per semplificare il processo ho creato un piccolo script in bash che avvierà quello in python con i parametri corretti.

```
1 #!/bin/bash  
2 ./sumo-launchd.py -vv -c sumo-gui  
3 exit 0
```

Si noti che il comando eseguito è `textit{sumo-gui}`, in modo tale da avviare la GUI di SUMO e visualizzare la simulazione ogni volta che viene avviata da VEINS.

---

# 4

## Scenari di attacco

### 4.1 Panoramica

Dati i protocolli, la VANET e gli strumenti di simulazione; ci siamo concentrati sull’ipotizzazione di vari modelli di attacco al fine di comprendere sia quale poteva essere simulato sia di gettare le basi al threat modeling sulle VANET di tipo V2I. Tali scenari sono frutto di varie considerazioni sul protocollo e su come viene usato. È bene notare che il protocollo di per sé non è vulnerabile ai tipici attacchi di intercettazione, manomissione, alterazione e replay. Dunque, è necessario minare l’infrastruttura in altri modi volti a comprometterne le funzionalità. Prima di proseguire si noti inoltre che con il termine nodo ci si riferisce ad un veicolo che circola e con il termine RSU (RoadSide Unit) ci si riferisce ad un’antenna facente altresì parte dell’infrastruttura. Nel nostro caso ogni nodo si registra tramite la RSU della sua zona; quest’ultima tiene facilmente traccia della posizione di ogni veicolo e conosce i veicoli in zona. Ogni nodo manda un messaggio WAVE periodico alla RSU e quest’ultima lo rilancia ad ogni nodo registrato.

Si ricordi che per il funzionamento dei messaggi WAVE la segnalazione non è imputata al singolo nodo, invece quest’ultimo traendo informazioni da più nodi determinerà se c’è un problema da segnalare al proprio guidatore o meno. Gli scenari che sono stati ipotizzati sono votati a minare proprio questo meccanismo, per farlo si

tenterà di minare le elaborazioni del singolo nodo o compromettere le comunicazioni.

## 4.2 Falla zero day su ECU o automobile

In questo scenario ci si concentra principalmente sui veicoli che comunicano all'interno della VANET. L'ipotesi è quella che, dati veicoli compatibili con il protocollo WAVE alcuni abbiano una falla che se sfruttata da un attaccante permette di far sì che il nodo vittima non aderisca in maniera costruttiva al protocollo e all'infrastruttura. Si suppone infatti che l'effetto di tale attacco è che il nodo non sia più in grado di ottenere i messaggi WAVE dell'infrastruttura o che comunque ne fermi l'elaborazione da parte dello stesso. Sempre tramite questa falla si può ipotizzare che il nodo segnali problemi inesistenti al proprio guidatore, compromettendone la sicurezza. Si evince che, in questo tipo di attacco riusciamo a minare le funzionalità dell'infrastruttura puntando sulla fase finale, poiché riuscendo a minare l'affidabilità delle informazioni fornite all'utente o le annulliamo, vanifichiamo l'infrastruttura rendendola inutile o inaffidabile.

Ad ogni modo infettare un solo nodo non sarà sufficiente a creare una quantità di confusione che impatti anche gli altri nodi, dunque è bene definire dei metodi di infezione nei successivi sotto paragrafi.

### 4.2.1 Veicoli malevoli con guidatori ignari

Si presupponga la necessità di attaccare i nodi e le ECU in maniera diretta poiché sono in grado di accedere alla rete VANET tramite certificato. In questo caso la falla viene sfruttata dall'attaccante per colpire più veicoli dello stesso tipo, sia con una manomissione effettuata di persona o più presumibilmente lasciando che un singolo nodo infetto, tramite le comunicazioni di servizio, contaghi i veicoli nella sua zona che hanno stessa marca e produttore ossia la stessa falla. In questo caso avremo più

guidatori ignari che parteciperanno all’infrastruttura senza trarne vantaggio, questo probabilmente vanificherebbe lo scopo dell’infrastruttura e creerebbe problemi anche a tutti gli altri guidatori non infetti.

È bene però notare che in questo caso la propagazione dell’infezione non è del tutto controllabile dall’attaccante e il risultato dipende tutto da variabili aleatorie, come il numero di veicoli infettibili e il loro percorso. Tali veicoli potrebbero essere vicini e dunque minare il consenso, oppure, allontanarsi fra di loro e non compromettere in maniera notevole l’infrastruttura.

#### 4.2.2 Veicoli malevoli controllati dall’attaccante

In questo scenario invece i veicoli infetti vengono controllati e immessi nel traffico di proposito da un’unica entità malevola. In questo caso dunque, i guidatori non sono ignari e si muovono, in forze, in maniera coordinata, negando il funzionamento dell’infrastruttura nella zona desiderata. Questo tipo di attacco può risultare ancora più pericoloso ed ha la massima probabilità di creare problemi a tutta l’infrastruttura e dunque ai nodi non infetti.

Si noti dunque, che sebbene questo tipo di attacco non colpisca il protocollo o l’infrastruttura in maniera diretta riesca a minarne sicurezza e affidabilità per chiunque vi partecipi.

### 4.3 Invio di messaggi WAVE inaffidabili da parte dei nodi

Partendo sempre dalla supposizione che è possibile creare un nodo malevolo sfruttando una falla nelle ECU degli stessi, possiamo aggiungere nel precedente scenario un ulteriore livello di compromissione di quest’ultima. Tale livello di compromis-

sione farà sì che il nodo invii messaggi WAVE con contenuto errato all'RSU che verranno rilanciati agli altri nodi.

Si noti che l'attacco funzionerebbe poiché il nodo accede regolarmente alla RSU in maniera legittima.

D'altro canto, una V2I basata sui messaggi WAVE si basa sul consenso e dunque un solo nodo non avrebbe grande effetto, sarà dunque necessario far sì che più nodi assumano lo stesso comportamento e che sia l'infezione e i messaggi inviati siano coordinati da un unico attaccante. Solo con quest'approccio sarà possibile indurre i nodi non infetti a comportarsi in maniera errata e soddisfare gli scopi dell'attaccante.

## 4.4 Jamming delle comunicazioni

Le comunicazioni effettuate dalle VANET usano metodi di comunicazione già conosciuti e comuni, come il WIFI, 4G e 5G. Questi metodi di comunicazione sono proni a interruzioni e interferenze e di conseguenza utilizzare un apposito jammer taglierebbe le comunicazioni in una data area e lascerebbe sia la RSU che i nodi completamente al buio, incapaci di comunicare. Questo tipo di attacco, sebbene molto generico, si può rivelare molto adatto ad un eventuale negazione del servizio fornito dalla VANET.

## 4.5 Attacco diretto alla RSU

Nessuna contromisura di sicurezza viene specificata per la RSU nello standard, questa la rende ipoteticamente prona ad una manomissione manuale. Se non vengono adottate valide misure di sicurezza accedendo fisicamente all'antenna si potrebbe compromettere tutta l'infrastruttura. Un attaccante in questo caso potrebbe fermare del tutto le comunicazioni, accedere ai nodi registrati e disconnettere o com-

promettere le informazioni inviate ad una porzione degli stessi o in tutta l'area. Inoltre, connettersi liberamente alla RSU in questo modo fornirebbe molti dati utili per un eventuale information gathering. Questo genere di attacco si può rivelare molto efficace nell'effettuare una compromissione mirata di una zona, strada oppure per attaccare il veicolo di uno specifico target. Ad ogni modo è bene notare che in questo caso l'attaccante non dovrebbe essere in grado di forgiare messaggi WAVE o generarne nuovi; questo perché la RSU non dovrebbe essere in grado di decifrare i messaggi né autorizzata a crearne nuovi, ma solo rilanciare quelli già esistenti.

## 4.6 Attaccante esterno con ECU compromessa

In questo scenario di attacco possiamo ipotizzare che l'attaccante riesca ad utilizzare una ECU compromessa per creare un dispositivo atto ad accedere alla RSU e compromettere l'infrastruttura. Si ipotizza che accedendo alla ECU ed ai certificati riesca a trovare un modo di autenticarsi liberamente. L'attaccante riuscirebbe dunque a registrare nodi a piacimento e immettere messaggi WAVE nella rete. Così facendo l'attaccante potrebbe creare ingorghi o altri problemi fittizi o sovraccaricare l'infrastruttura generando un numero ingestibile di nodi o messaggi.

Qualora quest'attacco fosse combinato con il precedente, l'attaccante avrebbe un controllo completo della VANET in quanto accederebbe ad ogni informazione, sarebbe capace di gestire i nodi a piacimento e di inviare ogni tipo di messaggio desiderato.

---

# 5

## Analisi e simulazioni

### 5.1 Lo scenario scelto

Prima di proseguire con una simulazione è stato necessario scegliere lo scenario più adatto da simulare. Questa scelta è stata principalmente fatta in base alle capacità di Veins, infatti il simulatore non è capace di simulare liberamente ogni tipo di scenario poiché i moduli che fornisce sono scritti per comportarsi esattamente in un dato modo. Ad esempio, non è ammesso che un nodo non funzioni come dovrebbe o che non aderisca bene al protocollo, dunque simulare qualcosa che vada fuori dagli schemi risulta molto complicato. Replicare scenari di attacco, o scenari in cui il comportamento della rete vada storto va fuori dagli schemi di VEINS e SUMO, dunque è necessario operare in maniera più indiretta al fine di riuscire a simulare dei comportamenti scorretti che possano portare ad una rete malfunzionante. Lo scenario scelto a tal proposito è simile al 4.2.1; vogliamo dapprima simulare il traffico nei dintorni di Tondo Gioeni, ossia una porzione di Catania. In questa zona, date le sue caratteristiche e la grande mole di veicoli spesso si formano ingorghi e lunghe code, si procederà dunque creando una VANET V2I in grado di prevenire quest'ultimi problemi.



Figura 5.1: Ingorgo nel tondo Gioeni di Catania

Successivamente, una volta appurata l'efficacia della VANET nel limitare gli ingorghi; si procederà nell'inserire, in maniera aleatoria e via via crescente dei nodi infetti che ignoreranno i messaggi WAVE ma che saranno anche in grado di indicare di essere state coinvolte in un incidente, ovviamente mentendo. Grazie alla simulazione potremo capire gli effetti di questo genere di attacco, che percentuale di infezione sia necessaria per ottenere effetti rilevanti e rilevare dunque in generale la resilienza del protocollo 1609.4 nei confronti di questo genere di situazione.

## 5.2 Analisi dell'esempio di Erlangen

### 5.2.1 L'esempio

Data la quasi totale assenza di documentazione, è stato necessario tentare di comprendere l'uso del sistema di simulazione sfruttando l'unica risorsa fornita dagli sviluppatori a tale scopo, una simulazione di traffico in una porzione di Erlangen, una cittadina tedesca. La simulazione imita una VANET che fa uso di messaggi BSM/WAVE che vengono rilanciati da una RSU. Lo scopo della VANET è quella di aggiornare tutti i nodi sullo stato degli altri, prevenendo incidenti e ingorghi. Tale esempio si adatta con il nostro caso di studio e ciò aiuta a comprendere come realizzare la simulazione. Innanzitutto, è presente una gran quantità di file xml, sfogliandoli si capisce che VEINS e sumo necessitano di alcuni file di configurazione xml per funzionare al meglio.

### 5.2.2 I file presenti all'interno della simulazione

Si nota inoltre che non sono presenti file C++. La motivazione si evince guardando il file .ini, l'omnet.ini in questo caso appare molto più complesso e denso di parametri rispetto al normale. Innanzitutto, sono presenti dei regolari parametri di simulazione, come la dimensione del riquadro di simulazione e le tempistiche; d'altro canto sono presenti molti parametri che non sono propri di omnet. In omnet.ini infatti vengono definiti i moduli di veins necessari nella configurazione ed è dunque possibile impostarne vari parametri come la potenza, la posizione e quanto tempo passa fra l'invio di un beacon e il successivo. Inoltre, deve anche essere specificato il nome dell'applicazione, ossia il codice C++ che indicherà la reazione dei nodi alla ricezione dei messaggi BSM/WAVE. Tale codice si deve trovare in veins/src/modules/application/traci; questo significa che non siamo noi a scrivere un modulo per un

nodo o di una RSU, ma definiamo solo una porzione di comportamento univoca per tutti i nodi e non abbiamo possibilità di intervento sulla RSU; che resta relegata alla funzione di rilancio. Ad avvalorare questa ipotesi vi è il file ned, che risulta invece molto minimale, infatti viene solo importato in modulo della RSU per definirne il colore e null'altro. Dunque, Veins funziona in maniera inaspettata e la realizzazione di simulazioni si differenzia nettamente rispetto al paradigma di OMNET++; questo, sebbene fornisca un ambiente failsafe, non permette grosse modifiche agli scenari e limita di molto le possibilità poste di fronte allo sviluppatore. Quest'ultimo infatti sarà limitato solo a definire le specifiche degli impianti usati, il loro posizionamento e la reazione univoca di tutti i nodi alla ricezione di un messaggio; senza potere invece modificare in alcun modo il comportamento della RSU. Inoltre, nel file omnet.ini si fa riferimento anche alla configurazione di TraCI, dove viene indicato indirizzo e porta del server in ascolto. Infine, nel file omnet.ini troviamo i riferimenti ad alcuni dei file xml; nei file antenna.xml e config.xml vengono definite alcune configurazioni relative alle impostazioni di invio e ascolto di antenna e ricevitori. Un altro file a cui viene fatto riferimento è erlangen.launchd.xml, che contiene le impostazioni di lancio per sumo. In questo xml, si fa riferimento ai quattro xml restanti, guardiamo innanzitutto erlangen.sumo.cfg, questo è il file di configurazione della simulazione sumo; qui vengono specificate varie impostazioni di report, gui, etc... Ma soprattutto vengono definiti i compiti dei restanti xml.

Il blocco di codice:

```
1 <net-file value="erlangen.net.xml"/>
```

Ci dice che i file \*.net.xml definiscono la rete stradale, infatti possiamo comprendere dal codice che vengono definiti vari punti di incontro e corsie, ognuno ha il suo id, lunghezza, forma e sono differenziate in base alla tipologia.

```
1      <edge id="12247702_4" function="internal">
```

```

2      <lane id=":12247702_4_0" index="0" speed="13.89" length="5.33"
3          shape="644990.55,5493966.28 644989.03,5493966.20
4              644987.98,5493966.65 644987.39,5493967.64 644987.26,5493969.15"/>
5      </edge>
```

Il blocco di codice:

```
1 <route-files value="erlangen.rou.xml"/>
```

Suggerisce che i file \*.rou.xml contengono i percorsi da simulare. Aprendo il file xml si può vedere infatti la definizione di un tipo di veicolo tramite la parola chiave vType, dove sono specificate caratteristiche come velocità e accelerazione.

```
1 <vType id="vtype0" accel="2.6" decel="4.5" sigma="0.5" length="2.5"
2     minGap="2.5" maxSpeed="14" color="1,1,0"/>
```

Tramite l'uso della parola chiave route viene invece specificato un percorso, indicandone gli id degli edge che ne fanno parte.

```
1 <route id="route0" edges="-39539626 -5445204#2 ... 4900041#1"/>
```

Infine, con la parola chiave flow, viene definito un flusso che indica quanti veicoli di un dato tipo devo muoversi su una data route.

```
1 <flow id="flow0" type="vtype0" route="route0" begin="0" period="3"
2     number="195"/>
```

Resta solo il file \*.poly.xml, in questi file di configurazione vengono inseriti gli edifici, specificandone id, forma e colore.

### 5.2.3 Come strutturare il progetto

Dunque, abbiamo finalmente compreso come strutturare il progetto. Non possiamo scrivere il comportamento di nodi o RSU sotto forma di moduli, il C++ dunque ci servirà solo per scrivere la reazione dei nodi ai messaggi. La RSU è preimpostata. Il ruolo del file ned è marginale, serve solo a definire lo stile e posizione della RSU. Il

file omnet.ini diventa molto importante, dobbiamo configurare qui le specifiche dei nodi, della rsu e della simulazione. Sono inoltre presenti molti file xml, alcuni sono necessari per il funzionamento di sumo; altri due invece, config.xml e antenna.xml, sono necessari a definire alcune configurazioni relative alle impostazioni di invio e ascolto di antenna e ricevitori; che fortunatamente sono piuttosto standard. Nel file \*.sumo.cfg dovremo mettere la configurazione della simulazione di sumo includendo anche il nome dei seguenti file xml, che andranno configurati opportunamente.

- \*.net.xml, in questo file di configurazione andranno inserite le strade che formano la rete stradale;
- \*.rou.xml, in questo file xml vanno definiti i veicoli e i percorsi che intraprenderanno;
- \*.poly.xml, qui verranno definiti gli edifici e i poligoni presenti nella mappa.

I file \*.net.xml e \*.poly.xml sembrano molto complessi se si ha l'obiettivo di definire una porzione di mappa derivata da una città reale; fortunatamente, come vedremo dopo, è disponibile un tool che genererà automaticamente questi file. Per farlo sarà sufficiente fornire la porzione di mappa tramite OpenStreetMaps.

## 5.3 La realizzazione della simulazione a Catania

### 5.3.1 La mappa

Come primo passo ho ritenuto saggio gettare le basi per la simulazione sumo, realizzando dunque la mappa. La stesura manuale delle informazioni risulterebbe quasi impossibile e pertanto, consultando la documentazione di sumo, si scopre la disponibilità di un insieme di tool adatti allo scopo. Il primo tool è OpenStreetMap

Web Wizard, si tratta di uno script in python che si trova in /tools/osmWebWizard.py; tale script una volta avviato permette di selezionare una porzione da open streetmap, fornendo fra l'altro anche varie opzioni di inclusione e configurazione. Tali impostazioni, permettono di configurare se importare i poligoni, la direzione del traffico, etc... Inoltre, permette di creare una simulazione completa a partire dalla porzione di mappa selezionata, dunque sarà possibile impostare la durata della stessa, l'importazione del trasporto pubblico e i veicoli (numero e tipo) che saranno ivi presenti.

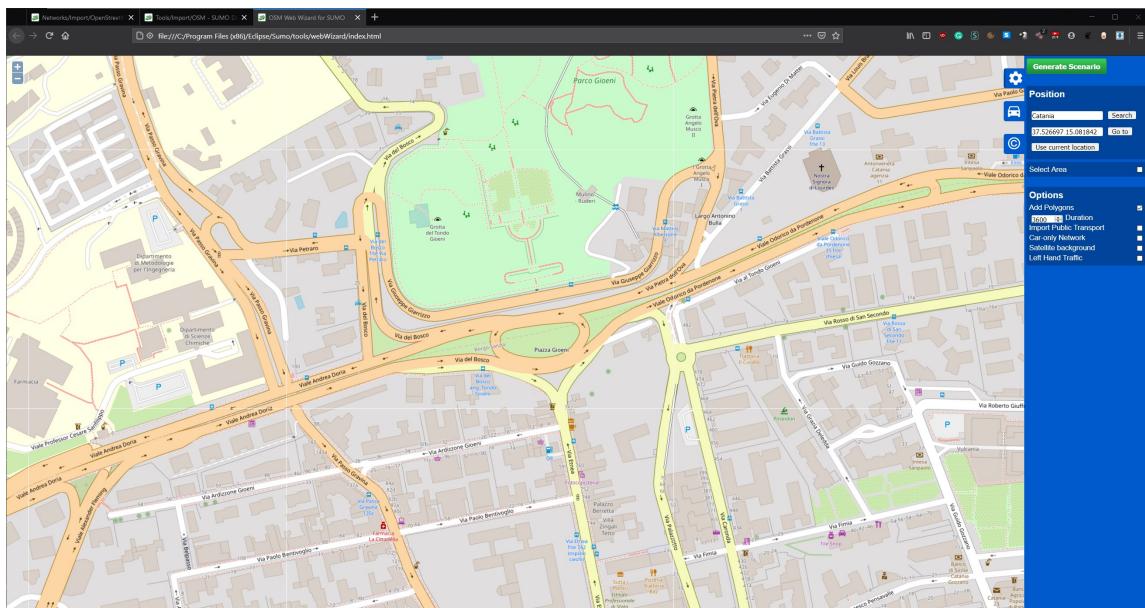


Figura 5.2: OpenStreetMap Web Wizard

Premendo sul tasto “Generate scenario” verrà generato una simulazione completa e avviabile con sumo. I file verranno salvati nella sottocartella tool in base a data e ora. Sono disponibili tutti i file necessari, inclusi quelli di cui abbiamo bisogno, ossia gli xml relativi alla rete stradale ed ai poligoni; catania.poly.xml e catania.net.xml. I file restanti verranno ignorati, in quanto andremo a scrivere il resto della simulazione manualmente. Si noti, che lo script, come già scritto in precedenza è solo uno dei tool disponibili. Infatti, lo stesso fa uso del tool netconvert, questo tool è infatti

in grado di generare, partendo dal file di importazione della mappa i file della rete stradale e degli edifici. Inoltre, sono presenti altri tool in python di conversione nel caso si volessero effettuare altri generi di modifiche automatiche. Per verificare che la mappa venga aperta da sumo correttamente, avviamo la simulazione vuota prima di procedere.



Figura 5.3: Mappa dei dintorni di tondo Gioeni vista dall'IDE di SUMO

Il risultato, visibile in figura 5.3, sembra soddisfacente, sono infatti presenti il tondo Gioeni e dintorni, le strade sono corrette, così come corsie e semafori.

Di seguito una porzione di codice di catania.poly.xml:

```

1  <poly id="164760392" type="building.yes" color="255,230,230" fill="
1" layer="-1.00" shape="164.429333,1389.654611
164.573096,1386.348607 163.486313,1386.292351
163.639000,1382.864316 165.724223,1382.954577
165.948665,1378.051049 182.595095,1378.806411
182.397513,1383.210713 184.394377,1383.300914
184.241714,1386.684572 182.774992,1386.616943
182.604320,1390.466549 164.429333,1389.654611"/>

```

```

2   <poly id="164760393" type="building.yes" color="255,230,230" fill="
1" layer="-1.00" shape="139.692771,1162.769058
140.625296,1156.201894 146.695163,1157.049430
145.940275,1162.340876 155.720964,1163.712524
157.248585,1152.907761 137.528143,1150.142169
136.719924,1155.877350 133.274146,1155.397818
132.679084,1159.624325 135.674262,1160.036967
135.372308,1162.155765 139.692771,1162.769058"/>
3   <poly id="164760394" type="building.yes" color="255,230,230" fill="
1" layer="-1.00" shape="164.779339,1443.340284
164.782031,1439.612594 163.562676,1439.611713
163.564287,1437.381756 165.313796,1437.383019
165.317571,1432.157597 181.602159,1432.169376
181.598639,1437.028686 183.056564,1437.029742
183.054796,1439.470492 181.711738,1439.469519
181.708925,1443.352529 164.779339,1443.340284"/>
```

E una porzione di catania.net.xml:

```

1   <edge id="404295539#0" from="253936717" to="1655286018" name="Viale
Andrea Doria" priority="12" type="highway.primary" spreadType="center"
shape="1108.68,1425.39 1044.76,1404.05 1033.44,1400.31
947.33,1371.39 901.75,1354.20 881.84,1346.03 871.30,1341.06
854.90,1331.28 810.28,1305.63">
2     <lane id="404295539#0_0" index="0" disallow="tram rail_urban
rail rail_electric rail_fast ship" speed="27.78" length="311.10"
shape="1101.30,1424.61 1044.26,1405.56 1032.93,1401.83
946.79,1372.90 901.17,1355.69 881.20,1347.50 870.54,1342.48
854.09,1332.66 813.17,1309.13">
3     <param key="origin" value="404295539"/>
4   </lane>
5     <lane id="404295539#0_1" index="1" disallow="tram rail_urban
rail rail_electric rail_fast ship" speed="27.78" length="311.10"
shape="1102.31,1421.57 1045.27,1402.53 1033.95,1398.79
```

```

947.87,1369.88 902.34,1352.71 882.49,1344.57 872.05,1339.65
855.71,1329.90 814.76,1306.35">
6      <param key="origId" value="404295539"/>
7      </lane>
8  </edge>
```

Si noti come vengono importati correttamente i dati delle vie, includendo nome, dimensione e velocità in base alle caratteristiche. Inoltre, ogni corsia viene identificata, anche in base alla funzione.

### 5.3.2 Configurazione di Sumo

Una volta realizzata la configurazione della mappa è necessario completare il resto della configurazione di sumo; scrivendo dunque il file della configurazione per sumo e definendo il tipo di veicoli e i vari flussi che dovremo simulare. È necessario dunque scrivere i file catania.rou.xml e catania.sumo.cfg.

#### 5.3.2.1 catania.rou.xml

```

1 <?xml version="1.0"?>
2
3 <routes>
4   <vType id="vtype0" accel="2.6" decel="4.5" sigma="0.5" length="2.5"
5     minGap="2.5" maxSpeed="16" color="1,1,0"/>
6   <flow id="flow0" type="vtype0" beg="0" end="0" number="24" from="
7     176458817#0" to="80440362#0"/>
8   <flow id="flow1" type="vtype0" beg="0" end="0" number="24" from="
9     581663152#0" to="80440362#0"/>
10  <flow id="flow2" type="vtype0" beg="0" end="0" number="10" from="
11    581663152#0" to="23650681#0"/>
12  <flow id="flow3" type="vtype0" beg="0" end="0" number="12" from="
13    176458817#0" to="23650681#0"/>
```

```
9 </routes>
```

È stata creata una singola tipologia di veicolo, che servirà come componente dei vari flussi definiti. Sono stati creati quattro flussi, si noti che le parole chiave from e to definiscono quali sono i punti di partenza e fine di quest'ultimi, tramite gli id degli edge presenti nel file catania.net.xml; la parola chiave number definisce il numero di veicoli; la parola chiave type di quali veicoli si tratta. Il primo flusso muove ventiquattro veicoli che vanno dall'inizio del tratto di circonvallazione che attraversa il tondo fino alla fine. Il secondo immette altri ventiquattro veicoli in circonvallazione da una strada secondaria. Il terzo flusso fa muovere dieci veicoli da una strada secondaria, fa attraversare la circonvallazione e li fa uscire dallo svincolo del tondo Gioeni di Via Passo Gravina. Il quarto flusso muove dodici veicoli dall'inizio della circonvallazione allo svincolo del tondo Gioeni di Via Passo Gravina.

### 5.3.2.2 catania.sumo.cfg

```
1   <input>
2       <net-file value="catania.net.xml"/>
3       <route-files value="catania.rou.xml"/>
4       <additional-files value="catania.poly.xml"/>
5   </input>
6
7   <processing>
8       <ignore-route-errors value="true"/>
9   </processing>
10
11  <routing>
12      <device.rerouting.adaptation-steps value="180"/>
13  </routing>
14
15  <report>
```

```

16      <verbose value="true" />
17      <duration-log.statistics value="true" />
18      <no-step-log value="true" />
19  </report>
20
21  <gui_only>
22      <start value="true" />
23  </gui_only>
24
25 </configuration>
```

Nel file di configurazione di cui sopra, vengono specificati i file xml da usare e alcune impostazioni di default relative alla simulazione sumo.

### 5.3.3 Configurazione di OMNET++ e VEINS

Una volta completata la configurazione di sumo, siamo sicuri di essere in grado di visualizzare il movimento dei veicoli desiderati in una porzione di Catania. È necessario dunque andare ad inserire la VANET in questo traffico che è stato creato, per farlo dobbiamo ovviamente sfruttare le restanti capacità di VEINS; basandoci sulle conoscenze che sono state acquisite dallo studio di quest'ultimo.

#### 5.3.3.1 omnetpp.ini

```

1 [General]
2 cmdenv-express-mode = true
3 cmdenv-autoflush = true
4 cmdenv-status-frequency = 1s
5 **.cmdenv-log-level = info
6
7 ned-path = .
```

```
8 image-path = ../../images
9
10 network = RSUExampleCatania
11
12 #####
13 #           Simulation parameters          #
14 #####
15 debug-on-errors = true
16 print-undisposed = true
17
18 sim-time-limit = 15000s
19
20 **.scalar-recording = true
21 **.vector-recording = true
22
23 *.playgroundSizeX = 3000m
24 *.playgroundSizeY = 3000m
25 *.playgroundSizeZ = 50m
26
27 #####
28 # Annotation parameters          #
29 #####
30 *.annotations.draw = true
31
32 #####
33 # Obstacle parameters          #
34 #####
```

```
35 *.obstacles.obstacles = xmldoc("config.xml", "//AnalogueModel[@type='
SimpleObstacleShadowing']/obstacles")

36
37 #####
38 #           TraCIScenarioManager parameters      #
39 #####
40 *.manager.updateInterval = 1s
41 *.manager.host = "localhost"
42 *.manager.port = 9999
43 *.manager.autoShutdown = true
44 *.manager.launchConfig = xmldoc("catania.launchd.xml")

45
46 #####
47 #           RSU SETTINGS                  #
48 #                                     #
49 #                                     #
50 #####
51 *.rsu[0].mobility.x = 2000
52 *.rsu[0].mobility.y = 2000
53 *.rsu[0].mobility.z = 3
54
55 *.rsu[*].applType = "TraCIDemoRSU11p"
56 *.rsu[*].appl.headerLength = 80 bit
57 *.rsu[*].appl.sendBeacons = true
58 *.rsu[*].appl.dataOnSch = false
59 *.rsu[*].appl.beaconInterval = 1s
60 *.rsu[*].appl.beaconUserPriority = 7
61 *.rsu[*].appl.dataUserPriority = 5
```

```
62 *.rsu[*].nic.phy80211p.antennaOffsetZ = 0 m
63
64 #####1#####
65 #           11p specific parameters          #
66 #                                     #
67 #           NIC-Settings                  #
68 #####1#####
69 *.connectionManager.sendDirect = true
70 *.connectionManager.maxInterfDist = 2600m
71 *.connectionManager.drawMaxIntfDist = false
72
73 *.*.nic.mac1609_4.useServiceChannel = false
74
75 *.*.nic.mac1609_4.txPower = 20mW
76 *.*.nic.mac1609_4.bitrate = 6Mbps
77 *.*.nic.phy80211p.minPowerLevel = -110dBm
78
79 *.*.nic.phy80211p.useNoiseFloor = true
80 *.*.nic.phy80211p.noiseFloor = -98dBm
81
82 *.*.nic.phy80211p.decider = xmldoc("config.xml")
83 *.*.nic.phy80211p.analogueModels = xmldoc("config.xml")
84 *.*.nic.phy80211p.usePropagationDelay = true
85
86 *.*.nic.phy80211p.antenna = xmldoc("antenna.xml", "/root/Antenna[@id='"
87     "monopole'])"
88 *.node[*].nic.phy80211p.antennaOffsetY = 0 m
89 *.node[*].nic.phy80211p.antennaOffsetZ = 1000 m
```

```

89
90 ######
91 #           App Layer      #
92 #####
93 *.node[*].applType = "TraCICT11p"
94 *.node[*].appl.headerLength = 80 bit
95 *.node[*].appl.sendBeacons = true
96 *.node[*].appl.dataOnSch = false
97 *.node[*].appl.beaconInterval = 1s
98
99 #####
100 #          Mobility       #
101 #####
102 *.node[*].veinsmobility.x = 0
103 *.node[*].veinsmobility.y = 0
104 *.node[*].veinsmobility.z = 0
105 *.node[*].veinsmobility.setHostSpeed = false
106 *.node[*0].veinsmobility.accidentCount = 3
107 *.node[*0].veinsmobility.accidentStart = 73s
108 *.node[*0].veinsmobility.accidentDuration = 50s

```

Innanzitutto, viene definito il nome del network, chiamato RSUExampleCatania.

Poi, nella sezione dedicata ai parametri di simulazione viene specificato un tempo di simulazione di 15000 secondi, un tempo che permette alla simulazione di completare la movimentazione di tutti i veicoli. Viene anche specificata la dimensione della mappa che dovrà essere di 3000x3000 metri. Nella sezione dedicata a TraCI indichiamo dove cercare il server, ossia in localhost nella porta 9999. Inoltre, viene anche passato il file di configurazione catania.launchd.xml che illustrerò dopo. Nella sezione dedicata alla RSU, specificiamo la posizione di quest'ultima e le istruiamo

di rilanciare i beacon ogni secondo; inoltre specifichiamo la lunghezza dell'header, che come recita il protocollo 1609.4 è di 80bit. Nella sezione dedicata alla NIC vengono invocati i moduli del mac e livello fisco dei protocolli 1609.4 e 802.11p, i parametri sono abbastanza standardizzati così come gli xml da allegare. Nella sezione legata al livello applicativo viene sincronizzato l'invio dei beacon con quello della RSU, inoltre si dà riferimento al tipo di applicazione, che andrà a richiamare il codice ned e C++ (relativo al comportamento dei nodi) che scriveremo in seguito. Infine, viene specificato l'avvenire di 3 incidenti, dopo 73 secondi di run dalla durata di 56 secondi.

### 5.3.3.2 catania.launchd.xml

```

1 <launch>
2   <copy file="catania.net.xml" />
3   <copy file="catania.rou.xml" />
4   <copy file="catania.poly.xml" />
5   <copy file="catania.sumo.cfg" type="config" />
6 </launch>
```

In questo file di configurazione vengono definiti quali file devono essere inviati al server TraCI. Si tratta di tutti i file xml relativi a sumo, sarà infatti compito di TraCI recapitarli al simulatore.

### 5.3.3.3 TraCICT11p.ned

Spostandoci nella cartella del layer applicativo descritta all'inizio di questo capitolo possiamo scrivere il file ned e il codice c++ relativo ai nodi.

```

1 package org.car2x.veins.modules.application.traci;
2 import org.car2x.veins.modules.application.ieee80211p.DemoBaseApplLayer
3 ;
```

```

4 simple TraCICT11p extends DemoBaseApplLayer
5 {
6     @class( veins :: TraCICT11p );
7     @display( " i=block/app2" );
8 }
```

Nel nostro file ned, ci limitiamo ad importare i moduli di veins relativi a TraCI e per lo standard di comunicazione VANET. Infine, specifichiamo il nome della rete e un'impostazione grafica di default.

#### 5.3.3.4 TraCICT11p.h

```

1 namespace veins {
2
3 class VEINS_API TraCICT11p : public DemoBaseApplLayer {
4 public:
5     void initialize(int stage) override;
6
7 protected:
8     simtime_t lastDroveAt;
9     bool sentMessage;
10    int currentSubscribedServiceId;
11    bool mal;
12    bool mal2;
13
14 protected:
15     void onWSM(BaseFrame1609_4* wsm) override;
16     void onWSA(DemoServiceAdvertisment* wsa) override;
17
18     void handleSelfMsg(cMessage* msg) override;
19     void handlePositionUpdate(cObject* obj) override;
20 };
21
```

```
22 } // namespace veins
```

Nell'header della classe specifichiamo il tipo di classe "VEINS\_API", e la ereditiamo da DemoBaseAppLayer, che come si intuisce dal nome fornisce la classe madre per la realizzazione di demo che sfruttano il livello applicativo di veins. Definiamo dunque un metodo initialize(...) che verrà chiamato una volta istanziato il nodo di riferimento. Le variabili *simtime\_t lastDroveAt; bool sentMessage;* e *int currentSubscribedServiceId;* servono al funzionamento di base del nodo, servono a tenere traccia dell'invio dell'ultimo beacon e se sono iscritti alla RSU. Oltre a queste variabili, ho inserito altri due booleani di cui spiegherò la necessità successivamente. Tutti gli altri metodi sono i trigger previsti da veins una volta ricevuto un messaggio e relativi alla loro gestione. SI noti che ad esempio viene preso come argomento una frame 1609.4.

### 5.3.3.5 TraCICT11p.cc

```
1 #include "veins/modules/application/traci/TraCICT11p.h"
2 #include "veins/modules/application/traci/TraCI11pMessage_m.h"
3 #include <random>
4
5 using namespace std;
6 using namespace veins;
7
8 Define_Module(veins::TraCICT11p);
9
10 void TraCICT11p::initialize(int stage)
11 {
12     DemoBaseApplLayer::initialize(stage);
13     if (stage == 0) {
14         random_device rd;
15         mt19937 gen(rd());
```

```
16     uniform_real_distribution<float> dis(0.0, 1.0);
17     sentMessage = false;
18     lastDroveAt = simTime();
19     currentSubscribedServiceId = -1;
20     mal2 = false;
21     if(dis(gen) > 0.6) {
22         mal = true;
23         findHost()->getDisplayString().setTagArg("i", 1, "red");
24     }
25     else mal = false;
26 }
27 }
28
29 void TraCICT11p::onWSA(DemoServiceAdvertisment* wsa)
30 {
31     if (currentSubscribedServiceId == -1) {
32         mac->changeServiceChannel(static_cast<Channel>(wsa->
33             getTargetChannel()));
34         currentSubscribedServiceId = wsa->getPsid();
35         if (currentOfferedServiceId != wsa->getPsid()) {
36             stopService();
37             startService(static_cast<Channel>(wsa->getTargetChannel()),
38                         wsa->getPsid(), "Mirrored Traffic Service");
39         }
40     }
41     void TraCICT11p::onWSM(BaseFrame1609_4* frame)
42 {
43
44     TraCI11pMessage* wsm = check_and_cast<TraCI11pMessage*>(frame);
45 }
```

```

46     if (!mal) {
47         findHost()->getDisplayString().setTagArg("i", 1, "green");
48
49         if (mobility->getRoadId()[0] != ':') traciVehicle->changeRoute(
50             wsm->getDemoData(), 9999);
51         if (!sentMessage) {
52             sentMessage = true;
53             wsm->setSenderAddress(myId);
54             wsm->setSerial(3);
55             scheduleAt(simTime() + 2 + uniform(0.01, 0.2), wsm->dup());
56         }
57     }
58
59 void TraCICT11p::handleSelfMsg(cMessage* msg)
60 {
61     if (TraCI11pMessage* wsm = dynamic_cast<TraCI11pMessage*>(msg)) {
62         // this code only runs when channel switching is enabled
63         sendDown(wsm->dup());
64         wsm->setSerial(wsm->getSerial() + 1);
65         if (wsm->getSerial() >= 3) {
66             // stop service advertisements
67             stopService();
68             delete (wsm);
69         }
70     }
71     else {
72         scheduleAt(simTime() + 1, wsm);
73     }
74 }
75 else {
76     DemoBaseApplLayer::handleSelfMsg(msg);
77 }
```

```
77 }
78
79 void TraCICT11p::handlePositionUpdate(cObject* obj)
80 {
81     DemoBaseApplLayer::handlePositionUpdate(obj);
82
83     // stopped for for at least 10s?
84     if (mal2) {
85         if (mobility->getSpeed() > 1) {
86             if (simTime() - lastDroveAt >= 10 && sentMessage == false)
87             {
88                 findHost()->getDisplayString().setTagArg("i", 1, "yellow");
89                 sentMessage = true;
90
91                 TraCI11pMessage* wsm = new TraCI11pMessage();
92                 populateWSM(wsm);
93                 wsm->setDemoData(mobility->getRoadId().c_str());
94
95                 // host is standing still due to crash
96                 if (dataOnSch) {
97                     startService(Channel::sch2, 42, "Traffic Info
98 Service");
99                     scheduleAt(computeAsynchronousSendingTime(1,
100 ChannelType::service), wsm);
101
102                 }
103             }
104         else {
105             sendDown(wsm);
106         }
107     }
108 }
```

```
105         lastDroveAt = simTime() ;
106     }
107 }
108 else {
109     if (mobility->getSpeed() < 1) {
110         if (simTime() - lastDroveAt >= 10 && sentMessage == false)
111     {
112         findHost ()->getDisplayString () .setTagArg ("i" , 1 , "
113         yellow");
114         sentMessage = true;
115
116         TraCI11pMessage* wsm = new TraCI11pMessage();
117         populateWSM(wsm);
118         wsm->setDemoData(mobility->getRoadId() .c_str());
119
120         // host is standing still due to crash
121         if (dataOnSch) {
122             startService(Channel::sch2 , 42 , "Traffic
123             Information Service");
124             scheduleAt(computeAsynchronousSendingTime(1 ,
125             ChannelType::service) , wsm);
126         }
127     }
128     else {
129         lastDroveAt = simTime();
130     }
131 }
132 }
```

Dato che il codice è univoco per tutti i nodi, per simulare l'ingresso di nodi infettati ho sfruttato una distribuzione randomica che mi permette di simulare l'ingresso di nodi malevoli in mappa, la probabilità che un nodo sia malevolo è del 40%. Dunque, la necessità dei booleani `textit{mal}` e `textit{tmal}` è proprio quella di identificare i nodi malevoli durante la simulazione.

Per prima cosa viene definita il metodo `initialize(stage);`, quest'ultima viene chiamata ogni volta la simulazione effettua un'iterazione, fortunatamente viene passata la variabile `stage` che ne indica il numero. In questa funzione faccio uso delle funzioni di C++ per generare una distribuzione uniforme di numeri reali compresi fra 0.0 e 1.0, istanziando `seed` e generatore. Pertanto, dopo aver effettuato le operazioni per inizializzare le variabili necessarie al funzionamento del nodo, pongo che se il numero estratto dalla distribuzione è maggiore di 0.6 il nodo verrà indicato come malevolo e colorato di rosso nella simulazione altrimenti il booleano verrà messo a falso e il nodo aderirà normalmente al protocollo.

Il metodo `onWSA(DemoServiceAdvertisment* wsa)` viene richiamato solo quando è attivo il channel switching, ossia se il canale di comunicazione usato dai veicoli e RSU può cambiare. Questo metodo gestisce tale eventualità permettendo al nodo di comunicare; infatti quando avviene un cambio di canale viene inviata un avviso di servizio con tutte le informazioni necessarie.

Il metodo `onWSM(BaseFrame1609_4* frame)` è invece di maggiore interesse per il nostro studio, questo metodo gestisce infatti le frame wave ricevute dal nodo tramite il protocollo 1609.4. Anzitutto, viene convertita la frame e successivamente, a seconda del booleano che indica se il nodo sia malevolo o meno, viene specificato un comportamento. Qualora quest'ultimo sia malevolo non verrà mandato alcun messaggio all'infrastruttura ne verranno prese azioni per evitare un eventuale rischio di ingorgo; altrimenti, il nodo aggiornerà l'infrastruttura sul traffico e prenderà eventuali provvedimenti, cambiando strada.

Il metodo **handleSelfMsg(cMessage\* msg)** serve di nuovo nel caso in cui sia abilitato il channel switching. Il nodo invierà infatti frame di servizio qualora il canale sia cambiato.

Il metodo **handlePositionUpdate(cObject\* obj)** viene usato per la segnalazione di incidenti. È stato predisposto per usare un altro flag malevolo. Infatti, per inviare una segnalazione di incidente un veicolo deve avere una velocità minore di uno per almeno dieci secondi, è qui infatti che entra in gioco la variabile *lastdroveAt*; nel nostro caso se il booleano *mal2* è attivo inviamo la segnalazione se invece il veicolo è in movimento, colorandolo di giallo.

## 5.4 Simulazioni

Una volta scritto il codice, è sufficiente andare nella IDE di OMNET++ per avviare la simulazione e dopo una moderata attesa possiamo analizzare le simulazioni per capirne i risultati.

### 5.4.1 Simulazione di Catania senza VANET

Prima di tutto è stata avviata una simulazione in cui i veicoli si muovevano senza alcun supporto da parte della VANET.

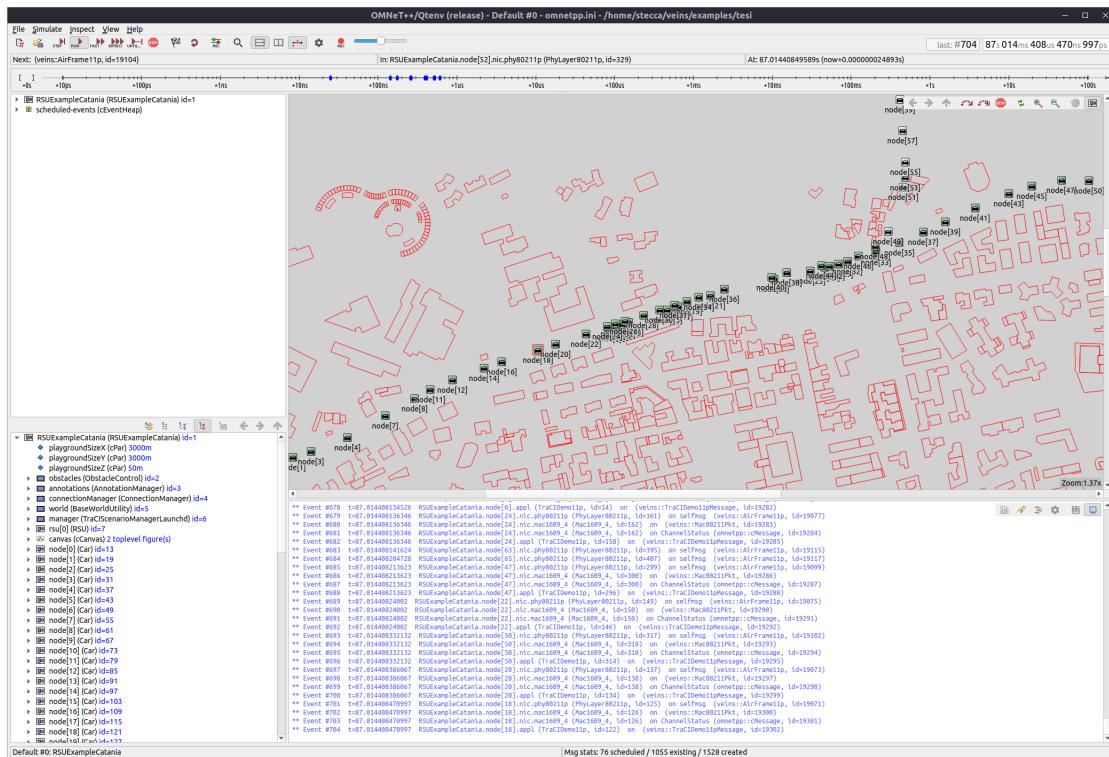


Figura 5.4: Simulazione del regolare traffico nei dintorni di tondo Gioeni

Si può vedere in figura 5.4 come il traffico forma le regolari code, e prende tutto lo stesso percorso. Il risultato è previsto, il comportamento dei veicoli è analogo a quanto avviene nella realtà. Questo dato è importante, perché ci conferma che la simulazione produce risultati realistici.

### 5.4.2 Simulazione di Catania con VANET funzionante

Successivamente sono state avviate varie esecuzioni in cui la VANET V2I funziona regolarmente e tutti i nodi rispettano correttamente il protocollo.

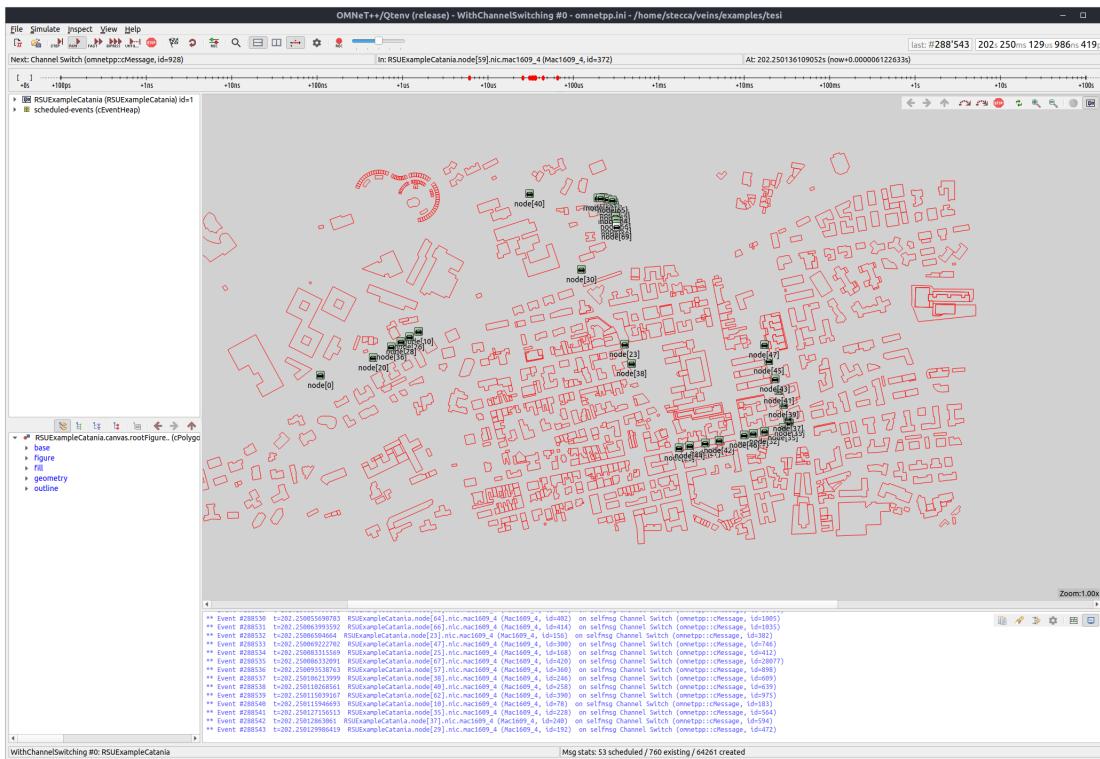


Figura 5.5: Simulazione di una VANET V2I a Catania vista dalla IDE di OMNET++

Questa volta in figura 5.5 possiamo appurare come il traffico è andato molto più veloce, al punto da rendere difficoltosa la realizzazione degli screenshot; i veicoli grazie alle funzionalità della VANET sono riusciti ad evitare la formazione di ingorghi, prendendo percorsi vari e diversi a seconda della situazione. Il risultato è un traffico scorrevole e distribuito lungo varie vie. Di seguito, nelle figure 5.6, 5.7 e 5.8, alcuni particolari presi da SUMO.

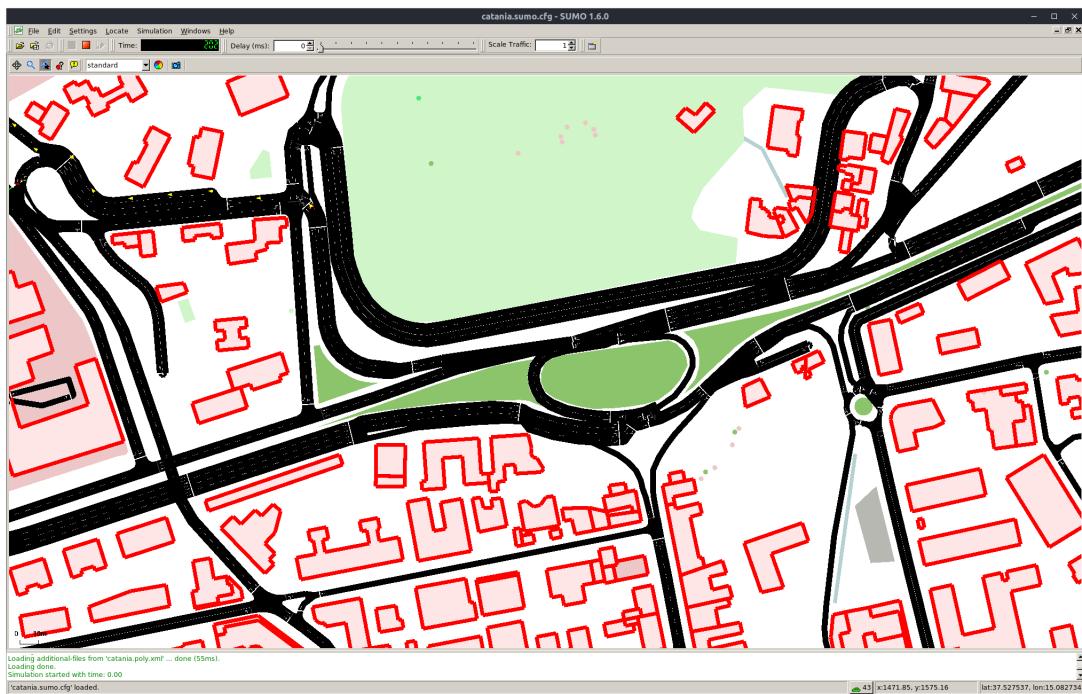


Figura 5.6: Tondo Gioeni privo di traffico

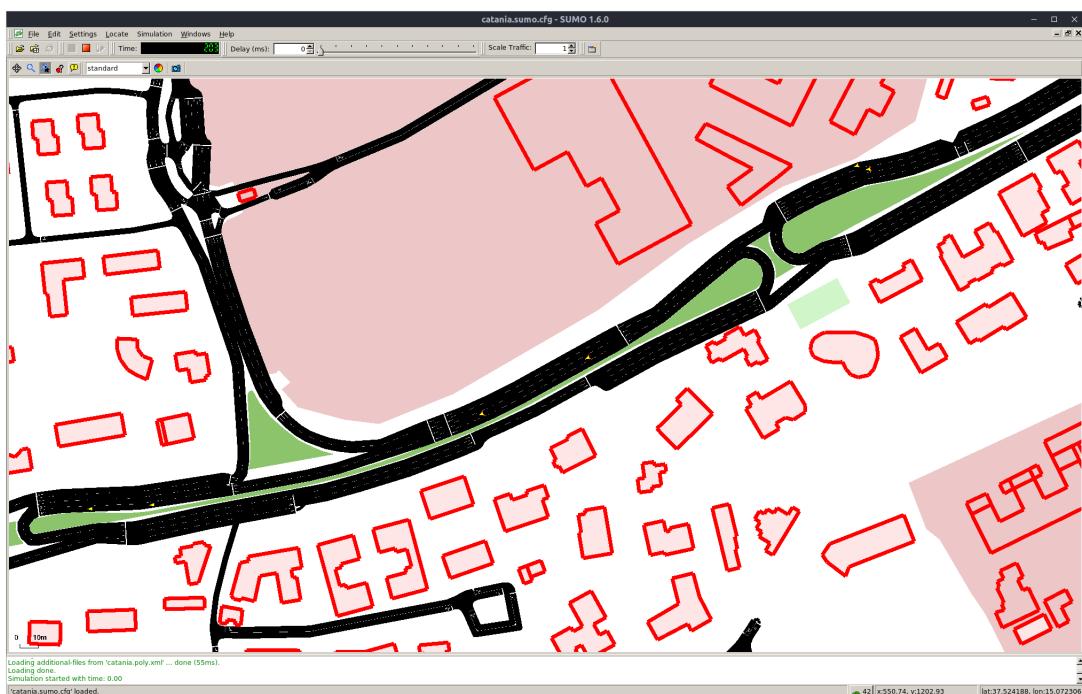


Figura 5.7: Traffico regolare in circonvallazione



Figura 5.8: Alcuni veicoli che intraprendono una strada alternativa passando da Via Ingegnere

Anche questo risultato è confortante, la VANET si comporta come previsto, adempiendo adeguatamente ai compiti svolti. A questo punto possiamo procedere.

#### 5.4.3 Simulazione di Catania con VANET compromessa

Una volta che ci siamo assicurati il corretto funzionamento della simulazione, proviamo ad attaccare la VANET tramite lo scenario descritto in precedenza. Con un numero esiguo di nodi infetti, l'impatto è stato inesistente, però una volta portata la percentuale dei nodi infetti al 40% la situazione è cambiata.

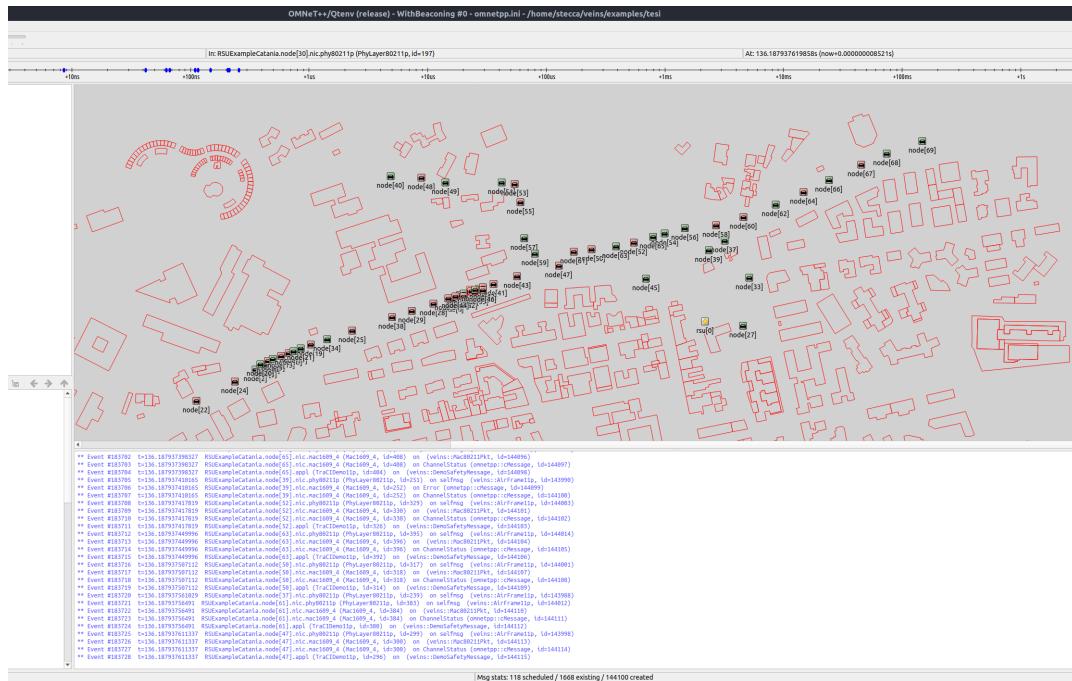


Figura 5.9: Presenza di nodi malevoli evidenziata nell'IDE di OMNET++

Guardando la figura 5.9 effetti dell'attacco si notano immediatamente, sono presenti una quantità rilevante di nodi infetti (in rosso), che una volta ammassati in un solo punto creano di nuovo un ingorgo a discapito dei nodi normali (in verde). Solo in pochi riescono a prendere una strada alternativa, grazie al loro posizionamento.

Il problema si evince anche tramite sumo, che mostra varie code.

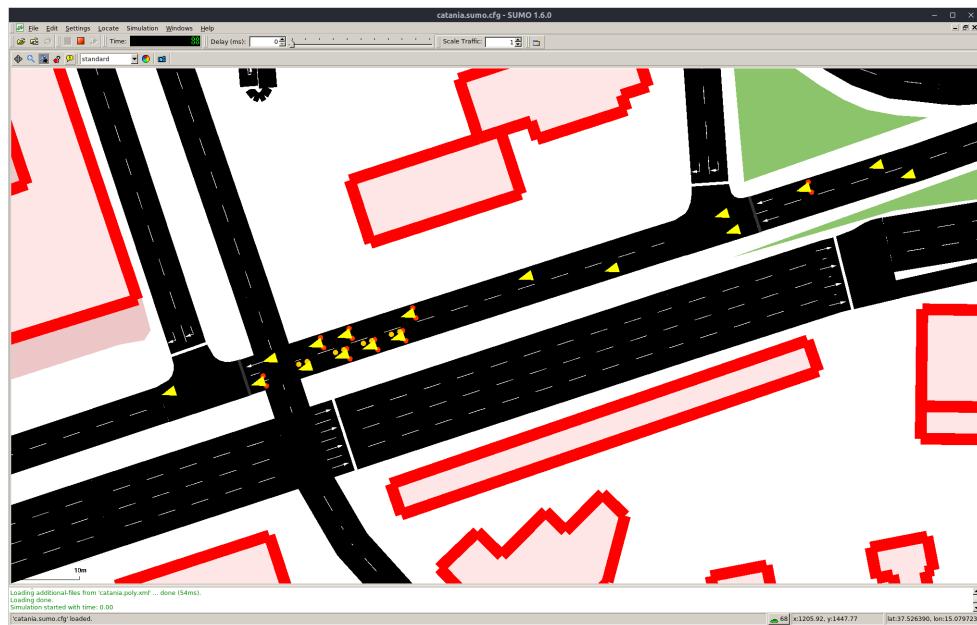


Figura 5.10: Particolare di ingorgo visto dalla GUI SUMO



Figura 5.11: Altro ingorgo visto dalla GUI SUMO

Si tratta di un buon risultato, che conferma le aspettative sull'attacco e riesce a dimostrare che una buona quantità di nodi malevoli può vanificare o rendere controproducente l'esistenza stessa della VANET.

---

# 6

## Conclusioni

### 6.1 Risultati ottenuti

Grazie a questi studi sono state poste con successo le fondamenta per la realizzazione di un threat modeling relativo alle VANET V2I. La simulazione realizzata e l'attacco effettuato su di essa dimostrano come le VANET non siano ancora esenti da problemi; difatti, nonostante il protocollo 1609.4 sia sicuro possono essere attuate altre mosse fantasiose da parte di un attaccante in grado di minarne effettivamente l'efficacia. Nella nostra simulazione, ponendo un numero sufficiente di nodi infetti, siamo riusciti a compromettere il sistema del consenso seminando il caos e vanificando lo scopo stesso della VANET. Inoltre, alcuni nodi segnalavano anche incidenti fasulli, questa parte di attacco ha avuto però un'efficacia limitata, in quanto non è stato possibile coordinare questa mossa insieme agli altri nodi malevoli. Ad ogni modo il rischio resta, è la simulazione, su cui è stato effettuato il più basilare degli attacchi lo dimostra, gli effetti sulla VANET sono chiari e non possono essere ignorati.

## 6.2 Limitazioni dei simulatori

Sfortunatamente gli altri scenari non possono essere simulati con VEINS, in quanto i limiti sono troppi. La documentazione è pressappoco inesistente, gran parte del codice risulta immodificabile e difficile da personalizzare. Se l'architettura del simulatore resta così, sarà difficile avere libertà sufficiente da poter testare efficacemente altre forme di attacco. Infatti, VEINS nasce per simulare VANET perfettamente funzionanti, svolge il compito in maniera egregia e permette di realizzare molti scenari, d'altro canto creare dei malfunzionamenti è praticamente impossibile. Questo dimostra poca lungimiranza nell'approccio alla realizzazione delle VANET. Anzi-tutto, è necessario poter testare in generale la resilienza di una rete a qualunque forma di problema e soprattutto, appare evidente che non vi è abbastanza sensibilizzazione relativa alla necessità di comprendere i rischi di sicurezza, studiarli e prevenirli. Allo stato attuale è necessario che tale approccio venga rivisto, va prodotta una documentazione seria ed effettuato un refactoring del framework che permetta di condurre test di sicurezza completi, permettendo anche di modificare le frame. Inoltre, sono presenti altri protocolli e modelli di messaggi oltre ai WAVE, anche questi andrebbero inclusi per poter condurre prove su questi ultimi. Ad aggravare la situazione, VEINS resta comunque l'opzione migliore, in quanto produce simulazioni funzionanti ed è l'unico framework realmente accessibile liberamente e ancora supportato.

## 6.3 Sviluppi futuri

Appare evidente che prima di poter diffondere su larga scala tecnologie applicative basate sulle VANET vada effettuato un passaggio, a mio avviso, essenziale ossia molteplici studi e simulazioni volti alla sicurezza delle stesse VANET. Il lavoro futuro

includerà sicuramente una modellazione completa dei rischi, studi su altri tipi di messaggi oltre ai WAVE e la realizzazione di piattaforme di simulazione o altri mezzi che mettano in condizione di effettuare in sicurezza delle prove soddisfacenti.

---

## Bibliografia

- [1] Saleh Yousefi, Mahmoud Siadat Mousavi, M.F.: Vehicular Ad Hoc Networks (VANETs): Challenges and Perspectives. (2006)
- [2] Khan, Zia, Z.: Vehicular ad-Hoc networks (VANETs)—An overview and challenges. (2006)
- [3] 802.11 WG - Wireless LAN Working Group: IEEE 802.11p-2010 - IEEE Standard for Information technology— Local and metropolitan area networks— Specific requirements— Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 6: Wireless Access in Vehicular Environments. [https://standards.ieee.org/standard/802\\_11p-2010.html](https://standards.ieee.org/standard/802_11p-2010.html) (2010)
- [4] 1609 WG - Dedicated Short Range Communication Working Group: 1609.4-2016 - IEEE Standard for Wireless Access in Vehicular Environments (WAVE) – Multi-Channel Operation. [https://standards.ieee.org/standard/1609\\_4-2016.html](https://standards.ieee.org/standard/1609_4-2016.html) (2016)
- [5] : VEINS Documentation. (<https://veins.car2x.org/documentation/>)
- [6] : OMNET++ Introduction. (<https://omnetpp.org/intro/>)
- [7] : SUMO Documentation. (<https://sumo.dlr.de/docs/index.html>)
- [8] : TraCI Documentation. (<https://sumo.dlr.de/docs/TraCI.html>)
- [9] : OMNET++ Download. (<https://omnetpp.org/download/>)

[10] : Veins GitHub Official Repo. (<https://github.com/sommer/veins>)

---

# Ringraziamenti

Ringrazio il mio relatore, **Prof. Giampaolo Bella**, che mi ha permesso di avviare e concludere questa tesi. Grazie alla sua professionalità, abilità e conoscenze mi ha fatto appassionare al mondo della sicurezza informatica; ha fornito tante opportunità ed occasioni sempre nuove per misurarsi con se stessi, dispensando una gran quantità di buoni consigli sulle scelte da intraprendere. Il suo aiuto sempre presente e a volte anche inaspettato è stato essenziale ed ha indirizzato il sottoscritto verso una strada promettente e piena di opportunità.

Ringrazio il mio correlatore ed amico prezioso, il **Dottor Pietro Biondi**, che ha fornito un supporto, a mia modesta opinione, irraggiungibile da nessun altro correlatore; ha risolto problemi di tutti i tipi, ha sempre elargito buoni consigli ed è stato sempre attento ad ogni piccolo dettaglio. Le sue azioni sono state risolutive alla realizzazione di questa tesi, che senza di lui oggi non sarebbe completa. Il suo supporto non si è fermato qui, è stato un eccellente compagno di corso e grazie alla sue estreme competenza ha sempre fornito ottimi aiuti e consigli durante gli studi. Inoltre, è stato un grande amico, sempre pronto a fornire una mano d'aiuto, sempre presente e compagno di mille avventure.

*Grazie Pietro, sei l'amico ed il correlatore che tutti vorrebbero.*

Ringrazio il **Prof. Salvatore Antonio Riccobene**, per avermi fatto appassionare al mondo del networking, influenzando quindi in maniera positiva le mie scelte accademiche, e per il suo inimitabile carisma che ha reso l'ambiente universitario più umano e vicino allo studente.

Vorrei ringraziare i colleghi e amici:

- **Angelo Zinna, Antonio Caragliano, Salvatore Leotta e Valentino Merlino** con i quali abbiamo affrontato insieme le varie competizioni di cybersecurity a cui siamo stati chiamati ad affrontare.
- **Davide De Pasquale, Salvatore Graci, Giulia Barchitta, Rosario Leonardi, Vincenzo Aliperti e Ludovico Trupia** con i quali abbiamo condiviso momenti felici e superato problemi grandi e piccoli legati allo studio.
- **Tutti i colleghi** che posso di cui posso aver dimenticato di scrivere e che mi perdoneranno per la dimenticanza.

I miei ex-colleghi di **Movia S.p.A.**, che hanno contribuito alla mia crescita professionale.

Ringrazio gli amici **Dario Porto, Angela Ficarelli, Alfio Giuliano Faro, Manuela Leonardi, Nino Pulvirenti, Maria Grazia Trovato e Chiara D'Agata**, che nonostante le distanze e le difficoltà hanno trovato il tempo per un saluto, anche telematico.

Ringrazio inoltre gli amici **Michele Napoli, Kristina Nicole Martin, Enrico Romano e Alessia Giuffrida**, conosciuti grazie a Pietro Biondi, con i quali ho trascorso momenti ed avventure indimenticabili.

**Federica Vella**, sempre pronta a dare una parola di conforto; in questi anni si è dimostrata una grande amica che mi ha affiancato durante momenti davvero belli e irripetibili.

**Giuseppe La Mela**, amico a cui è stato affidato il mio precedente incarico in Movia S.p.A. e che ha saputo affrontare in maniera eccellente.

L'amica **Victoria Abbattista**, che è sempre stata presente e mi ha capito aiutato anche quando nessun altro lo ha fatto; compagna di mille momenti allegri e indelebili.

L'amica **Anna Lo Trovato**, sempre stata pronta a proporre qualcosa di folle e a risollevarmi l'umore; è stata anche una collega alla laurea triennale, sono innumerevoli i problemi universitari e le avventure affrontate insieme ed mi ha sempre sorpreso con le sue creative dimostrazioni di amicizia.

L'amico **Livio Arcidiacono**, con il quale è nata in circostanze del tutto inusuali un'amicizia salda; è sempre stato pronto a sostenermi nei momenti di difficoltà ed è sempre stato un ottima spalla e compagno di grandi avventure.

Un grandissimo ringraziamento va alla mia ragazza, **Martina Romeo** il cui arrivo inaspettato nella mia vita ha reso ogni giorno pieno di sorprese. Sempre pronta a seguirmi in qualche avventura e a starmi accanto, mettendomi sempre al primo posto; mi sorprende ogni giorno con piccoli gesti inaspettati.

*Grazie per i tuoi piccoli gesti d'amore e per le grandi esperienze vissute insieme, spero di viverne altre insieme a te.*

Ringrazio mia zia, **Maria Antonietta Cutuli**, che nei momenti più inaspettati è stata pronta a darmi sostegno nei momenti di sconforto riuscendo a riservarmi delle parole buone ed aiuto.

Ringrazio mia cugina **Claudia Privitera**, che è stata per me come una prima sorella e con cui ho passato momenti, partendo dalla mia infanzia, che non dimenticherò mai.

Ringrazio il mio zio e padrino **Antonio Massimo Cutuli**, che sin da bambino mi ha sempre strappato un sorriso e dato la spinta giusta per andare avanti.

Voglio ricordare mio zio **Pancrazio Privitera**, che ci ha lasciato in maniera improvvisa e ingiusta; lo voglio ringraziare per essermi sempre stato d'esempio con il suo modo di affrontare la vita, sempre con il sorriso e pronto a risolvere ogni problema. Ha sempre creduto in me con fermezza e fierezza, raccontando a tutti di me con gioia.

*Mi ha fatto imparare tante cose pratiche e non. Spero da qualche parte continui*

*a vivere anche in me, dandomi un pò della sua forza soprattutto nei momenti di difficoltà maggiore; in cui lui riusciva sempre a distinguersi.*

Ricordo, ovviamente anche *mia nonna Gaetana Scandura*, che sin da piccolo mi ha sempre sostenuto ed amato, credendo sempre in me e guardando con fierezza ogni mia singola conquista. Ricordo il suo sguardo fiero e la sua commozione quando ottenni la mia laurea di primo livello, e lo sento anche oggi, vivo fra noi, mentre sto per raggiungere questo ennesimo traguardo.

*Grazie Nonna, i tuoi consigli ed il tuo sorriso riecheggieranno sempre nell'eternità, nella mia mente e nel mio cuore.*

Un ricordo va anche a *mio nonno Sebastiano Cutuli*, che era sempre pronto ad elargirmi qualche aneddoto o storia delle sue, lasciandomi sempre qualche insegnamento.

**GRAZIE a mia madre, Giuseppa Simona Cutuli.** Senza i suoi sacrifici e la sua dedizione nel crescermi non sarei mai riuscito a fare tutta questa strada nè avrei raggiunto importanti risultati. Sempre pronta a proteggermi e a dare tutta se stessa per me.

*Mamma, ti devo tantissimo, grazie, i tuoi sforzi non verranno mai dimenticati.*

**GRAZIE a mio padre, Alfredo Steccanella.** È stato lui ad alimentare il mio amore nei confronti dell'informatica, dandomi lui stesso i primi preziosi insegnamenti. È stato lui che ha fatto sì, ad ogni costo, che io potessi studiare e imparare, aiutandomi a coronare dei sogni avuti sin da bambino ed ad intraprendere una valida carriera.

*Papà, grazie per il tuo lavoro, i tuoi consigli e i mezzi che hai sempre cercato di fornirmi; sono qui grazie a te.*

**Grazie a mia sorella, Valeria Steccanella.** Che ho cresciuto un pò anche io, mi è sempre stata vicina e con cui ho vissuto momenti ed emozioni indelebili; una sorella fantastica, a cui spero di aver insegnato qualcosa di utile e di averle dato

buoni consigli, come ha fatto lei con me.

*Grazie sorellina, con te non mi sono mai sentito solo.*

*Infine grazie a voi, che avete avuto l'interesse, il tempo e la pazienza di  
leggere questa tesi.*

Luca Steccanella