

UNIVERSITÀ  
degli STUDI  
di CATANIA

DIPARTIMENTO DI INGEGNERIA  
ELETTRICA ELETTRONICA E INFORMATICA

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

RELAZIONE FINALE DEL PROGETTO PER L'INSEGNAMENTO "COMPUTER SECURITY"

STUDENTI:

*Luigi Fontana N. 1000037171*

## Sommario

1. Introduzione.....	3
1.1 Threat Model .....	3
1.2 Architetture Intra-Veicolari .....	5
1.3 Architetture Inter-Veicolari .....	6
1.4 Obiettivi del progetto .....	8
2. Organizzazione di VEINS .....	8
2.1 Caricamento della configurazione .....	9
2.2 Scenario di attacco.....	10
3.Simulazioni .....	10
3.1 Scenario 0.....	11
3.2 Scenario 1.....	12
3.3 Scenario 3.....	15
4.Conclusioni.....	18

## 1. Introduzione

Negli ultimi anni l'automobile si è evoluta da un sistema puramente fisico ad un sistema cyber-fisico, che incorpora al suo interno componenti elettroniche con capacità di calcolo atte a migliorare le prestazioni e la safety del guidatore.

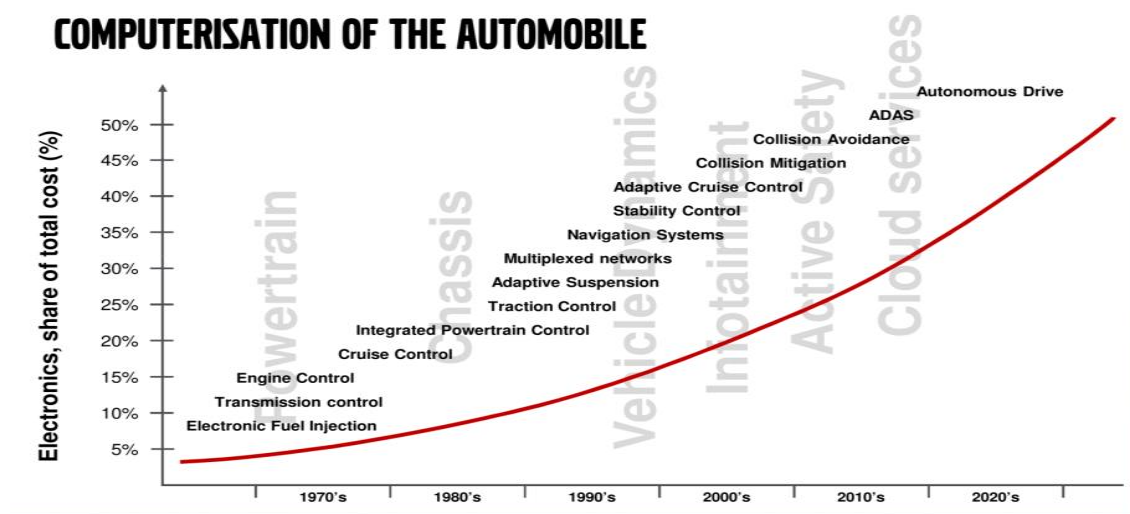


Figura 1 – Elettronica in Automotive

L'utilizzo di queste tecnologie ha portato l'interscambio di un gran numero di informazioni che possono essere di utilizzo benigno per incentivare il risparmio energetico e l'ottimizzazione del traffico, o maligno nel caso in cui un malintenzionato voglia usarle per il proprio beneficio.

Le ragioni che potrebbero spingere un attaccante a compromettere un'auto sono molteplici: furto dati, estorsione, frode, inganno, spionaggio industriale, accesso illegale a proprietà intellettuali.

Per meglio difendersi dunque, bisogna individuare una superficie di attacco.

### 1.1 Threat Model

Si può creare un Threat Model partendo dalle informazioni sull'architettura, per formulare dei diagrammi che permettano di identificare e mitigare le minacce. Nel Threat Model di livello 0, Fig.2, si mostra come le informazioni possano attraversare il veicolo.

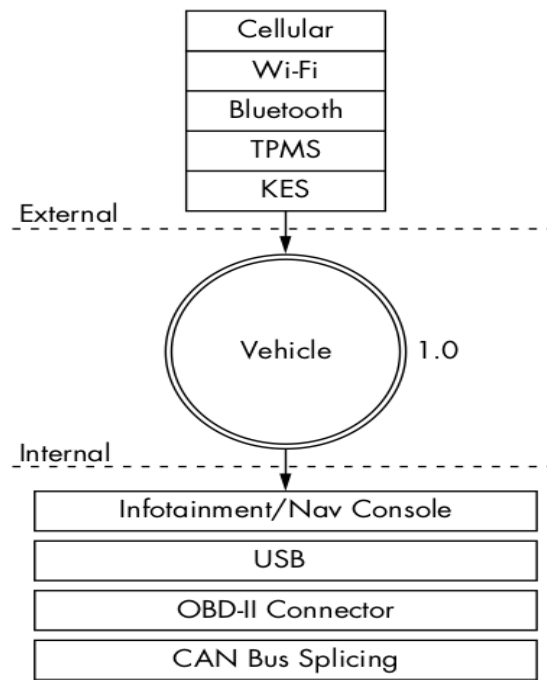


Figura 2 – Threat Model di livello 0.

Si hanno due tipologie di ingressi. Input provenienti da un dispositivo esterno e input provenienti dall'architettura interna del veicolo. Esplorando il veicolo si sviluppa il Threat Model di livello 1 con relativo diagramma, Fig. 3, in cui si evidenziano i collegamenti che gli input hanno con le diverse parti del veicolo.

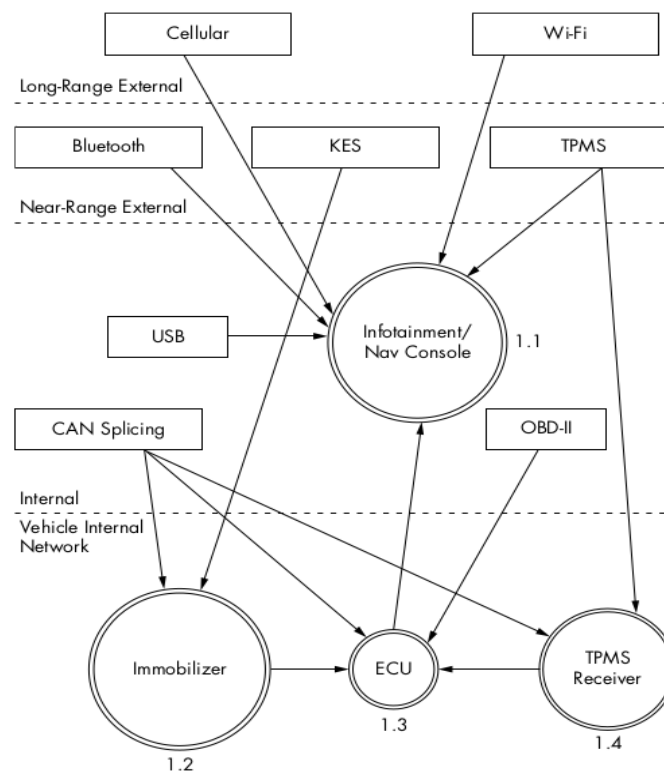


Figura 3 – Threat Model di livello 1.

Le linee tratteggiate rappresentano le trust boundaries; più trust boundaries vengono attraversate più rischioso diviene quel canale di comunicazione. È necessario risolvere i threat in base a quale parte all'architettura viene presa di mira. In automotive si distinguono in: Intra-Veicolari ed Inter-Veicolari.

## 1.2 Architetture Intra-Veicolari

Le Architetture intra-veicolari sono formate da diverse centinaia di centraline che coadiuvate da protocolli di comunicazione formano i domini funzionali, i quali garantiscono dei servizi al guidatore.

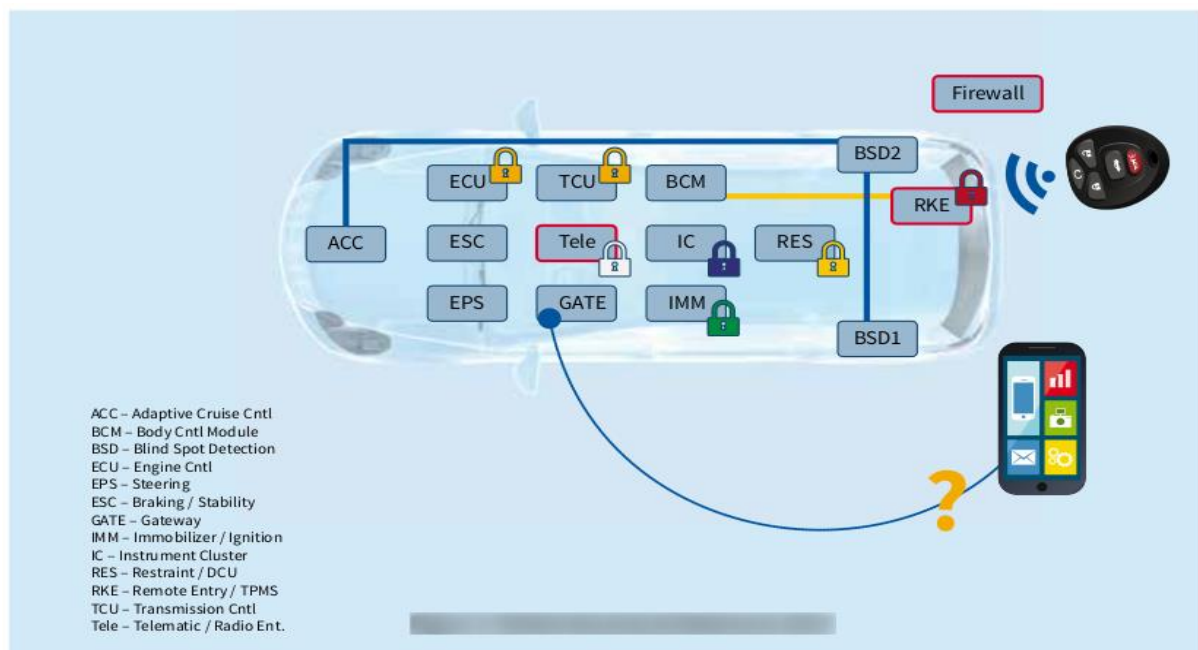


Figura 4 – Architettura di sicurezza Intra-Veicolare.

Uno dei modi per mettere in sicurezza l'interscambio di informazioni è quello di limitare la connettività a poche funzioni e proteggere i sistemi integrati con moduli che ne impediscano la manomissione, Fig 4.

È possibile implementare un approccio multilivello dove i sistemi critici sono protetti da gateway che si occupano di proteggere l'intera gerarchia dei sistemi elettronici.

Nel gateway viene utilizzato un High Assurance Hardware Trust Anchor che si occupa di eseguire un controllo chiamato trust transitive. Quando il sistema viene avviato, il Trust Anchor passa attraverso una serie di controlli per assicurarsi che non siano state effettuate modifiche alle componenti elettroniche, Fig 5.

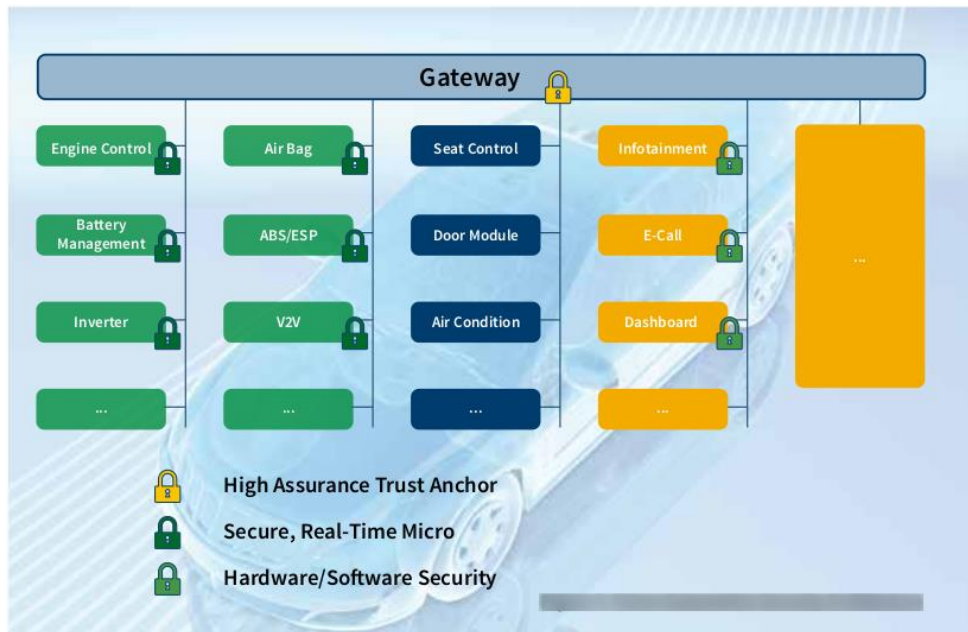


Figura 5 – Architettura di sicurezza Intra-Veicolare.

### 1.3 Architetture Inter-Veicolari

Le Architetture inter-veicolari, conosciute come VANET, Vehicular Ad Hoc Network permettono le comunicazioni con altri veicoli (V2V) con l'infrastruttura stradale (V2I) e con l'utente (U2V). Le VANET sono state standardizzate in vari protocolli come lo standard IEEE 801.11p e la sua evoluzione IEEE 1609.4

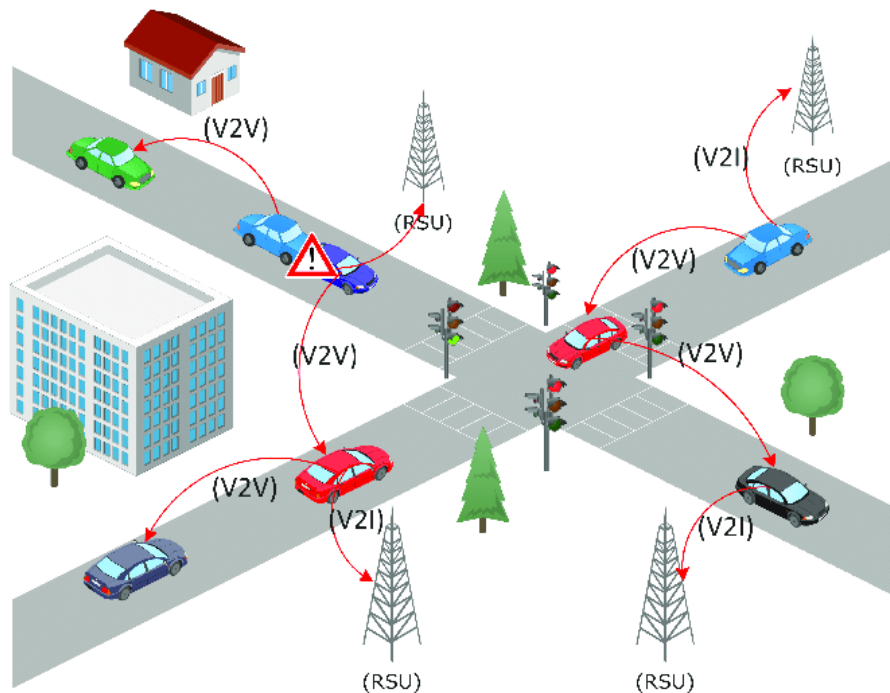


Figura 6 – Esempio di rete Vanet.

Lo standard IEEE 1609.4 definisce come rendere sicuri i messaggi WAVE (Wireless Access in Vehicular Environments) utilizzando la crittografia e i certificati. Lo standard si preme 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.

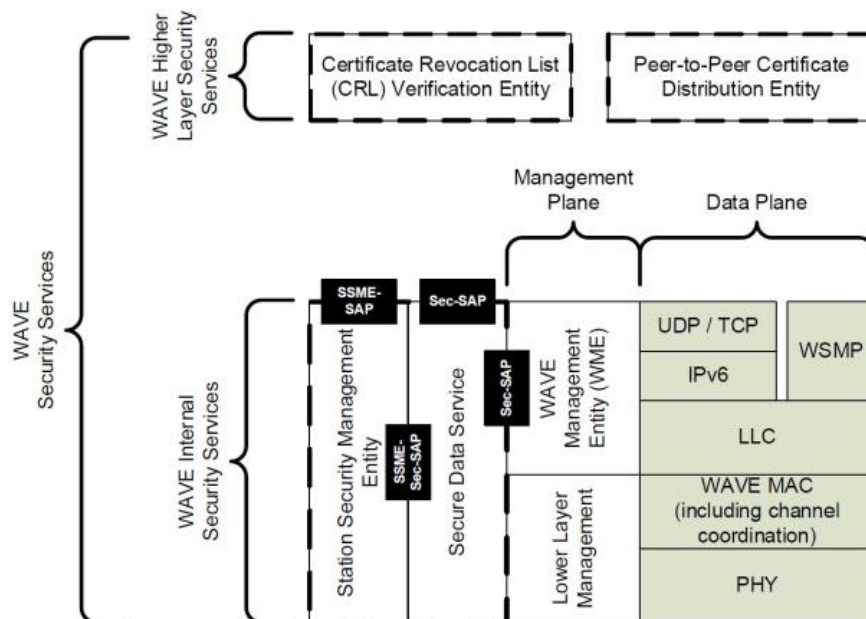


Figura 7 – Wave Security Services.

In Figura 6, si mostra la struttura del layer di comunicazione basato su protocolli come 801.11p, IP, TCP/UDP. La presenza dei WAVE security services forniscono le primitive di sicurezza descritte in precedenza.

I messaggi Wave contengono delle informazioni standard, come la posizione, la velocità, l'attivazione di dispositivi di sicurezza, freni ed altre ancora; ciò permette agli elementi di rete la valutazione di cosa accade nell'ambiente circostante. Per esemplificare, quando un nodo riceve la segnalazione che più veicoli sono fermi in un punto, questo verrà comunicato all'utente per fargli risparmiare tempo, oppure alla ricezione di un avviso di attivazione multipla dell'ABS l'utente sarà cosciente di un imminente pericolo. Per evitare di dare troppo potere ad un singolo nodo, sono necessarie più segnalazioni del problema; quindi, si evita che un attaccante, modificando la sua ECU, abbia spazio di azione. In questo modo si ha un sistema che si avvicina al consenso, infatti, sono necessari più nodi per realizzare che ci sia un problema.

## 1.4 Obiettivi del progetto

Il progetto punta alla simulazione di due scenari applicativi riguardanti le VANET ed in particolare la tipologia V2I tramite il framework VEINS, un simulatore di reti automotive. È stato preso in esame il paese di mia provenienza dove non esiste un'infrastruttura di rete capace di gestire il traffico veicolare e sono stati studiati i benefici di una sua implementazione. Successivamente sono stati studiati due possibili scenari in cui un attaccante compromettendo l'infrastruttura porti ad un malfunzionamento del protocollo WAVE.

## 2. Organizzazione di VEINS

Per prima cosa è stato studiato il framework VEINS e le sue caratteristiche.

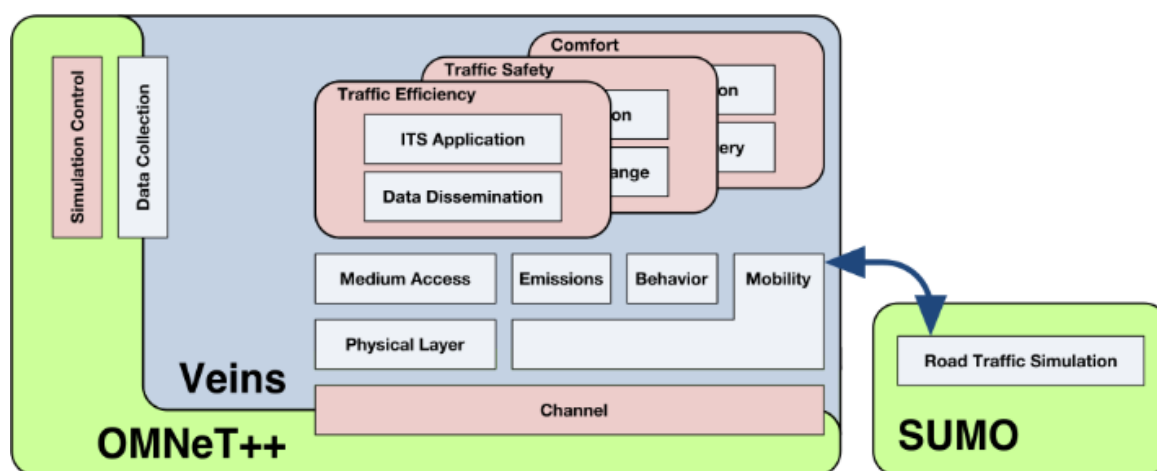


Figura 8 – Componenti del VEINS Framework.

Nello schema in figura 7 sono presenti le componenti che permettono di simulare le VANET.

Alla base è presente OMNET++, un simulatore che agevola la rappresentazione di reti di comunicazione. Al suo interno troviamo moduli necessari per simulare le VANET e TraCI che permette di far comunicare OMNET++ con SUMO, un simulatore di traffico veicolare.

VEINS presenta delle elevate limitazioni, tra cui la documentazione, che è pragmaticamente inesistente salvo il minimo indispensabile. Inoltre, il framework non è progettato per simulare liberamente scenari che vanno contro i canoni del protocollo poiché i moduli sono scritti per comportarsi in un determinato modo e ciò non permette che un nodo funzioni in maniera non consona a ciò che le VANET per loro natura offrono. Pertanto, replicare scenari di attacco in cui la rete è malfunzionante risulta estremamente complesso; Per questo motivo è stato adottato un approccio indiretto.



## 2.1 Caricamento della configurazione

Dopo aver studiato il framework, averlo installato ed averne compreso il funzionamento, è stato scelto lo scenario di applicazione per gli esperimenti effettuati. Il comune di Canicatti (AG) presenta una morfologia ottimale per l'obiettivo dello studio, poiché è affetto da numerose code soprattutto durante l'orario di punta.

Per caricare una mappa su VEINS è necessario sfruttare il tool OpenStreetMap Web Wizard Fig. 8, ovvero uno script python chiamato `osmWebWizard.py` presente all'interno della cartella del software SUMO dentro la cartella `/tools`.

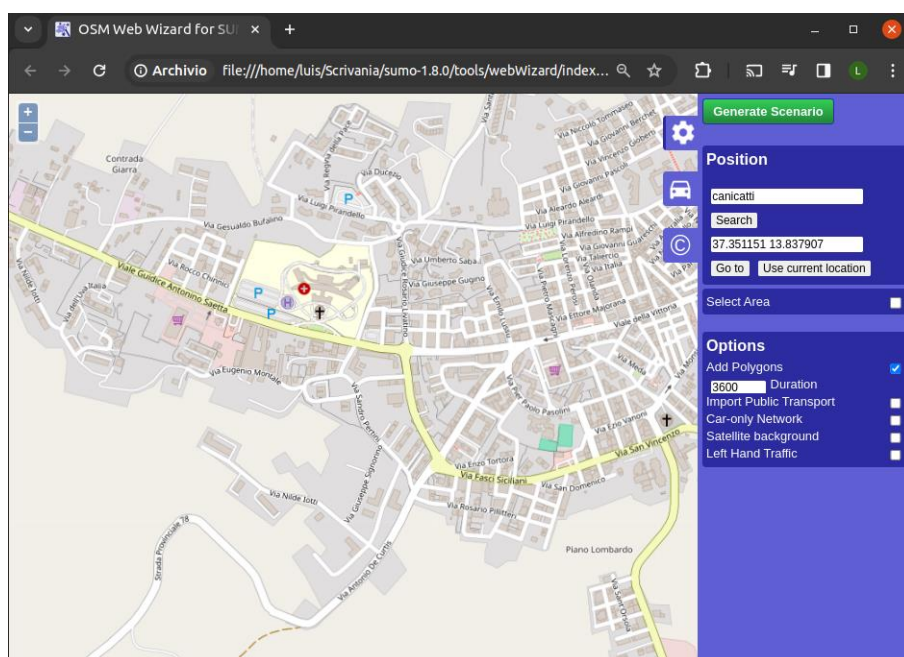


Figura 9 – OpenStreetMap Web Wizard.

Questo tool produrrà diversi file salvati in una sottocartella in base alla data e all'ora. I file di interesse sono `*.net.xml` che contiene i dettagli delle strade che formano la rete stradale e `*.poly.xml` che definisce gli edifici e i poligoni presenti nella mappa.

Ulteriori file necessari per la configurazione sono stati scritti a mano e sono:

- `*.rou.xml`, necessario per definire i veicoli e i percorsi dei nodi.
- `*.sumo.cfg`, specifiche dei file xml da usare ed alcune impostazioni di default relative alla simulazione sumo.
- `*.launch.xml`, configurazione dei file inviati al server TraCI.
- `omnetpp.ini`, definizione di tutto lo scenario ovvero l'insieme dei file dei quali fare riferimento.

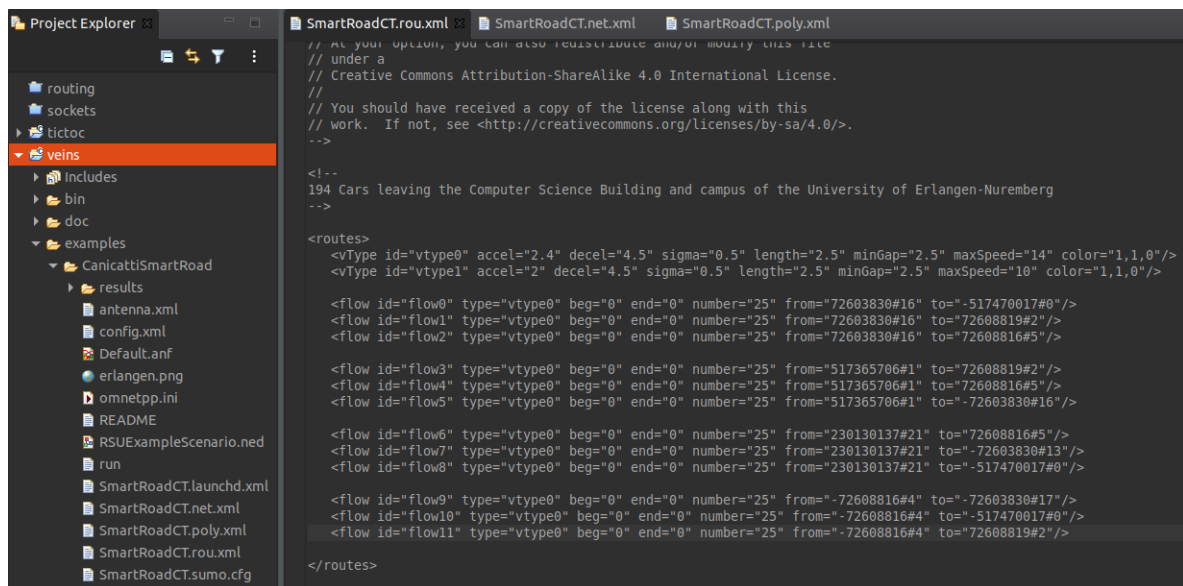


Figura 10 – Vista di progetto.

## 2.2 Scenario di attacco

Lo scenario preso in considerazione tiene conto del protocollo e di come viene utilizzato. Di per sé il protocollo non è vulnerabile ad attacchi di intercettazione, manomissione, alterazione e replay. Pertanto, è necessario minare l'infrastruttura nelle sue funzionalità. Il funzionamento dei messaggi WAVE prevede che la segnalazione al guidatore avvenga traendo informazioni da più nodi e ciò determinerà la presenza di un problema.

È stato supposto che i nodi vengano attaccati in maniera diretta, manomettendo le ECU o attaccando in maniera remota una falla del sistema.

Lo scopo dell'attaccante è quello di colpire più veicoli dello stesso tipo o con la medesima vulnerabilità manomettendo manualmente le ECU o lasciando che uno o più nodi infetti, tramite le comunicazioni di servizio, diffondano l'infezione.

Dati i protocolli, la VANET e gli strumenti di simulazione è stato preso in considerazione uno scenario che vede diversi guidatori ignari che partecipano all'infrastruttura di rete senza trarne vantaggio, vanificando lo scopo dell'infrastruttura e creando dei problemi a sé stessi e a tutti i guidatori non infetti.

## 3. Simulazioni

Le simulazioni effettuate sono tre. La prima vede la VANET funzionante per come i protocolli prevedono. La seconda vede l'inserimento di nodi malevoli che non rispettano il protocollo, mentre nella terza i nodi malevoli fingono l'avvenuta di un incidente ed inviano una segnalazione all'infrastruttura. Per tutte le simulazioni sono state estrapolate due metriche con lo scopo di

evidenziare ulteriormente l'impatto che dei nodi malevoli hanno sulle prestazioni della rete. Inoltre, grazie alle funzionalità di OMNET++ e ad uno script in Python è stato possibile plottarne i risultati e calcolare le medie dei valori presenti in dei file in formato csv.

### 3.1 Scenario 0

La prima simulazione implementa la VANET e mostra come le automobili seguono il protocollo correttamente, Fig 9.

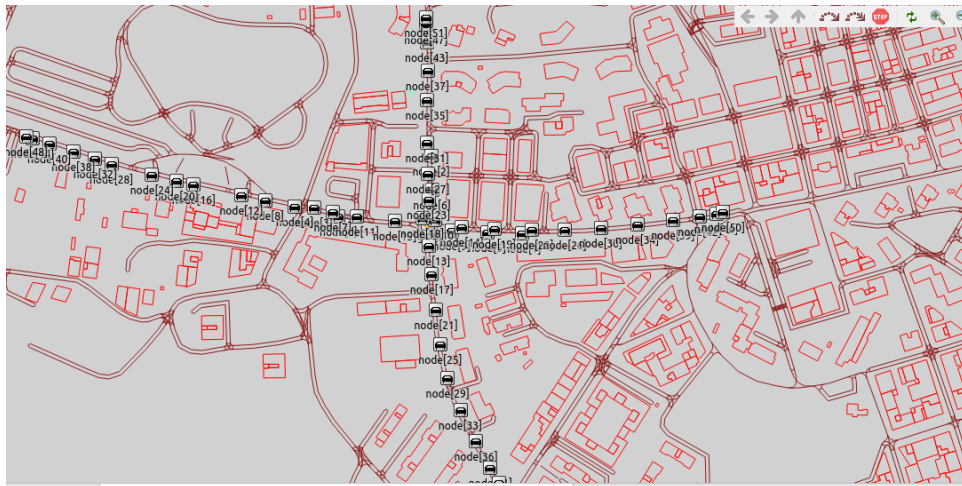


Figura 11 – Implementazione della VANET.

I veicoli grazie alle funzionalità della VANET sono riusciti ad evitare la formazione di ingorghi, prendendo percorsi vari a seconda della situazione. Ciò risulta in un traffico scorrevole lungo le varie vie, che agevola i guidatori.

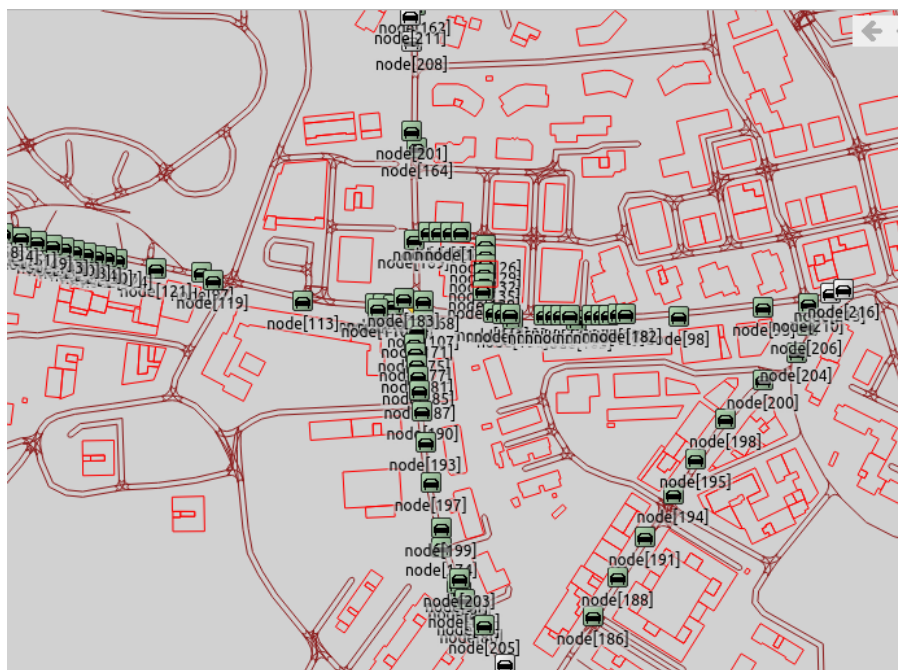


Figura 12 – Cambio di percorso.

Le metriche prese in considerazione sono presenti in Fig 11 e 12. Queste immagini evidenziano le emissioni e i rallentamenti effettuate dai nodi.

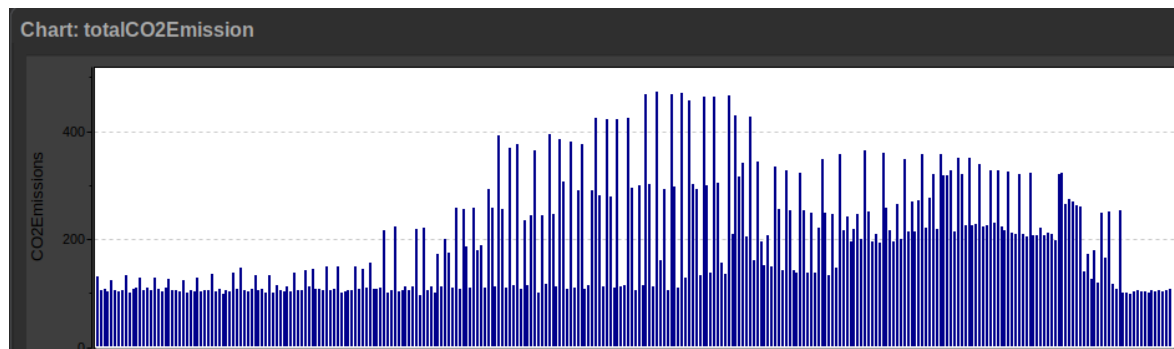


Figura 131 – Emissioni CO2 Scenario 0.

Nell'immagine sopra, è presente il valore delle emissioni di CO2, asse delle ordinate, di ogni singolo nodo, asse delle ascisse. Si nota come i valori non superano i 400 punti e la media si aggira intorno ai 197 punti.

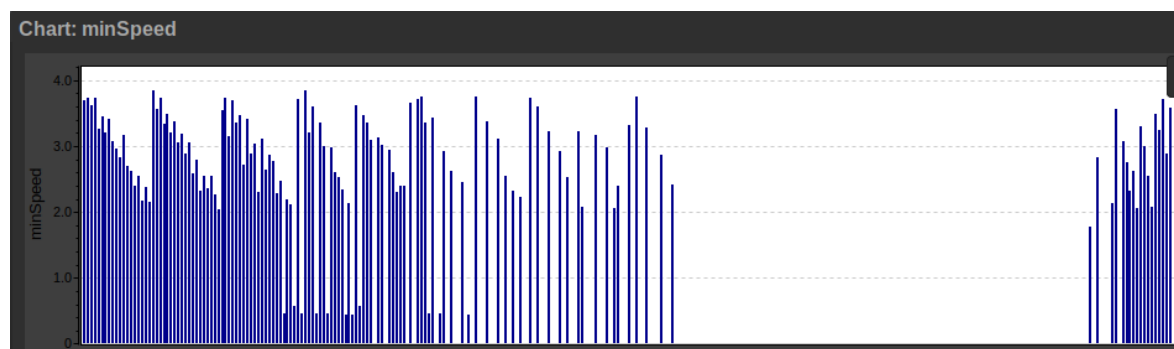


Figura 142 – Velocità Minima Scenario 0.

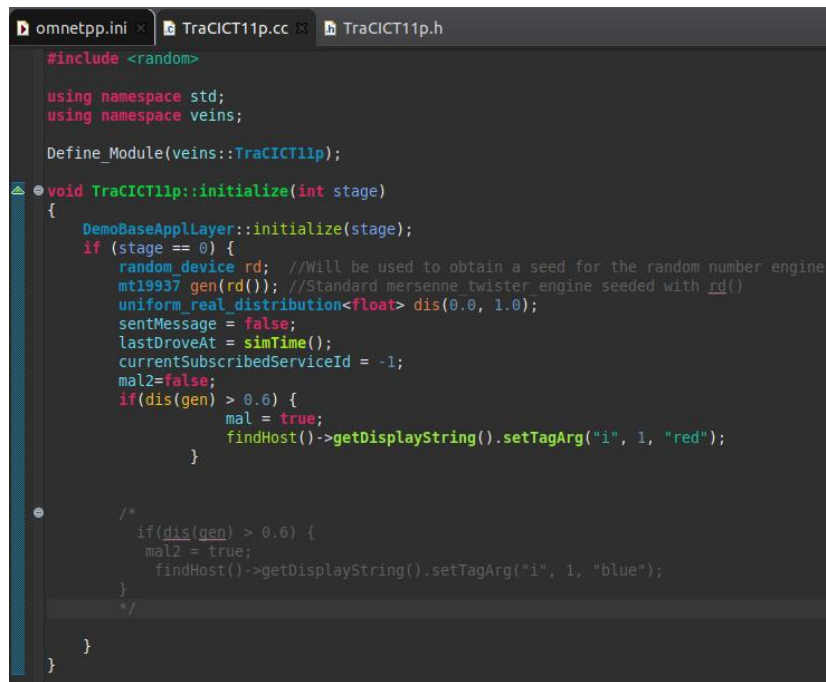
Mentre nella figura 12 sono presenti i nodi e la relativa velocità minore raggiunta. La media in questo caso è di 1.27 punti.

In questo modo si hanno dei valori di partenza, che aggiunti all'analisi visiva, ci mostrano le differenze tra i diversi scenari.

È importante notare che non è stato possibile individuare le corrispondenti unità di misura del sistema internazionale per la cospicua mancanza di documentazione.

### 3.2 Scenario 1

Per la seconda simulazione sono state fatte delle modifiche al codice C++ appartenente al layer applicativo dei nodi che indicherà la reazione degli stessi alla ricezione dei messaggi WAVE. Tale codice si trova nella cartella di veins dentro /src/modules/application/traci e definisce solo una porzione di comportamento univoca per tutti i nodi.



```

#include <random>

using namespace std;
using namespace veins;

Define_Module(veins::TraCICT11p);

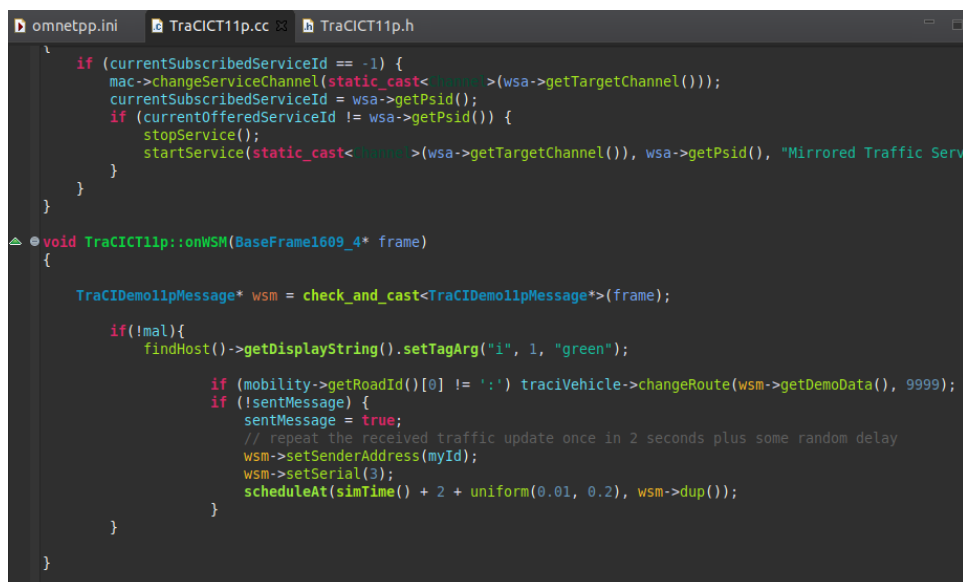
void TraCICT11p::initialize(int stage)
{
    DemoBaseApplLayer::initialize(stage);
    if (stage == 0) {
        random_device rd; //Will be used to obtain a seed for the random number engine
        mt19937 gen(rd()); //Standard mersenne twister engine seeded with rd()
        uniform_real_distribution<float> dis(0.0, 1.0);
        sentMessage = false;
        lastDroveAt = simTime();
        currentSubscribedServiceId = -1;
        mal2=false;
        if(dis(gen) > 0.6) {
            mal = true;
            findHost()->getDisplayString().setTagArg("i", 1, "red");
        }

        /*
        if(dis(gen) > 0.6) {
            mal2 = true;
            findHost()->getDisplayString().setTagArg("i", 1, "blue");
        }
        */
    }
}

```

Figura 15 – Funzione Initialize(stage).

Per simulare l'ingresso nella mappa di nodi infetti, Fig 13, è stata modificata la funzione **inititalize(stage)**, chiamata ogni qual volta la simulazione effettua un'iterazione. Utilizzando una distribuzione randomica con una probabilità del 40% viene posto il nodo come malevolo da un flag e dal colore rosso.



```

if (currentSubscribedServiceId == -1) {
    mac->changeServiceChannel(static_cast<Channel*>(wsa->getTargetChannel()));
    currentSubscribedServiceId = wsa->getPsid();
    if (currentOfferedServiceId != wsa->getPsid()) {
        stopService();
        startService(static_cast<Channel*>(wsa->getTargetChannel()), wsa->getPsid(), "Mirrored Traffic Serv");
    }
}

void TraCICT11p::onWSM(BaseFrame1609_4* frame)
{
    TraCIDemolpMessage* wsm = check_and_cast<TraCIDemolpMessage*>(frame);

    if(!mal){
        findHost()->getDisplayString().setTagArg("i", 1, "green");

        if (mobility->getRoadId()[0] != ':') traciVehicle->changeRoute(wsm->getDemoData(), 9999);
        if (!sentMessage) {
            sentMessage = true;
            // repeat the received traffic update once in 2 seconds plus some random delay
            wsm->setSenderAddress(myId);
            wsm->setSerial(3);
            scheduleAt(simTime() + 2 + uniform(0.01, 0.2), wsm->dup());
        }
    }
}

```

Figura 16 – Funzione onWSM(BaseFrame1609\_4\* frame).

Nella figura 14 viene mostrata la modificata effettuata sulla funzione **onWSM(BaseFrame1609\_4\* frame)**, responsabile di gestire le frame WAVE ricevute dal nodo tramite protocollo 1609.4. Qualora un nodo 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.

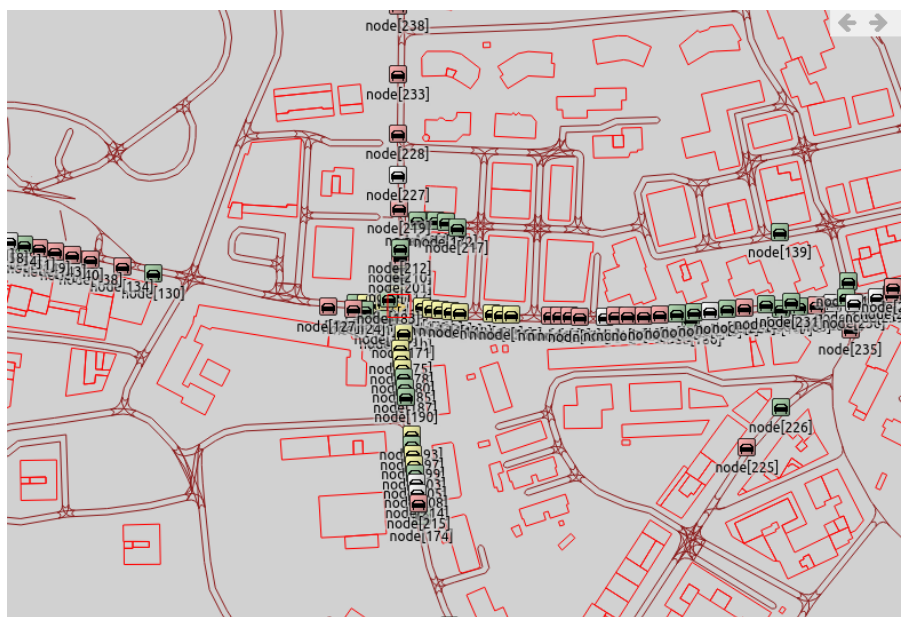


Figura 17 – Ingorgo generato dai nodi malevoli in rosso.

Come mostra la figura 15, in questo caso, il numero di nodi che cambiano il proprio percorso è nettamente inferiore, inoltre il numero crescente di nodi di colore giallo dimostra come le automobili sono ferme in attesa a causa dell'ingorgo; infatti, questo è determinato da una funzione che verrà descritta in seguito, che colora i nodi di giallo quando questi sono fermi per più di 10 secondi.

Le metriche mostrano un notevole aumento delle emissioni di CO2 con picchi che sfiorano i 650 punti ed una media di circa 212 punti. Questo è dovuto all'aumento delle code e di conseguenza allo speco di energia.

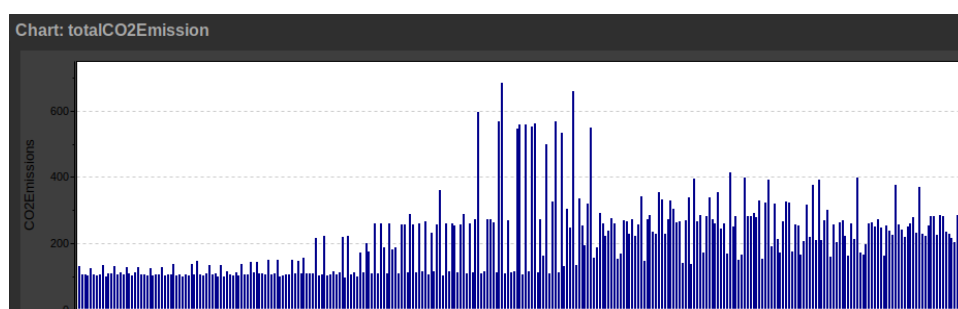


Figura 18 – Emissioni Co2 Scenario 1.

La figura 16 mostra le velocità minime raggiunte dai nodi. Questo per dimostrare come un numero maggiore di nodi abbia raggiunto la velocità 0 pertanto si sia fermato ad aspettare in coda. La media in questo caso e di circa 1.11 punti.

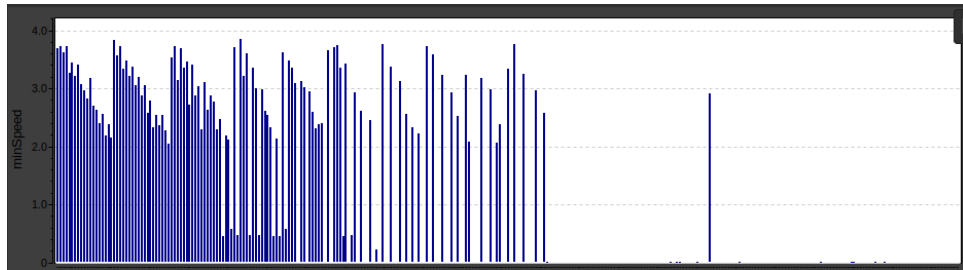


Figura 16– Velocità Minima Scenario 1.

Quindi si può affermare che una distribuzione di nodi malevoli al 40% rende l'infrastruttura non funzionante, i nodi più lenti e conseguentemente le emissioni ne risentono.

### 3.3 Scenario 3

Per la terza simulazione le modifiche effettuate al codice C++ appartenente al layer applicativo mirano alla creazione di nodi malevoli il cui compito è quello di generare messaggi ed inoltrarli all'infrastruttura fingendo di rilevare un incidente.

```
omnetpp.ini  TraCICT11p.cc  SmartRoadCT.rou.xml

Define_Module(veins::TraCICT11p);

void TraCICT11p::initialize(int stage)
{
    DemoBaseApplLayer::initialize(stage);
    if (stage == 0) {
        random_device rd; //Will be used to obtain a seed for the random number engine
        mt19937 gen(rd()); //Standard mersenne twister engine seeded with rd()
        uniform_real_distribution<float> dis(0.0, 1.0);
        sentMessage = false;
        lastDroveAt = simTime();
        currentSubscribedServiceId = -1;
        //mal2=false;
        mal=false;
        /*
        if(dis(gen) > 0.6) {
            mal = true;
            findHost()->getDisplayString().setTagArg("i", 1, "red");
        }
        */

        if(dis(gen) > 0.6) {
            mal2 = true;
            findHost()->getDisplayString().setTagArg("i", 1, "blue");
        }
    }
}
```

Figura 17– Funzione initialize(stage).

Le modifiche includono la funzione **initalize(stage)**, Fig 17, necessaria per generare i nodi che fingeranno l'avvenuta di un incidente; questi verranno marcati da un flag e verranno colorati in blu con una distribuzione del 40%. La funzione che è di maggiore interesse però è la funzione **handlePositionUpdate(cObject\* obj)**, Fig 18, responsabile di gestire la segnalazione di incidenti.





Il comportamento che si nota nell'immagine è l'aumento esponenziale nel numero di nodi che cambiano il suo percorso, perché convinti che nelle prossimità vi sia un ingorgo causato da un incidente. In casi estremi questo attacco potrebbe portare ad interdire una zona o a causare traffico in altre zone della città scombussolando i guidatori ignari del problema.

Le metriche mostrano dei comportamenti non in linea con quello che si potrebbe pensare. Nella maggior parte dei nodi l'emissione di CO2 è in linea con quella dello scenario 0, tuttavia la media ci mostra come il valore è sceso a 188 punti. Mentre le velocità minori risultano quasi invariate con una media di 1.15 punti.

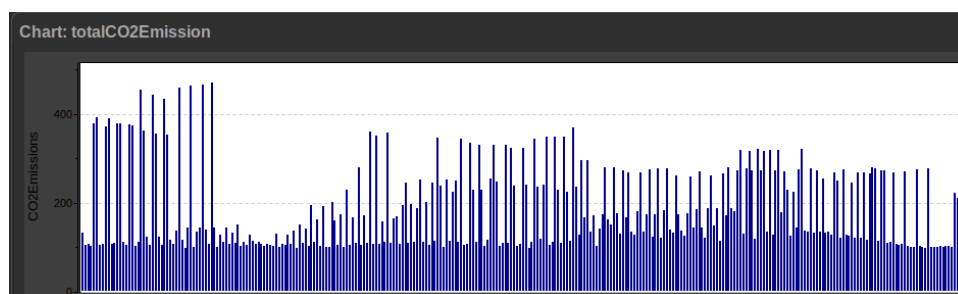


Figura 20– Funzione initialize(stage).

Questo potrebbe significare che le auto hanno giovato dei cambiamenti nei percorsi, perché non sono stati prodotti ingorghi ed il traffico è stato più scorrevole; questo è vero per lo scenario ideale preso in considerazione, infatti, non si può dire lo stesso per uno scenario realistico in cui traffico è casuale, e non statico.

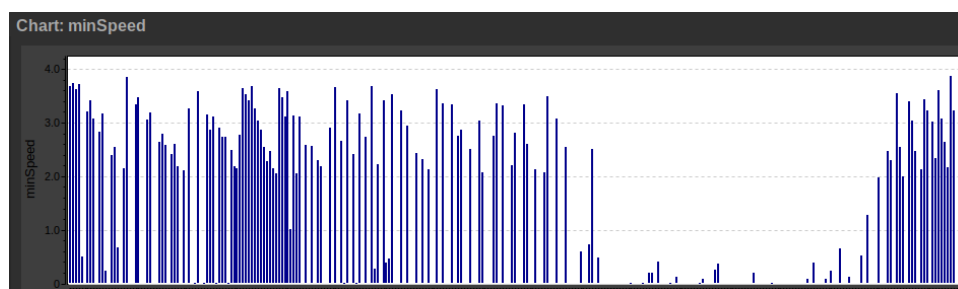
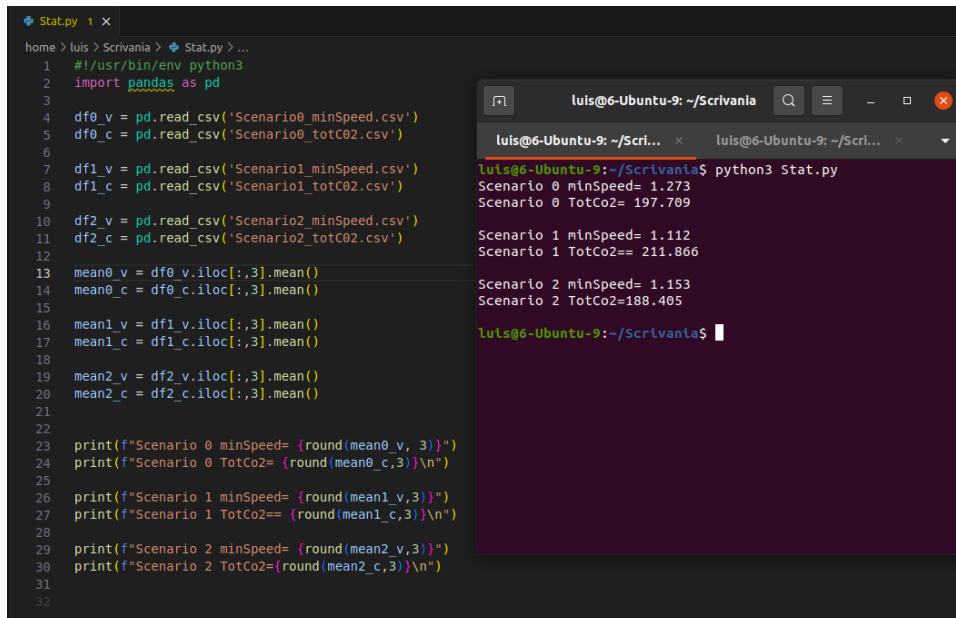


Figura 21– Funzione initialize(stage).

## 4. Conclusioni

In Fig 22, è presente uno screen del codice e dei valori delle medie calcolate.



```
Stat.py 1 X
home > luis > Scrivania > Stat.py > ...
1  #!/usr/bin/env python3
2  import pandas as pd
3
4  df0_v = pd.read_csv('Scenario0_minSpeed.csv')
5  df0_c = pd.read_csv('Scenario0_totCO2.csv')
6
7  df1_v = pd.read_csv('Scenario1_minSpeed.csv')
8  df1_c = pd.read_csv('Scenario1_totCO2.csv')
9
10 df2_v = pd.read_csv('Scenario2_minSpeed.csv')
11 df2_c = pd.read_csv('Scenario2_totCO2.csv')
12
13 mean0_v = df0_v.iloc[:,3].mean()
14 mean0_c = df0_c.iloc[:,3].mean()
15
16 mean1_v = df1_v.iloc[:,3].mean()
17 mean1_c = df1_c.iloc[:,3].mean()
18
19 mean2_v = df2_v.iloc[:,3].mean()
20 mean2_c = df2_c.iloc[:,3].mean()
21
22
23 print(f"Scenario 0 minSpeed= {round(mean0_v, 3)}")
24 print(f"Scenario 0 TotCo2= {round(mean0_c,3)}\n")
25
26 print(f"Scenario 1 minSpeed= {round(mean1_v,3)}")
27 print(f"Scenario 1 TotCo2== {round(mean1_c,3)}\n")
28
29 print(f"Scenario 2 minSpeed= {round(mean2_v,3)}")
30 print(f"Scenario 2 TotCo2={round(mean2_c,3)}\n")
31
32

luis@6-Ubuntu-9: ~/Scrivania
luis@6-Ubuntu-9: ~/Scrivania$ python3 Stat.py
Scenario 0 minSpeed= 1.273
Scenario 0 TotCo2= 197.709

Scenario 1 minSpeed= 1.112
Scenario 1 TotCo2== 211.866

Scenario 2 minSpeed= 1.153
Scenario 2 TotCo2=188.405

luis@6-Ubuntu-9: ~/Scrivania$
```

Figura 22– File utilizzato per calcolare le medie dei valori di emissioni e velocità.

L'analisi dimostra come le VANET siano un ottimo strumento per gestire il traffico autostradale e diminuire i pericoli per i guidatori. Il protocollo 1609.4 è abbastanza sicuro contro attacchi di intercettazione, manomissione, alterazione e replay tuttavia un attaccante in grado di alterare il comportamento delle auto, modificandone l'ECU o tramite vulnerabilità, è in grado di compromettere il sistema di consenso generando flussi di traffico che non sono a beneficio della popolazione e quindi generando confusione e caos. Questo va a colpire lo scopo stesso della VANET, che rimane a meno dei problemi sopra elencati, uno strumento utilissimo per l'organizzazione dei flussi stradali e per il benessere di tutti.