Università degli Studi di Firenze

Scuola di Ingegneria - Dipartimento di Ingegneria dell'Informazione

---

Tesi di Laurea Triennale in Ingegneria Informatica

Bio-inspired Network Forming Protocols Design Applied to Vehicular Fog Computing

Candidato
Stiven Metaj

Relatore
Prof. Francesco Chiti

Correlatore
Dott. Alessio Bonadio

Anno Accademico 2018-2019

It is the long history of humankind (and animal kind, too) that those who learned to collaborate and improvise most effectively have prevailed.

- Charles Darwin

Indice

Elenco delle figure

Listings

Introduzione

Negli ultimi anni, la crescente quantità, a livello globale, di automobili presenti nelle strade ha portato alla ricerca di nuove tecnologie ausiliarie in ambito veicolare. Queste vogliono offrire al conducente supporto in fatto di sicurezza, affidabilità e informazioni, al fine di diminuire il numero di incidenti e situazioni pericolose, aumentando così viabilità e fluidità della rete stradale.

Siamo abituati ai più comuni dispositivi di sicurezza, tra cui airbag, ABS (Anti Braking System), ESP (Elecronic Stability Program) o TCS (Traction Control System) [1] [2]. Questi, insieme alle tecnologie più recenti come l'ISA (Intelligent Speed Adaptation) o l'AT (Attention Assist), sono ormai degli standard internazionali e offrono un'esperienza totalmente diversa e, soprattutto, molto più sicura rispetto a quella di 20 o 25 anni fa.

Riflettendo però, più nel dettaglio, sulle tecnologie elencate, ci si accorge che esse siano pensate per la singola vettura e non per la vera entità in gioco: la rete stradale. Si ha bisogno infatti di tecnologie che utilizzino a loro favore la natura del sistema viario dando origine ad una *cooperazione* tra veicoli. La struttura che offre la possibilità di implementare questo tipo di tecnologie è detta VANET (Vehicular Ad-hoc Network) [3].

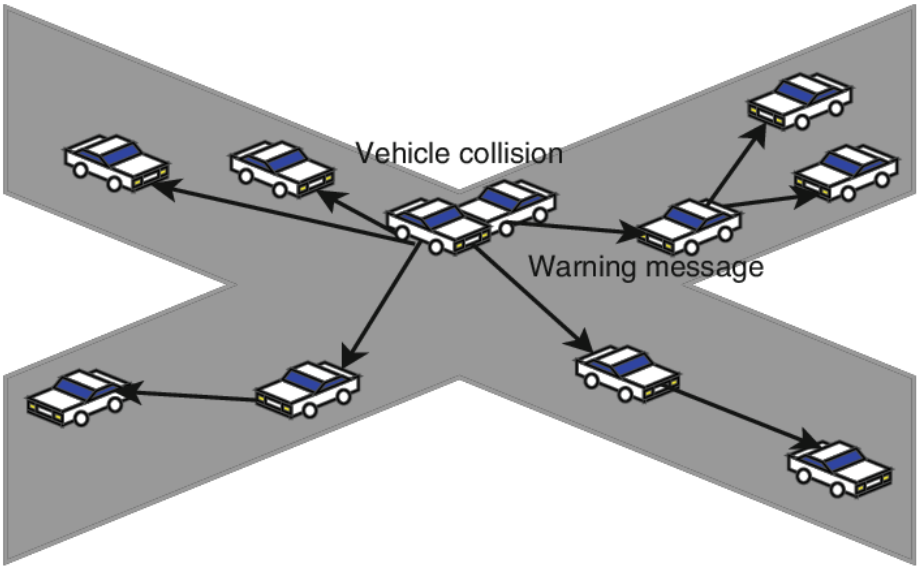

Figura 1: Un possibile esempio dell'utilizzo di una rete VANET

L'uso delle VANET è al centro della ricerca in ambito stradale e le più importanti case automobilistiche, tra cui Volvo, BMW e Mercedes-Benz [4], iniziano ad esplorare possibili metodi per realizzare connessioni tra veicoli.

La tesi si sofferma sulla formazione della rete all'interno di questo modello. In particolare, essendo le VANET reti P2P (Peer-To-Peer), ci chiediamo quale sia il motivo che spinga i nodi a connettersi e in che modo lo facciano. La soluzione trovata è ispirata alla formazione di colonie e micro-colonie batteriche. Vengono presentati i motivi della scelta e la progettazione di un protocollo relativo.

Il documento ha la seguente struttura:

- il Capitolo 1 introduce le reti Ad-Hoc e illustra le principali tipologie di VANET, con particolare enfasi sull'importanza di quest'ultime; si prosegue successivamente con i possibili tipi di informazione che i veicoli si inviano e con l'esposizione dettagliata del problema posto.

- Il Capitolo 2 presenta l'idea di studiare modelli simili presenti in natura; in particolare si parla dell'analogia tra batteri e veicoli.
- Nel Capitolo 3 viene esposto il progetto realizzato evidenziando i tools utilizzati, il codice prodotto e i risultati ottenuti.
- Il Capitolo 4, infine, analizza i risultati traendo conclusioni e idee per progetti futuri.

Capitolo 1

Le reti VANET

In questo capitolo troviamo l'introduzione all'argomento principale della tesi: le reti VANET. Per chiarire l'importanza di queste è necessario introdurre prima le reti Ad-Hoc (dette, più comunemente, reti P2P) e le MANET (conseguenti reti mobili).

Inoltre vengono elencate le principali tipologie di informazioni adatte a comunicazioni all'interno delle VANET. Infine viene presentato il problema posto riguardo al protocollo di formazione della rete.

1.1 P2P, MANET

Le reti Peer-to-Peer, dette anche reti Ad-Hoc, sono reti costituite da nodi completamente autonomi che collaborano tra loro tramite lo scambio di informazioni e messaggi. Come suggerisce il nome, al contrario dei più comuni modelli cloud-based, i nodi che formano la rete hanno lo stesso livello di importanza: ciascuno di essi si comporta sia da server che da client rendendo lo scambio di informazioni più veloce ed esteso, pur abbassando nettamente i costi.

Troviamo, inoltre, 2 tipologie di reti Ad-Hoc:

- reti Ad-Hoc statiche, nelle quali la posizione dei nodi non cambia nel tempo.
- reti Ad-Hoc mobili, dove, invece, i nodi hanno la possibilità di muoversi anche dopo essere entrati a far parte della rete.

Queste ultime vengono anche dette MANET (Mobile Ad-hoc NETwork) [5]. La mobilità dei nodi e la topologia dinamica permettono diversi scenari applicativi [6]. Uno dei tanti scenari è quello veicolare.

1.2 VANETs

Le reti VANET (Vehicular Ad-hoc NETwork), dette anche "Mobile Ad-Hoc Network for InterVehicle Communications", sono una sotto categoria delle MANET. A differenza delle MANET però, i nodi nella rete si muovono a velocità sostenuta e di conseguenza la topologia della rete muta continuamente.

Per renderne possibile il funzionamento si ha il bisogno di installare dei dispositivi wireless all'interno dei veicoli (prendono il nome di OBU, ovvero On Board Unit) ed eventualmente di infrastrutture fisse con collegamento internet (chiamate RSU, ovvero Road Side Unit) [7]. La presenza o meno di RSU distingue 2 diversi tipi di VANET.

1.2.1 Tipi di reti VANET

Seguono le due categorie principali di reti VANET:

- V2V (Vehicle to Vehicle): in questo caso nessuna infrastruttura di tipo RSU è presente nella rete. La comunicazione avviene univocamente tra i veicoli, i quali si scambiano informazioni senza collegamento internet.

- V2I (Vehicle To Infrastructure): in questo caso, invece, troviamo la presenza di RSU all'interno della rete. La comunicazione avviene tra veicolo e infrastrutture. Le informazioni ricevute dal RSU possono essere inviate al cloud ed essere processate.

In particolare, nel primo caso abbiamo una rete distribuita e quindi un'architettura complicata e caratteristiche variabili nel tempo; tuttavia la realizzazione pratica è semplice, infatti non si ha il bisogno di installare nella rete stradale infrastrutture fisiche. Al contrario, nel secondo caso, si ha un'architettura più comune (modello client/server) che ha però bisogno delle infrastrutture RSU menzionate precedentemente. Per RSU si intendono semafori, dispositivi installati ai bordi della carreggiata o anche strutture semi-fisse.

L'obiettivo, in generale, è quello di avere una rete VANET ibrida, in questo modo si ha sia comunicazione tra veicoli, per informazioni di tipo *locale*, sia comunicazione tra veicoli ed RSU per informazioni di tipo *globale* (grazie per esempio al collegamento a internet delle strutture RSU).

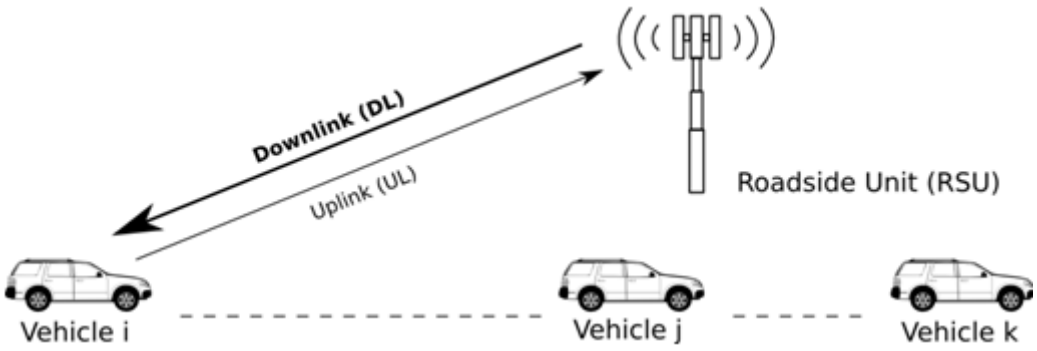

Figura 1.1: Informazioni "cloud-based" in presenza di RSU

1.2.2 Motivi dello studio delle VANET

Una domanda sorge spontanea: perché viene studiata questa tipologia di reti? Quali sono i vantaggi che esse possono portare?

I motivi sono molti; primo fra tutti, la sicurezza. Come precisato nell'introduzione sono già vari i dispositivi che aumentano il livello di sicurezza dei veicoli; l'utilizzo delle VANET permetterebbe un ulteriore ampliamento dei servizi dedicati ad essa.

Inoltre, si ha una grande differenza tra le tecnologie usate fino ad ora e quelle derivabili dall'impiego delle VANET: le prime sono *reattive*, infatti si hanno dispositivi (come fotocamere o sensori) che percepiscono un certo segnale/pericolo e successivamente eseguono una certa operazione (come il rilascio del freno evitando il bloccaggio delle ruote o la frenata automatica in presenza di ostacoli di fronte alla vettura); le seconde invece offrirebbero un servizio *predittivo*: è direttamente la fonte del pericolo (o, più in generale, dell'informazione) che invia, senza ritardi, il segnale utilizzabile a seconda del caso.

Le VANET hanno un ulteriore pregio: superano uno dei difetti più incisivi riguardo alle MANET, ovvero quello del consumo di energia, infatti l'uso dei veicoli permette ai dispositivi installati all'interno di essere continuamente alimentati dalla batteria della vettura [8].

In generale le VANET seguono la corrente degli ultimi anni riguardo ad IoT (Internet of Things) ed IoE (Internet of Everything) [9]; proprio per questo si ha il bisogno di studiarle per apprenderne limiti e potenzialità.

1.2.3 Vehicular Fog Computing

É utile, inoltre, pensare più nel dettaglio a cosa sono le reti VANET rispetto a quello che viene definito come "Internet". Le reti VANET, infatti, sono delle reti

P2P locali dove i nodi appartengono al bordo (edge) della rete Internet globale: proprio per questo motivo si ha nel nostro caso un esempio di Edge Computing [10].

Dal concetto di Edge Computing ci spostiamo a quello di Fog Computing [11], termine coniato da Cisco, che indica una computazione ai bordi della rete (basata quindi sui concetti di Edge Computing) data, però, da un grande numero di device di tipo eterogeneo con una possibile mobilità (da cui il termine "Fog", ovvero nebbia). É interessante pensare a questa "nebbia" composta da device connessi tramite reti P2P nel caso di reti VANET, dove la mobilità è un fattore di grande rilevanza. Da questo particolare caso troviamo, appunto, il termine Vehicular Fog Computing [12].

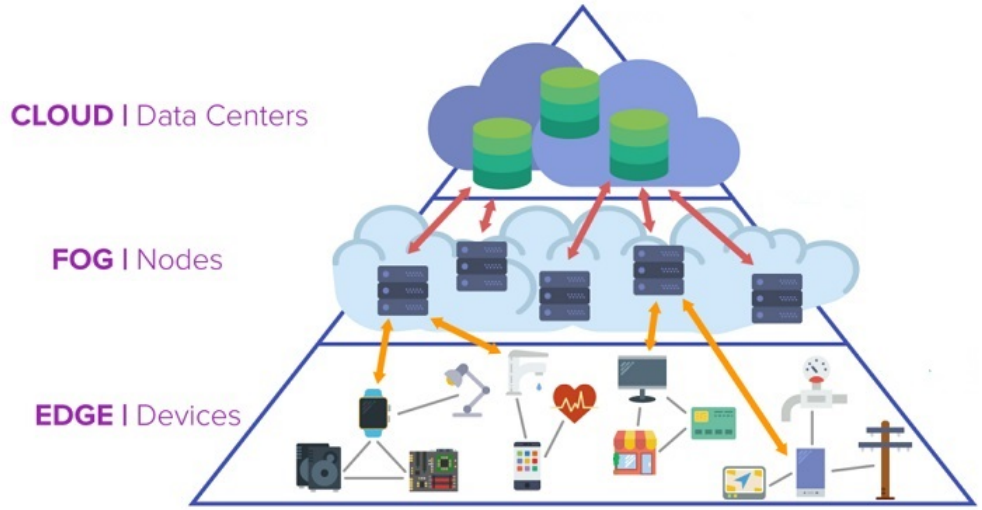

Figura 1.2: Modello piramidale di Cloud/Fog/Edge computing

1.3 Tipologia di informazioni

Andiamo ora a elencare le principali tipologie di informazioni che i veicoli si scambiano:

- **informazioni di carattere generale**: fanno parte di questa categoria tutte quelle informazioni che non indicano pericoli ma che hanno funzioni di avviso, come ad esempio informazioni sul tempo atmosferico, o di intrattenimento, come pubblicità personalizzate in base alla posizione.
- **informazioni di sicurezza**: fanno parte di questa categoria tutte quelle informazioni la cui perdita può compromettere la sicurezza alla guida, come ad esempio informazioni sul comportamento degli altri veicoli o sulle condizioni dell'asfalto.
- **informazioni di tragitto individuale**: fanno parte di questa categoria tutte quelle informazioni relative al tragitto di un veicolo, come ad esempio informazioni su lavori in corso o su veicoli fermi.
- **informazioni di traffico**: fanno parte di questa categoria tutte quelle informazioni che tentano di ottimizzare il flusso veicolare.

Tenendo presente questa distinzione è possibile considerare diversi scenari: nel caso di una brusca frenata in presenza di scarsa visibilità, sono le informazioni di sicurezza ad aver maggiore importanza, mentre nel caso di una congestione sono le informazioni di traffico che permettono agli altri conducenti di modificare il proprio percorso [13].

1.4 Problema del Network Forming

Lo studio di questa tecnologia comporta la ricerca di possibili soluzioni a eventuali problemi. Questi ultimi sono molteplici, per esempio come in tutte le reti i fattori di sicurezza e integrità sono fondamentali: le informazioni non possono perdersi durante la comunicazione e non possono essere manipolate da terzi [14]. Inoltre troviamo importanti studi su quali protocolli utilizzare a livello Network

data l'ampia mobilità della rete (aspetto interessante, ma davvero problematico sotto questo punto di vista) [15] [16].

Per quanto ci riguarda invece, ci occupiamo di un aspetto diverso: la formazione della rete. Ci chiediamo che logica adottare e quale sia la modalità ottimale di creazione della rete. Inoltre, essendo le VANET reti P2P, si ha bisogno di un modello collaborativo di dare/avere; ci poniamo allora la questione del motivo che spinge i veicoli a connettersi fra loro.

Per quesiti simili si ha il bisogno di un modello che rispecchi le caratteristiche considerate, su cui poter dare origine in seguito a possibili riflessioni e deduzioni. Nel nostro caso abbiamo ricercato questo modello nella natura, in particolare, nel campo della biologia.

Capitolo 2

Analogia tra veicoli e batteri

Questo capitolo presenta l'analogia tra batteri e veicoli all'interno rispettivamente di colonie batteriche e rete stradale.

Sono già molti i protocolli ispirati alla natura progettati per MANET e in particolare VANET; essi sono basati sul comportamento, per esempio, di vermi o formiche e, principalmente, vengono utilizzati come protocolli di routing (problema di grande spessore all'interno delle VANET) [15] [17].

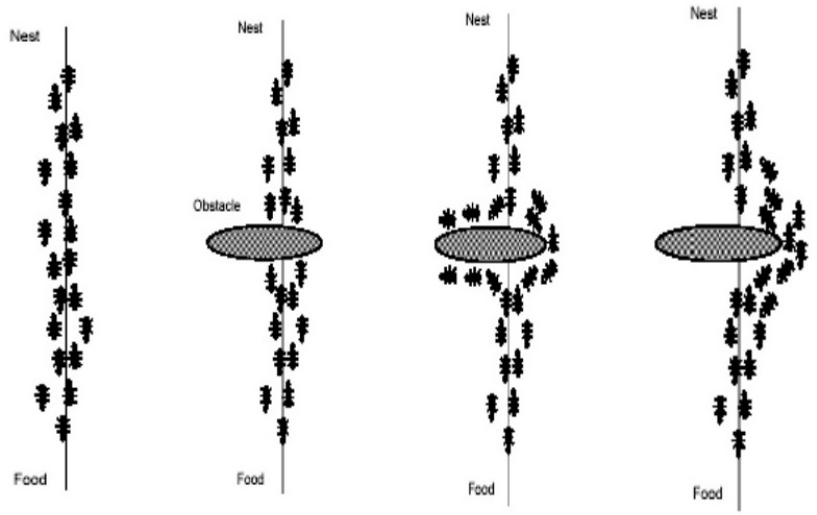

Figura 2.1: Esempio di cooperazione tra formiche

Nel nostro caso si ha bisogno di ideare un protocollo per la formazione della rete. Per farlo si studiano i comportamenti dei batteri sia individualmente che in gruppo.

2.1 I batteri

I batteri sono microrganismi unicellulari caratterizzati dalla presenza di una parete cellulare ed un'assenza di clorofilla. Ne esistono di mobili e di immobili; ovviamente, prendiamo il caso di batteri mobili, che possano così muoversi nello spazio come le auto si muovono all'interno della rete stradale.

Il modello che andiamo ad analizzare è quello dello stress; si pensa ad un batterio come ad un entità individuale (quasi egoista); fino a quando ha risorse disponibili non ha il bisogno di cercare altri batteri a cui connettersi (queste risorse possono essere molteplici, ma nel modello studiato, semplificato il più possibile, si pensa che la risorsa necessaria sia l'acqua). Quando però esso non ha più disponibilità sufficiente di risorse assume il comportamento di un batterio sotto stress, tentando di avvicinarsi ad altri batteri iniziando cosi la formazione della colonia (nei primi momenti solo micro-colonia) batterica [18].

Abbiamo appena trovato il motivo della ricerca di nodi (nel nostro caso veicoli) a cui connettersi: una situazione di stress. Anche un veicolo infatti può trovarsi in una situazione simile: una bassa velocità tenuta per un certo periodo di tempo è, per esempio, un buon motivo per passare ad uno stato di "stress" e chiedere informazioni ai veicoli vicini tentando la formazione di una rete VANET.

2.2 Bacteria Social Behavior

Trovato il motivo per cui una rete veicolare inizia a svilupparsi, è necessario pensare a come la rete, nodo dopo nodo, continui la sua formazione. Si va anche qui ad utilizzare il modello batterico e a studiare quindi quello che si chiama "Bacteria Social Behavior".

Il paradigma di comunicazione di questo modello è quello molecolare il quale, al contrario del normale paradigma di telecomunicazioni, utilizza le molecole per trasportare informazioni. Viene usato questo paradigma proprio perché la comunicazione molecolare è pensata per una trasmissione locale, perfetta nel caso di reti VANET e in generale di reti P2P [19].

Andiamo a vedere più nel dettaglio il comportamento sociale all'interno di una colonia batterica: dopo essere passato allo stato di stress, un batterio inizia ad avvicinarsi ad altri batteri per tentare una connessione; se però questa collaborazione temporanea ha più costi che benefici i batteri si separano per individuare un'altra connessione con la speranza di avere maggiori benefici. Anche se questo è, apparentemente, un modo di agire egoista, la colonia che viene così creata gode di connessioni solide e stabili (un batterio resterà legato al gruppo più a lungo se esso gli porterà più benefici) [20].

Ecco che, anche in questo caso, troviamo che il modello batterico incontra uno dei criteri fondamentali di una rete P2P: il concetto di collaborazione tra nodi. Possiamo adattare quest'ultimo per la progettazione del protocollo di Network Forming voluto.

2.3 Modello del protocollo

Ispirandosi ai concetti di stress e costi/benefici delle sezioni precedenti si deve ora dare forma al modello del nostro protocollo. Prima di farlo però sarebbe neces-

sario procedere con uno studio della correttezza formale dell'analogia presentata in questo capitolo; fortunatamente altri studi hanno già dimostrato che un modello di questo tipo si adatta perfettamente a reti P2P, soprattutto a reti Ad-Hoc mobili come le VANET [20] [18].

Questo modello può presentare caratteristiche simili a quelle che si trovano nella teoria dei giochi; fondamentalmente è proprio di questo che si tratta: un "gioco" in cui i nodi della rete (che siano appunto batteri o veicoli) cercano di massimizzare l'utilità derivata dalle connessione con altri nodi. Inoltre l'utilità di cui si parla è resa trasferibile durante la formazione della rete: i benefici aumentano con l'aumentare dei nodi connessi tra loro [20].

In particolare, nel nostro caso (figura 2.2), un veicolo si trova in situazione di stress se ha un velocità minore o uguale a 10 km/h per più di 5 secondi; ha poca utilità pensare a situazioni di stress a grandi velocità, al contrario in questo modo si considerano vari casi in cui le VANET trovano utilità (una coda, un semaforo rosso, un incidente, un rallentamento...) e si va incontro all'obiettivo di questa tesi che non considera il problema della mobilità della rete.

Una volta che il veicolo è stressato inizia una fase di Network Discovery, in cui i veicoli vicini trasmettono le proprie informazioni (quantità di carburante, distanza al momento della discovery, durata del viaggio corrente, velocità...) al nodo stressato. Quest'ultimo utilizza le informazioni ricevute da ciascun veicolo per il calcolo di un coefficiente normalizzato, con valore tra 0.0 e 1.0 e, successivamente, seleziona il nodo riferito al coefficiente più alto inviandogli un messaggio di Association; se il tutto avviene in modo corretto si ha un messaggio di Acknowledgement in risposta che indica la regolarizzazione della connessione.

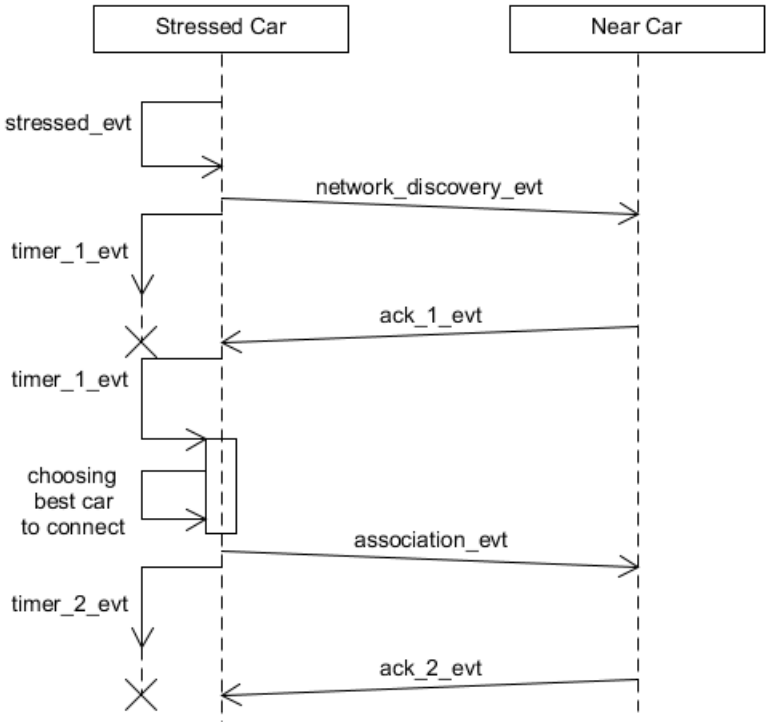

Figura 2.2: Sequence Diagram degli eventi che portano ad una connessione

Per poter formalizzare questo modello si ha bisogno di tools appositi e di framework che permettano la simulazione di reti stradali; tra le varie possibili scelte si è optato per il framework open-source Veins, il quale utilizza a sua volta 2 principali simulatori: OMNeT++ e SUMO.

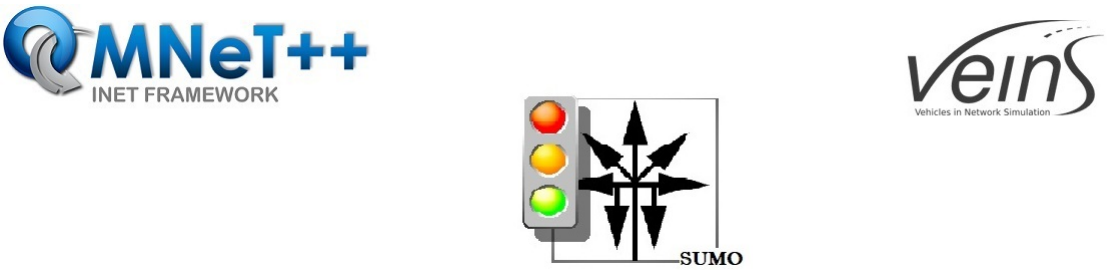

Figura 2.3: I tools usati per la simulazione veicolare

Capitolo 3

Realizzazione del progetto

In questo capitolo viene illustrato il progetto realizzato per simulare il protocollo di Network Forming delle reti VANET.

Il modello del protocollo è quello presentato a fine del capitolo precedente. Di seguito si trovano i dettagli dell'implementazione, i problemi riscontrati (con relative soluzioni adottate) e i risultati dei test effettuati.

Vengono inoltre presentati tutti i tools adoperati nel progetto, infatti abbiamo bisogno di framework appositi per simulare questa tipologia di reti.

3.1 Tools

Il primo importante compito in qualsiasi progetto implementativo è quello di trovare, e soprattutto scegliere, gli strumenti su cui lavorare. La scelta deve tenere conto delle funzionalità offerte, della frequenza degli aggiornamenti e della dimensione della community, a livello mondiale, relativa.

3.1.1 OMNeT++

Il primo dei tools di cui parliamo è OMNeT++. Esso è un framework che implementa in C++ librerie e moduli che permettono la creazione di simulazioni di reti [21]. Inoltre offre un IDE (Integrated Development Environment) basato su Eclipse intuibile e di facile utilizzo.

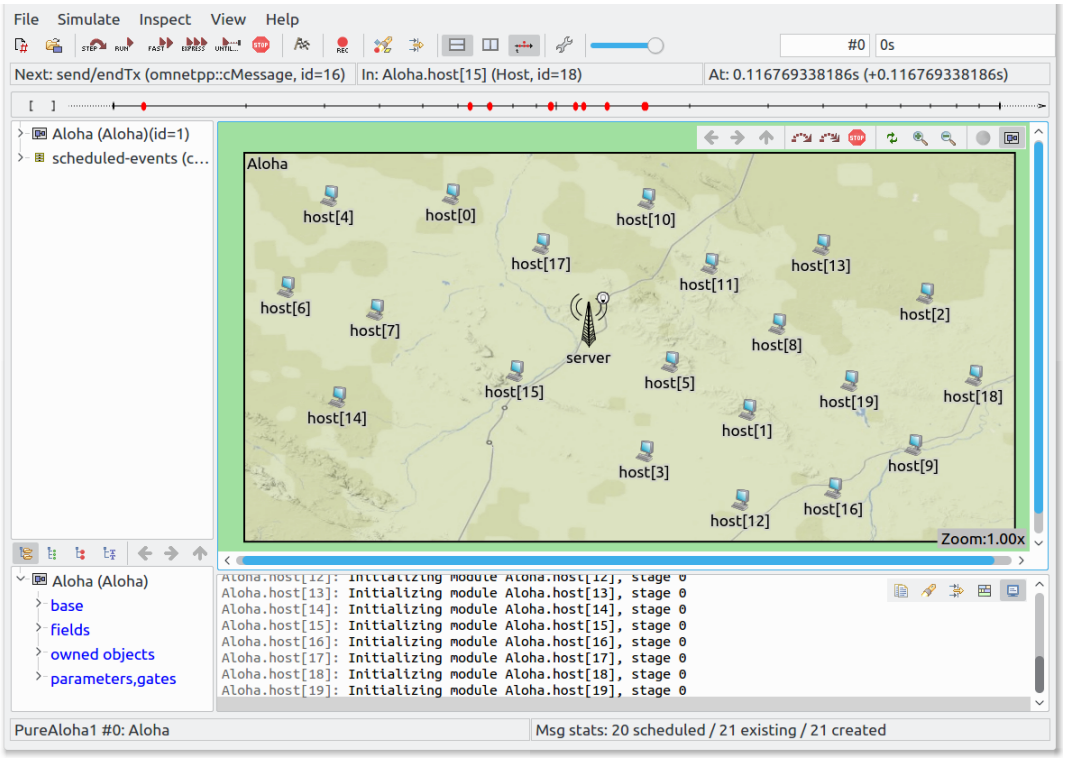

Figura 3.1: Screen di esempio (OMNeT++)

All'interno troviamo una grande varietà di estensioni con relativi linguaggi di programmazione:

- **Moduli Python**: essi sono script Python che permettono la creazione, tramite diversi parametri, dei file usati da SUMO per simulare la rete stradale. Questi possono non essere considerati per progetti come quello presentato in

questo caso, ma in caso di bisogno possono essere modificati per avere un comportamento differente da parte del simulatore.

- **File XML**: questi sono i file in uscita dai moduli python e che, invece, sono l'input di SUMO. Essi vanno a indicare i principali parametri della simulazione: dal banale numero di vetture alla più complicata lista degli ostacoli nella simulazione (ogni palazzo, struttura o edificio all'interno della mappa considerata è convertito in un poligono rappresentato da una riga nel file XML relativo).

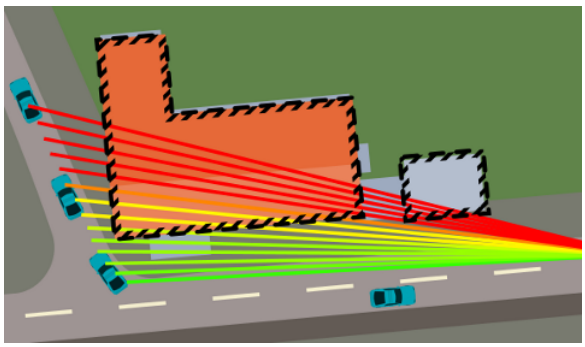

Figura 3.2: Gestione degli ostacoli in Veins

- **Componenti in C++**: questi sono i file che implementato il comportamento di ogni modulo di OMNeT++. Troviamo moduli che implementano i livelli del modello ISO/OSI, le funzioni di ogni nodo della simulazione di rete, le tipoogie di messaggio che i nodi trasmettono, i protocolli di routing ecc...

- **File .msg**: come indicato dall'estensione, sono i file che specificano i tipi di messaggio. Per ogni nuovo file "msg" si ha la creazione, al momento della compilazione, di un file cpp e un file header, letti successivamente in fase di simulazione. Il linguaggio è basato sul C++ ma ha una sintassi differente.

- **Moduli .NED**: questi sono i moduli, scritti in linguaggio diverso dal C++, a più alto livello, i quali creano la vera e propria rete a partire dai diversi sotto-moduli C++.

- **file .ini**: infine, questo è il file che indica i parametri più "generali" riguardo alla simulazione. Per esempio troviamo il limite massimo di secondi della simulazione, oppure le volte in cui si vuole ripetere la simulazione (in caso si voglia poi fare una media su più simulazioni).

Grazie a tutti i moduli presenti e funzioni implementate da OMNeT++ è quindi possibile la creazione di reti più o meno complicate, nonché di svariati tipologie. Purtroppo nel nostro caso si ha bisogno di una rete VANET e quindi molto particolare; ci serviamo quindi di SUMO per simulare l'ambito veicolare di cui necessitiamo.

3.1.2 SUMO

SUMO (ovvero Simulation of Urban MObility) è un pacchetto open-source di simulazione microscopica del traffico stradale, progettato per gestire reti stradali di grande dimensione.

Il suo livello di dettaglio è senza pari: coordina informazioni riguardo carreggiata, corsie, velocità dei veicoli, semafori rossi o verdi, precedenze, incidenti e addirittura (da cui l'aggettivo "microscopico") gestisce particolari come l'uso degli indicatori di direzione prima degli incroci.

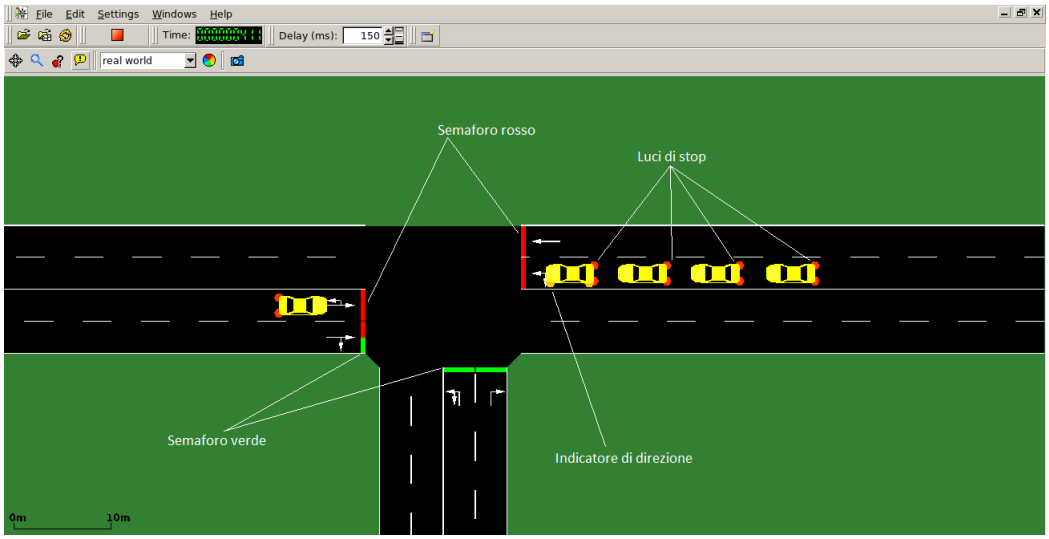

Figura 3.3: Screen di esempio (SUMO)

I moduli di cui è composto non sono stati considerati durante il progetto, infatti il corretto funzionamento del simulatore è stato già, più volte, testato [22] e, a meno di particolari motivi, non si ha il bisogno di applicare delle modifiche.

3.1.3 Veins

Eccoci arrivati al più importante dei "tool" adoperati per questo progetto. Veins (Vehicles in Network Simulaton) è un framework open-source che si occupa di simulare reti veicolari. A prima vista la definizione è molto simile, se non identica, a quella data per SUMO; la differenza, in realtà, è molta, anche se a cambiare è solo una parola.

SUMO, infatti, è un pacchetto che permette la simulazione di una rete veicolare; al contrario, Veins, è un framework che consente di scrivere la propria applicazione volta alla simulazione tra veicoli. Ha moduli personalizzati (rispetto a quelli standard presenti in OMNet++) per i livelli protocollari più bassi lasciando ma-

no libera al programmatore per quanto riguarda il livello application, dove si può progettare il comportamento desiderato dei veicoli all'interno della rete VANET.

Per essere utilizzato ha bisogno di appoggiarsi ai simulatori anticipati nel capitolo: OMNeT++ e SUMO (come spiegato meglio dalla figura 3.4).

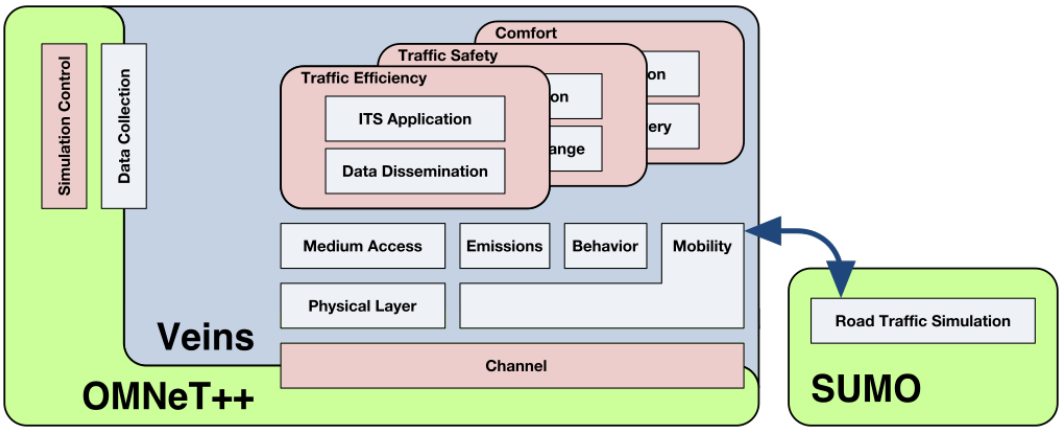

Figura 3.4: Moduli e interazioni di OMNet++, SUMO e Veins

3.1.4 Pycharm

Infine, usati Veins, OMNeT++ e SUMO per implementare il protocollo e salvare i risultati di alcune simulazioni, si ha il bisogno di procedere con un analisi ed una elaborazione dei dati. L'analisi ed elaborazione di dati è sempre stato il punto di forza (oltre al motivo della sua recente ascesa) di Python ed avendo avuto già esperienze precedenti con il linguaggio la scelta è ricaduta su di esso.

L'IDE relativo, usato in questo caso, è PyCharm, prodotto e distribuito dall'azienda ceca JetBrains. Tra le varie features troviamo analisi del codice, debug con interfaccia grafica e test di unità integrati. Anche in questo caso partecipazione della community e frequenza degli aggiornamenti sono i principali fattori che hanno portato a questa scelta.

3.2 Implementazione

In questa sezione si va nel dettaglio dell'implementazione di tutto il progetto; seguono infatti i frammenti di codice principali per quanto riguarda la simulazione.

Prima però, può essere utile tornare alla figura 2.2) per capire meglio il codice che si trova nelle sottosezioni che seguono. Inoltre come ogni protocollo che si rispetti si ha bisogno di una macchina a stati finiti per comprendere il meccanismo che sta dietro alla connessione dei nodi (figura 3.5).

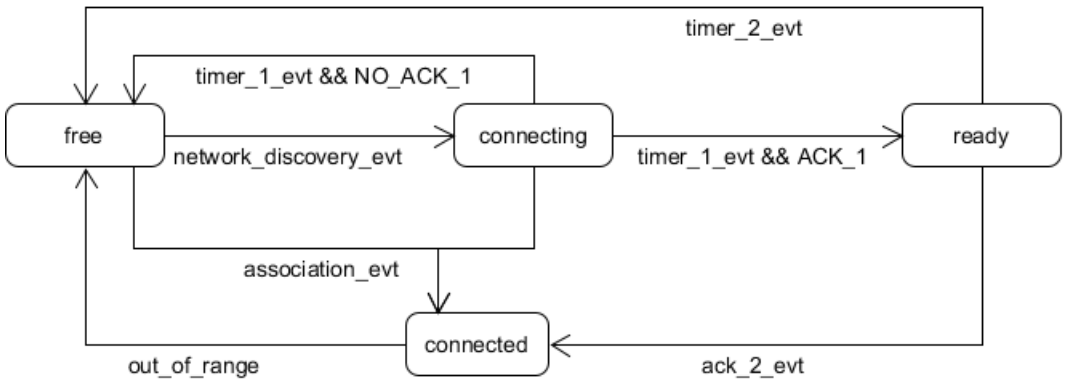

Figura 3.5: Macchina a stati finiti del protocollo

3.2.1 Simulazione veicolare

La simulazione veicolare implementata su OMNeT++ prevede l'uso dello scenario di default offerto da SUMO (figura 3.6): i nodi, i quali sono in questo caso i veicoli che andranno a formare la rete VANET, si muovono, uno dietro l'altro, lungo lo stesso tragitto. La mappa usata nello scenario ha dimensioni pari a 2500 x 2500 metri, mentre il raggio di copertura dell'antenna di ogni veicolo è uguale a 150 metri. L'unica modifica riguarda il parametro relativo al numero di nodi all'interno della simulazione; esso, nel nostro caso, ha un valore pari a 30 (per valori

superiori si aumentava la complessità della simulazione senza andare a migliorare i dati ottenuti).

La simulazione prevede un rallentamento durante il tragitto dei veicoli; sarà proprio questo a stressare i veicoli, dando così il via alla formazione della VANET. Durante la fase di creazione della rete vengono salvati i dati elaborati successivamente; questi sono relativi alle distanze dei veicoli che stabiliscono una connessione (pseudo-distanze per essere più precisi) e ai loro indici (rendendo possibile in fase di analisi la produzione delle matrici di adiacenza relative ai grafi delle reti create; esse saranno usate per le varie misurazioni dipendenti dai grafi).

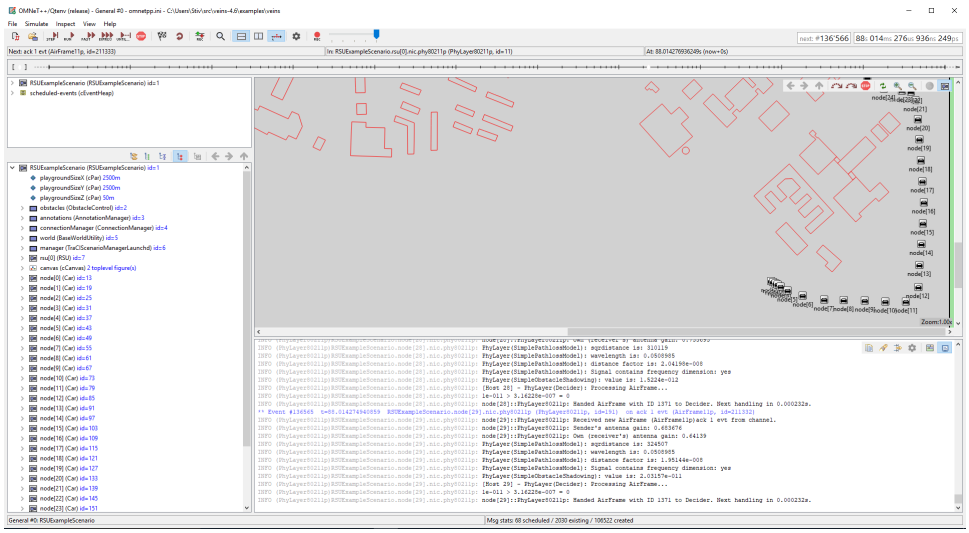

Figura 3.6: Topologia dei veicoli nelle simulazioni OMNeT++

Di seguito troviamo le sottosezioni che vanno a presentare l'implementazione e il codice prodotto su OMNeT++ per quanto riguarda la simulazione in ambito veicolare.

Header

All'interno dell'header troviamo i vari elementi usati poi nel file sorgente; tra questi ci sono la variabile booleana *stressed*, *lastDroveAt* (che servirà per individuare la situazione di stress) e i vari stati possibili (dati dalla macchina a stati in figura 3.5).

Inoltre troviamo delle variabili di tipo *simsignal_t*: esse provvedono al salvataggio dei dati a fine simulazione. Infine *benefitParameter* sarà il parametro che andremo a modificare nelle diverse simulazioni.

```
class MyVeinsApp: public BaseWaveApplLayer {
public:
    enum WaveApplMessageKinds {
        NETWORK_DISCOVERY_EVT,
        ACK_1_EVT,
        ASSOCIATION_EVT,
        ACK_2_EVT,
        TIMER_1_EVT,
        TIMER_2_EVT,
        STRESSED_EVT
    };

protected:
    bool stressed;
    simtime_t lastDroveAt;

    enum State {
        free, connecting, ready, connected
    };

    State state;
    std::list<informations> informationsList;
    std::list<int> connectedList;

    simsignal_t distanceSignal;
    simsignal_t connectedSignal;
    double benefitParameter;

    cMessage* timer_1_evt;
    cMessage* timer_2_evt;
    cMessage* stressed_evt;
};
```

File .NED

OMNeT++ ha bisogno di un file .NED (NEtwork Description) che indichi i parametri del modulo associato. Nel nostro caso andiamo a definire i parametri ad alto livello di cui necessitiamo.

É qui che andiamo a dire ad OMNeT++ che ci saranno un parametro *benefitFactor* e 2 signal per il salvataggio dei dati.

```
simple MyVeinsApp extends BaseWaveApplLayer {
  parameters:
       double benefitParameter = default(0.0);
       @signal[distance]();
       @statistic[distanceStats](title="distanceStats";
                                 source="distance";
                                 record=histogram;
                                 interpolationmode=none);
       @signal[connected]();
       @statistic[connectedStats](title="connectedStats";
                                 source="connected";
                                 record=histogram;
                                 interpolationmode=none);
}
```

Inizializzazione

Nel file sorgente si trovano i metodi della classe *myVeinsApp*. Primo fra tutti è *initialize()*, il quale ha il compito di inizializzare le variabili indicate nell'header. Questo metodo viene chiamato automaticamente da OMNeT++ al momento della creazione, all'interno della simulazione, di ogni nodo (in questo caso di ogni veicolo).

```
void MyVeinsApp::initialize(int stage) {
    stressed = false;
    state = free;
    lastDroveAt = simTime();
    timer_1_evt = new cMessage("timer 1 evt", TIMER_1_EVT);
    timer_2_evt = new cMessage("timer 2 evt", TIMER_2_EVT);
    stressed_evt = new cMessage("stressed evt", STRESSED_EVT);
    distanceSignal = registerSignal("distance");
    connectedSignal = registerSignal("connected");
    benefitParameter = par("benefitParameter").doubleValue();
}
```

Stress del nodo

Il veicolo all'interno della rete è stressato se ha una velocità minore di 5 km/h per più di 5 secondi.

La velocità è ottenuta grazie a *getSpeed()*, metodo presente nei moduli di SUMO; *lastDroveAt* permette di memorizzare l'ultimo momento in cui la macchina ha velocità maggiore di 5 km/h; *scheduleAt()* è il metodo offerto da OMNeT++ per inviare self-message.

Troviamo inoltre il metodo *simTime()*, offerto anche lui da OMNeT++, che ritorna il tempo corrente di simulazione (fondamentale per qualsiasi timer, infatti se voglio un timer di 3 secondi dovrò indicare all'interno di *scheduleAt()* il parametro *simTime()*+3).

```
1  void MyVeinsApp::handlePositionUpdate(cObject* obj) {
2      if (!stressed) {
3          if (mobility->getSpeed() < 5) {
4              if (simTime() - lastDroveAt >= 5) {
5                  stressed = true;
6                  scheduleAt(simTime(), stressed_evt);
7              }
8          } else {
9              lastDroveAt = simTime();
10         }
11     }
12
13     if (stressed) {
14         if (mobility->getSpeed() > 5) {
15             if (simTime() - lastDroveAt >= 5) {
16                 stressed = false;
17             }
18         } else {
19             lastDroveAt = simTime();
20         }
21     }
22 }
```

Gestione dei self-message

Arrivando alle parti più importanti dell'implementazione troviamo il metodo che gestisce i vari self-message. Andando sempre ad osservare la figura 3.5 si

individuano 3 tipologie di self-message:

- *stressed_evt*: si ha l'evento di stress; il veicolo passa allo stato *connecting* e invia in broadcast un messaggio di Network Discovery.
- *timer_1_evt*: questo timer indica che è arrivato il momento di scegliere il veicolo a cui connettersi mandandogli un messaggio di tipo Association. Inoltre troviamo il caso in cui non si è ricevuta nessuna informazione per più di 3 secondi; in questo caso, se si è ancora stressati, si rinizia la fase di Network Discovery.
- *timer_2_evt*: questo timer indica invece che il veicolo a cui abbiamo inviato il messaggio di Association non ha risposto entro 3 secondi (infatti la comunicazione del protocollo è asincrona). Come prima se il veicolo è ancora stressato la fase di Network Discovery ricomincia.

Veins offre la soluzione dello switch case in questi casi: il parametro del metodo è un *cMessage* generico; grazie al metodo *getKind()* si recupera il tipo di messaggio tra quelli inizializzati nell'header.

```
1  void MyVeinsApp::handleSelfMsg(cMessage* msg) {
2      switch (msg->getKind()) {
3
4      case STRESSED_EVT: {
5          if (state == free) {
6              state = connecting;
7
8              WaveShortMessage* networkDiscoveryEvt =
9              new WaveShortMessage("network discovery evt",
10                                  NETWORK_DISCOVERY_EVT);
11
12             populateWSM(networkDiscoveryEvt);
13             sendDown(networkDiscoveryEvt);
14             scheduleAt(simTime() + 3, timer_1_evt);
15         }
16         break;
17     }
18
19     case TIMER_1_EVT: {
20         if (state == connecting) {
```

```
21              if (!informationsList.empty()) {
22                  state = ready;
23
24                  int tempValue =
25                      informationsList.front().benefitFactor;
26                  int tempAddress =
27                      informationsList.front().senderAddress;
28                  for (informations i : informationsList) {
29                      if (i.benefitFactor < tempValue) {
30                          tempValue = i.benefitFactor;
31                          tempAddress = i.senderAddress;
32                      }
33                  }
34
35                  informationsList.clear();
36                  AssociationMessage* assm =
37                  new AssociationMessage("association evt",
38                                         ASSOCIATION_EVT);
39
40                  populateWSM(assm);
41                  assm->setAssociatingAddress(tempAddress);
42                  sendDown(assm);
43                  if (timer_2_evt != NULL) {
44                      cancelEvent(timer_2_evt);
45                  }
46
47                  scheduleAt(simTime() + 3, timer_2_evt);
48              } else {
49                  state = free;
50
51                  if (stressed)
52                      scheduleAt(simTime(), stressed_evt);
53              }
54          }
55          break;
56      }
57
58      case TIMER_2_EVT: {
59          if (state == ready)
60              state = free;
61
62          if (stressed)
63              scheduleAt(simTime(), stressed_evt);
64          break;
65      }
66  }
```

Gestione dei Wave Short Message

Troviamo qui, invece, la gestione dei messaggi ricevuti dagli altri veicoli presenti nella simulazione. In questo caso abbiamo più possibilità riguardo al tipo di messaggio:

- *network_ discovery_ evt*: un veicolo stressato ha iniziato la scoperta della rete. Viene creato e inviato un messaggio di Acknowledgement dove si incapsulano le proprie informazioni. Da notare il metodo chiamato per l'invio del messaggio: se non si usasse *sendDelayedDown()* avremmo una congestione e il veicolo stressato riceverebbe solo una risposta.

- *ack_ 1_ evt*: ricevo le informazioni di un veicolo vicino; le salvo in una lista (di cui andrò poi a scegliere il migliore). Inoltre rigenero il self-message *timer_ 1_ evt* per la durata di 1 secondo (al contrario dei 3 secondi iniziali).

- *association_ evt*: dopo la fase di ricerca della connessione con maggior benefici, un veicolo ha inviato un messaggio di associazione. Se non sono a mia volta in attesa di una risposta aggiungo il mittente alla lista dei veicoli connessi e rispondo con un messaggio *ack_ 2_ evt*.

- *ack_ 2_ evt*: il veicolo scelto per la connessione ha risposto come previsto. Aggiungo a mia volta il suo id alla lista dei veicoli connessi ed elimino il *timer_ 2* precedentemente programmato. Da notare che è in questo caso che la connessione tra 2 veicoli è correttamente stabilita; si può quindi, tramite il metodo *emit()*, salvare le informazioni desiderate (nel nostro caso la distanza dei veicoli connessi e l'id del veicolo mittente, utile successivamente per costruire la matrice di adiacenza del grafo relativo alla rete VANET).

Segue, alla prossima pagina, il codice del metodo *onWSM()*. Anche in questo caso troviamo uno switch case basato sul tipo del messaggio.

```
void MyVeinsApp::onWSM(WaveShortMessage* wsm) {
    switch (wsm->getKind()) {

    case NETWORK_DISCOVERY_EVT: {
        if (state == free || state == connecting) {
            AcknowledgementMessage* am =
                new AcknowledgementMessage("ack 1 evt", ACK_1_EVT);

            unsigned seed =
                    std::chrono::system_clock::now().
                            time_since_epoch().count();

            std::default_random_engine generator(seed);

            double benefitParameter =
                    par("benefitParameter").doubleValue();

            std::uniform_real_distribution<double>
                    benefitDistribution(benefitParameter,
                                        benefitParameter + 0.05);
            std::uniform_real_distribution<double>
                    delayAck(0.001, 0.015);

            populateWSM(am);
            am->setReceivingAddress(getIndex(
                                wsm->getSenderAddress()));

            double temp = benefitDistribution(generator);
            am->setBenefitFactor(temp);
            sendDelayedDown(am, delayAck(generator));
        }
        break;
    }

    case ACK_1_EVT: {
        if (state == connecting) {
            AcknowledgementMessage* am =
                dynamic_cast<AcknowledgementMessage*>(wsm);

            if (am->getReceivingAddress() == getIndex(myId)) {
                cancelEvent(timer_1_evt);

                informations temp;
                temp.benefitFactor = am->getBenefitFactor();
                temp.senderAddress = getIndex(
                                    am->getSenderAddress());
                informationsList.push_back(temp);

                scheduleAt(simTime() + 1, timer_1_evt);
            }
        }
        break;
    }
    case ASSOCIATION_EVT: {
```

```
55          if (state == free || state == connecting) {
56              AssociationMessage* assm =
57              dynamic_cast<AssociationMessage*>(wsm);
58
59              if (assm->getAssociatingAddress() == getIndex(myId)) {
60                  state = connected;
61                  connectedList.push_back(getIndex(
62                                  assm->getSenderAddress()));
63
64                  if (timer_1_evt != NULL) {
65                      cancelEvent(timer_1_evt);
66                  }
67                  AssociationMessage* am2 =
68                  new AssociationMessage("ack 2 evt", ACK_2_EVT);
69
70                  populateWSM(am2);
71                  am2->setAssociatingAddress(getIndex(
72                                  wsm->getSenderAddress()));
73
74                  sendDown(am2);
75              }
76          }
77          break;
78      }
79
80      case ACK_2_EVT: {
81          if (state == ready) {
82              AssociationMessage* am2 =
83              dynamic_cast<AssociationMessage*>(wsm);
84              if (am2->getAssociatingAddress() == getIndex(myId)) {
85                  state = connected;
86                  connectedList.push_back(getIndex(
87                                  wsm->getSenderAddress()));
88
89                  int distance = std::abs(
90                      getIndex(myId) - getIndex(wsm->getSenderAddress()
    ));
91
92                  emit(distanceSignal, distance);
93                  emit(connectedSignal, getIndex(wsm->getSenderAddress
    ()));
94
95                  if (timer_2_evt != NULL) {
96                      cancelEvent(timer_2_evt);
97                  }
98              }
99          }
100         break;
101     }
102 }
```

File di configurazione

Infine nel file *omnetpp.ini* vengono indicati i parametri più di alto livello riguardo, per esempio, alla simulazione o all'esecuzione.

Nel nostro caso si sono indicati i diversi valori di *benefitFactor* su cui eseguire la simulazione. Inoltre viene indicato ad OMNeT++ quante diverse simulazioni realizzare per ogni valore (più sarà alto il numero di *repeat* più i risultati saranno precisi, grazie al Teorema del Limite Centrale [23]).

```
1 *.benefitParameter = ${0.0, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3,
2                       0.35, 0.4, 0.45, 0.5, 0.55, 0.6, 0.65,
3                       0.7, 0.75, 0.8, 0.85, 0.9, 0.95}
4 repeat = 50
```

Avremo quindi i risultati di 1000 simulazioni: 50 per ogni possibile valore del parametro (20 valori in totale).

3.3 Risultati

Andiamo ora a vedere in che modo vengono analizzati i dati tramite Pycharm. Seguono successivamente i risultati ottenuti, sotto forma di grafici (sempre molto utili per una facile e diretta comprensione).

3.3.1 Analisi ed elaborazione dei dati

Una volta eseguite tutte le simulazioni OMNeT++ offre la possibilità di esportare i risultati per ogni simulazione in formato CSV. Nei risultati troviamo righe CSV dove vengono indicati identificativo della simulazione, nome del modulo di cui andiamo a vedere i dati, media dei valori e soprattutto i dati registrati dai *signal* indicati precedentemente.

Grazie all'esportazione in questo formato è possibile analizzare facilmente i dati tramite linguaggi di uso comune come MATLAB oppure, nel nostro caso, Python.

Di seguito si trovano le parti più significative del codice scritto.

Da CSV a dataframes

La prima azione da effettuare è quella di trasformare i dati scritti nel CSV in dati all'interno di un dataframe, che permette una elaborazione più semplice. Per fare questo si è usata la libreria Pandas che permette, tramite il metodo *read_csv*, di fare "facilmente" quello di cui si ha bisogno.

Purtroppo il formato esportato da OMNeT++ non è normalizzato e i dati del CSV non hanno la struttura di default; per questo ho avuto la necessità di elaborare il file per avere in uscita un dataframe utilizzabile in modo più semplice.

```
def getSingleDataframesFromCSV(csvFileName, experiments):
    with open(csvFileName, 'r') as f:
        csvList = csv.reader(f)
        csvList = list(csvList)
    dataFrames = []
    firstLine = ','.join(csvList[0])
    lines = row_count(csvFileName)

    for i in range(lines):
        if i%(2*numCars*experiments) == 1:
            tempList = []
            for experiment in range(experiments):
                string = firstLine
                for j in range(2*numCars):
                    string = string + '\n' +
                        ','.join(csvList[i +
                         (experiment*2*numCars) + j])

                tempList.append(getPandaDataframeFromString(string))
            dataFrames.append(tempList)
    return dataFrames

def getPandaDataframeFromString(s):
    cols = (s.split('\n', 1)[0]).split(',')
    CSV = pd.read_csv(StringIO(s), sep='\n', names=['run'])
    CSV[cols] = CSV.run.str.split(',', n=11, expand=True)
    CSV.data = CSV.data.str.split(',')
    CSV = CSV.iloc[1:]
    return CSV
```

In uscita avremo una lista di liste: *dataFrames* avrà una lunghezza pari ai possibili intervalli del parametro variabili; per ogni possibile valore avremo una lista delle simulazioni fatte con quel valore (più simulazioni avremo a disposizione più sarà preciso il valore medio calcolato).

Da dataframes a grafico

Una volta trasformato il CSV in una lista di dataframe (una per ogni simulazione) dobbiamo implementare il codice che disegna il grafico al variare del parametro scelto. Per questo compito viene usata la libreria *matplotlib.pyplot*.

Segue il codice che produce il grafico relativo alle distanze medie (ci sono altri 2 grafici prodotti che però hanno un codice relativo molto simile a quello delle distanze e quindi inutile da presentare).

```
1  def plotAverageDistanceFromCSV(csvFileName, experiments):
2      dataframes = getSingleDataframesFromCSV(csvFileName, experiments)
3      averageDistances = []
4      max = getMaxDistance(numCars)
5      for i in range(len(dataframes)):
6          sum = 0.0
7          for j in range(len(dataframes[i])):
8              sum = sum + getAverageDistanceFromDataframe(
9                                                  dataframes[i][j])
10         averageDistances.append(sum/experiments)
11     plotGraphic(listOfValues, averageDistances,
12                         "Media delle distanze tra i veicoli connessi",
13                         "Distanza media al variare dell'intervallo" +
14                         "del parametro benefitFactor")
15
16 def plotGraphic(listOfValues, values, label, title):
17     plt.plot(listOfValues, values, label=label)
18     plt.legend(loc='upper left')
19     plt.title(title)
20     plt.xlabel('Intervallo del parametro benefitFactor')
```

Da dataframe a grafo

La parte di codice relativa ai risultati prodotti dall'esperimento è terminata. Rimane però da illustrare il codice prodotto che permette di rappresentare un

dataframe (e quindi una simulazione di OMNeT++) con un grafo. Il grafo mostrerà gli id dei veicoli presenti nella simulazione (che saranno i nodi del grafo) e le connessioni tra i veicoli (rappresentate dagli archi del grafo).

La libreria usata in questo caso è *networx* che, grazie anche a *numpy*, permette di disegnare grafi a partire da matrici (avremo quindi bisogno di trasformare un dataframe in una matrice di adiacenza relativa alla rete).

```
1  def drawGraphFromMatrix(matrix):
2      G = nx.from_numpy_array(matrix)
3      pos = nx.drawing.spring_layout(G)
4      nx.draw(G, pos)
5      labels = {i : i for i in G.nodes()}
6      nx.draw_networkx_labels(G, pos, labels, font_size=10)
7
8  def getGraphMatrixFromDataframe(df):
9      matrix = np.zeros((numCars, numCars))
10     nodeNumber = ''
11     for index, row in df.iterrows():
12         if index % 2 != 0:
13             first = df['data'][index]
14             del first[:1]
15             second = df['data'][index + 1]
16             del second[:2]
17             regex = re.search('.node(.+?).ap', df['module'][index])
18             if regex:
19                 nodeNumber = regex.group(1)
20                 nodeNumber = nodeNumber[1:]
21                 nodeNumber = nodeNumber[:-1]
22                 nodeNumber = int(nodeNumber)
23             for j in range(len(first)):
24                 if second[j] == '1':
25                     matrix.itemset((nodeNumber, int(first[j])), 1)
26                     matrix.itemset((int(first[j]), nodeNumber), 1)
27     return matrix
```

Esempi di grafi in uscita da queste funzioni sono quelli in figura 3.7 e 3.8 (pagina successiva).

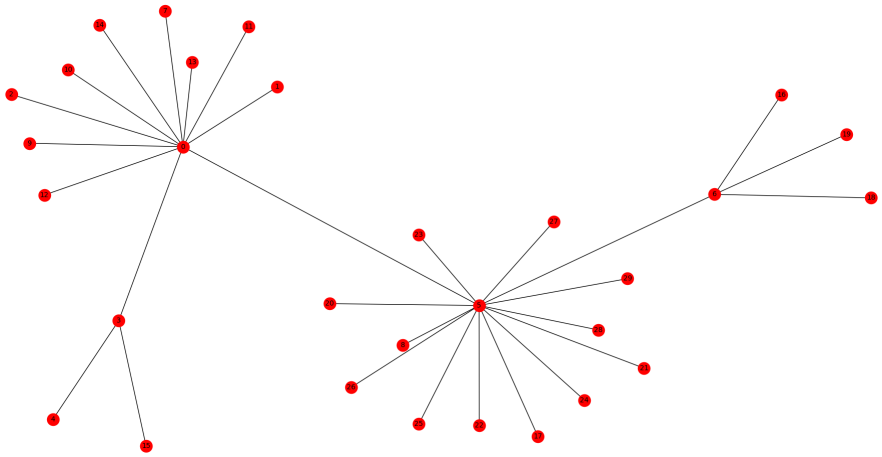

Figura 3.7: Grafo (1) di esempio relativo ad una simulazione

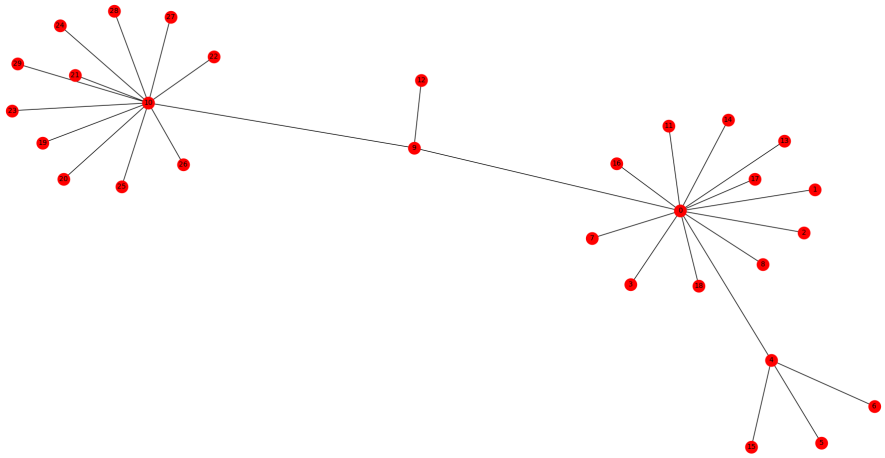

Figura 3.8: Grafo (2) di esempio relativo ad una simulazione

3.3.2 Grafici relativi ai dati

I risultati sono relativi all'analisi di 4 valori: distanza media del grafo, numero di componenti fortemente connesse, branching factor dell'albero (in generale le reti create dal modello implementato non saranno dei grafi con connessioni tra più nodi ma alberi, infatti tornando ai batteri ricordiamo che questi si "connettono" in coppia) e numero medio di hop per la comunicazione tra 2 veicoli casuali.

Distanza media

L'idea è quella di valutare la distanza media tra i veicoli connessi. Il valore di questa non sarà espresso attraverso una lunghezza (in metri per esempio) ma, data la topologia dei veicoli nella simulazione (vedi figura 3.6) sarà calcolata come il numero di veicoli presenti tra i nodi connessi (come se contassimo gli hop della connessione).

Per valutare la distanza pensiamo ai 2 casi in cui la rete ha una forma indesiderata, o comunque una forma degenere.

- Ogni nodo si connette al successivo: in questo caso avremo una forma simile a quella in figura 3.9. La distanza media sarà pari a 1 e in generale i nodi agli estremi della rete saranno penalizzati in caso vogliano comunicare con un nodo lontano.

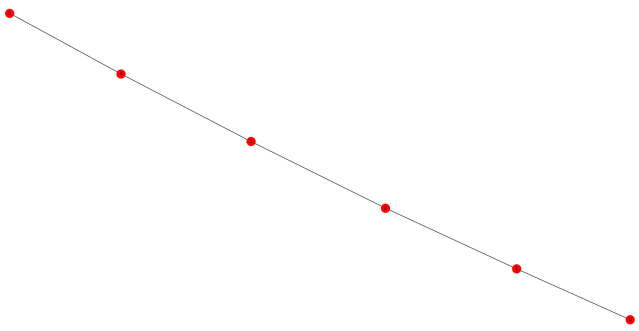

Figura 3.9: Grafo degenere: distanza media = 1

- Ogni nodo dopo il primo si connette a questo: in questo caso avremo una forma simile a quella in figura 3.10. La distanza sarà pari a $\frac{(1+2+...+n)}{n}$ dove n è il numero di archi nella rete. Stavolta i nodi comunicano tramite un numero minore di hop ma la probabilità di congestione aumenta notevolmente; inoltre troviamo una rete con topologia centralizzata, cosa che invece vogliamo evitare per natura della rete P2P.

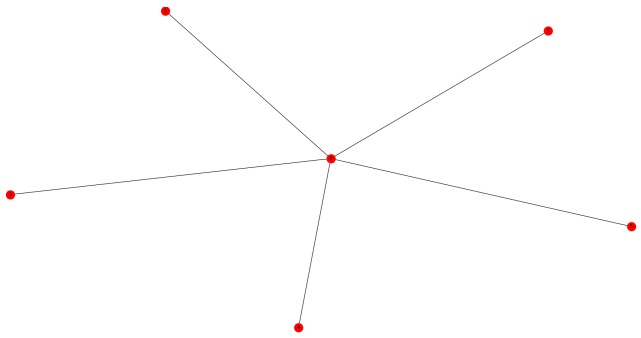

Figura 3.10: Grafo degenere: distanza media massima

Segue il grafico relativo alle distanze medie delle simulazioni. Sull'asse x troviamo gli intervalli del parametro *benefitFactor*; gli intervalli hanno ampiezza 0,05 e i valori sono compresi tra 0,00 e 0,95 (così da arrivare ad un valore massimo di 1,00). Sull'asse delle y abbiamo la media delle distanze medie. Per avere una media corretta sono state fatte 50 simulazioni per ogni intervallo relativo al parametro (un totale, quindi, di 1000 simulazioni).

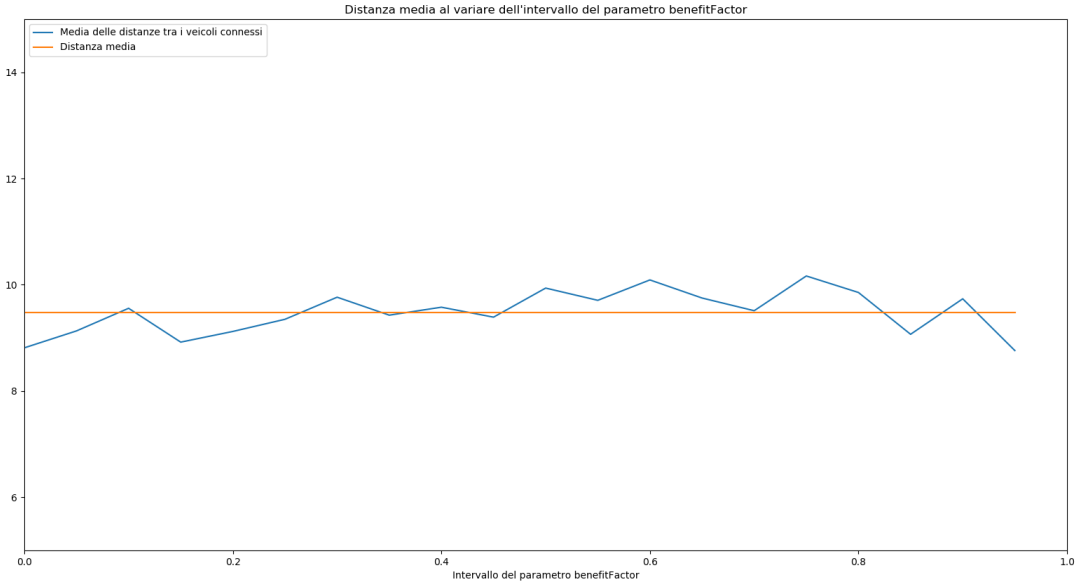

Figura 3.11: Grafico delle distanze medie

Inoltre, parlando di distanze tra nodi, è possibile effettuare un'altra importante misura. Come visto dal grafo in figura 3.7 le reti VANET che si formano (ricordandoci che seguono il modello batterico, il quale ha proprio questa caratteristica) sono composte da nodi principali (in numero minore) e da nodi secondari (in numero maggiore) che si connettono ai primi.

Le reti che godono di questa caratteristica sono dette reti "Small-World", nome che deriva dal fatto che il grafo è composto da piccoli gruppi connessi tra loro tramite un unico arco [24]. La struttura di queste reti permette la formulazione di una legge, con distribuzione esponenziale, che dimostra la robustezza della rete [25] [26].

Quello che allora è stato fatto è la verifica di questa legge (che segue l'idea del principio di Pareto, ovvero della "regola 80/20"). Per farlo è stata estrapolata, per ogni simulazione, una lista in ordine decrescente della somma delle distanze delle connessioni per ogni nodo; successivamente è stata fatta una media su tutte le 1000 simulazioni. In questo caso sull'asse x troviamo i nodi della rete, sull'asse y invece troviamo la somma delle distanze per ogni nodo.

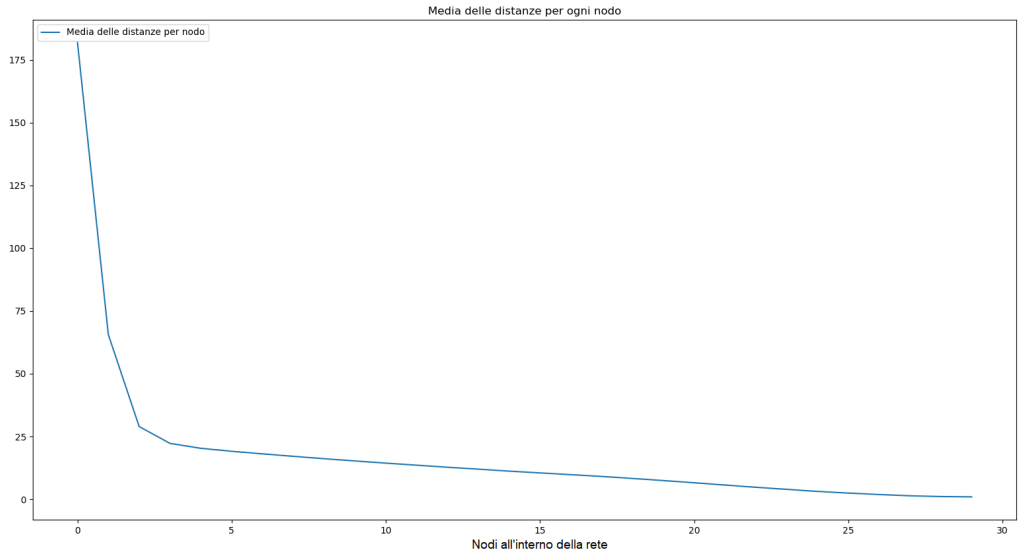

Figura 3.12: Grafico power law delle "Small-World"

Numero di componenti fortemente connesse

Visionando alcune tra le varie simulazioni effettuate è stato notato che a volte i grafi presentano due diverse componenti fortemente connesse, al contrario di altri casi in cui i nodi sono tra loro tutti connessi. Per componente fortemente connessa, dato un grafo G, si intende un sotto grafo massimale di G in cui esiste un cammino orientato tra ogni coppia di nodi ad esso appartenenti.

Questo numero presenta un importante valore data la topologia della simulazione: i veicoli in simulazione si trovano a poca distanza l'uno dall'altro (la causa è un incidente simulato, il quale all'atto pratico può prendere il significato di semaforo rosso, segnale di stop, ecc...) e quindi ci aspettiamo una sola componente fortemente connessa.

Segue il grafico della media del numero di componenti fortemente connesse.

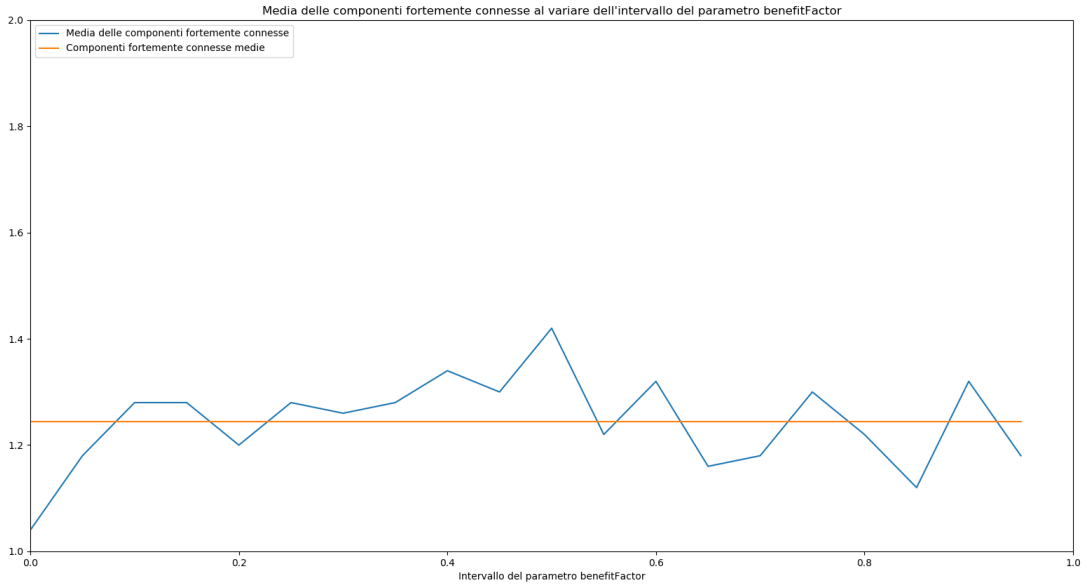

Figura 3.13: Grafico delle componenti fortemente connesse

Branching Factor

Il branching factor, come intuibile dal nome, è il fattore di diramazione di un albero. Dato che il modello batterico prevede che ogni batterio si connetta ad un solo altro batterio il grafo che risulterà in uscita dalle simulazioni sarà privo di cicli e quindi sarà, in realtà, un albero.

Anche qui, come nel caso della distanza abbiamo dei valori limite nei due casi degeneri presi in considerazioni precedentemente:

- nel caso della figura 3.9 il branching factor sarà pari a $\frac{\sum_i^n 1}{m}$ con n uguale al numero di connessioni nella rete e m pari al numero di nodi a cui sono connessi 1 o più nodi.

  In questo caso allora il branching factor sarà uguale a 1.

- nel caso della figura 3.10 il numero di connessioni nella rete sarà lo stesso; al contrario m sarà pari a 1 (tutti i nodi sono connessi ad uno stesso nodo).

  In questo caso allora il branching factor sarà uguale a $\sum_i^n 1$.

Segue il grafico relativo alla media dei branching factor.

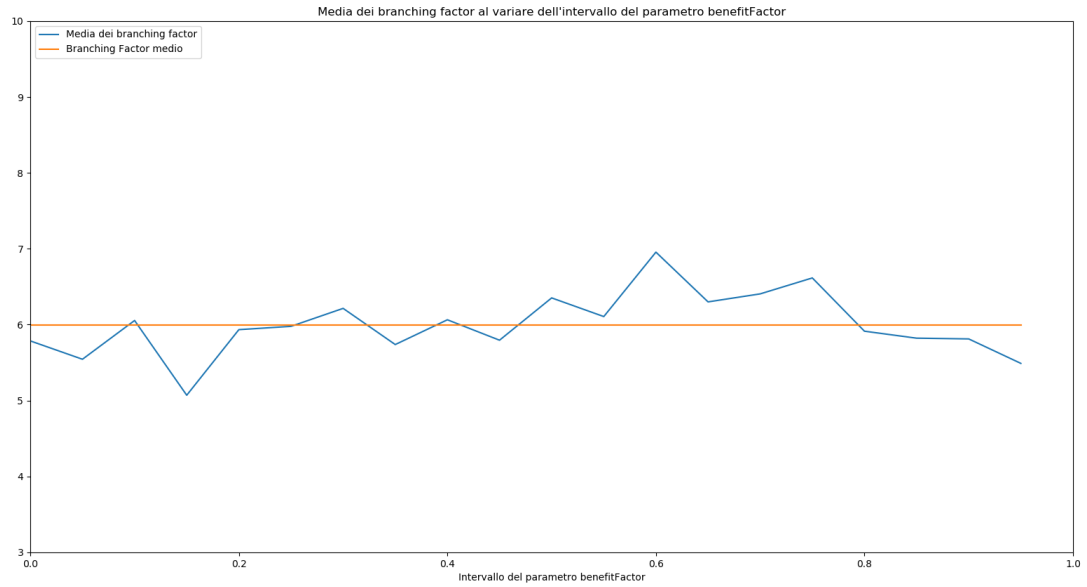

Figura 3.14: Grafico dei branching factor

3.3.3 Numero medio di hop

L'ultima misura effettuata è quella del numero medio di hop, all'interno del grafo, per la comunicazione tra 2 veicoli casuali. Questa misura è fondamentale quando si parla di teoria dei grafi, soprattutto in casi, come questo, di reti "Small-World".

Grazie al lavoro di Watts e Strogatz si dispone di un modello che indica il valore del numero medio di hop nel caso di reti "Small-World" ben strutturate [27]. Questo, idealmente, è pari a $\lg n$, dove n è il numero totale di nodi all'interno della rete.

Ciò significa che le reti di questo tipo sono altamente scalabili e il numero di nodi può crescere repentinamente senza andare a modificare in modo critico il tempo impiegato per le comunicazioni (proprio per questo l'intera rete Internet funziona a dovere, dandoci la possibilità di connettere due tra miliardi di nodi in un numero di hop minore, generalmente, di 10).

Quello che è stato fatto riguardo le nostre simulazioni è proprio il calcolo, al variare dell'intervallo di *benefitParameter*, del numero medio di Hop tra 2 veicoli casuali. Segue il grafico relativo.

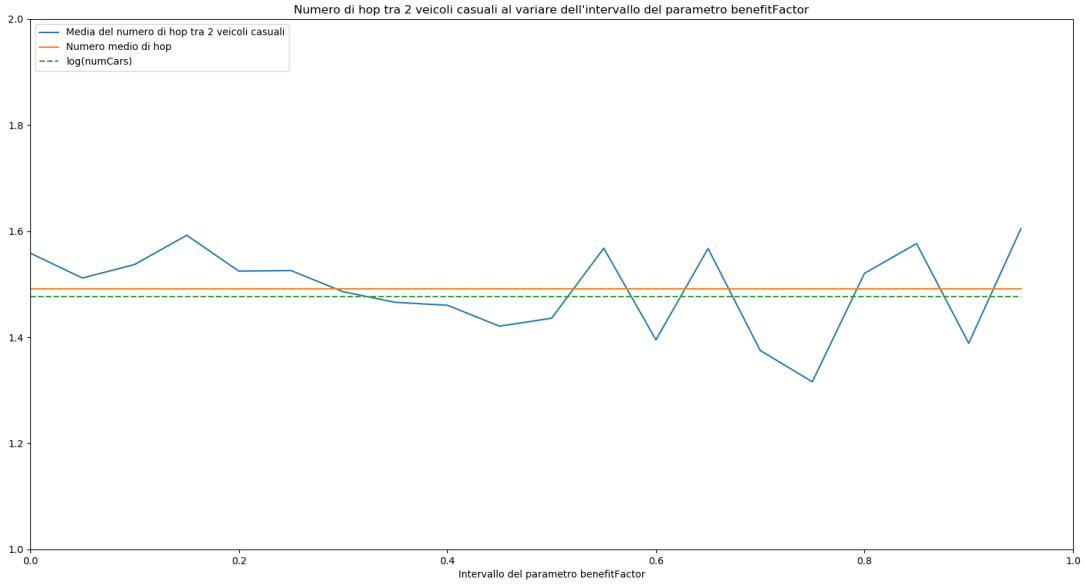

Figura 3.15: Grafico del numero medio di hop

3.3.4 Analisi dei grafici

Dai grafici presentati si nota come, in generale, i valori delle tre misure (distanza media, componenti fortemente connesse e branching factor) non abbiano punti critici: la modifica dell'intervallo di *benefitParameter* non incide particolarmente su queste.

La distanza media tra i veicoli connessi è pari a poco più di 9 lunghezze (dove una lunghezza è la distanza geografica tra 2 macchine consecutive nel caso di un rallentamento, pari quindi a 3/4 metri).

Il secondo grafico (figura 3.12) verifica la legge esponenziale prevista all'interno delle reti Small-World. Questa va ad indicare, appunto, che i grafi in uscita dalle simulazioni sono formati da tanti piccoli gruppi. I risultati sul branching factor

(figura 3.14) indica inoltre il numero di nodi, pari a 6 in media, per ognuno di questi gruppi.

Inoltre, il grafico relativo alle componenti fortemente connesse indica, tramite il suo valor medio, che 1 simulazione ogni 5 rende un grafo con 2 componenti connesse (andando ad analizzare nel dettaglio le componenti si nota che quando succede la seconda componente è formata dagli ultimi 3/4 veicoli della coda); probabilmente questo è un effetto dovuto alla geografia e topologia della simulazione.

Infine, l'importante analisi del numero medio di hop ci porta a dimostrare che il modello, basato sula teoria dei giochi (offerto dagli autori dello studio riguardo alle reti dinamiche basate sull'organizzazione dei batteri [20]), risulta verificare non solo la legge esponenziale che governa le reti "Small-World", ma anche i risultati portati da Watts e Strogatz [27] riguardo la scalabilità di queste (in figura 3.15 si può notare che il valor medio è molto vicino al valore ideale $\lg n$).

Capitolo 4

Conclusioni e sviluppi futuri

In quest'ultimo capitolo vengono presentate le conclusioni dello studio effettuato; un riassunto di quello che è stato scritto, quello che è stato dimostrato e la risposta alla questione del parametro *benefitParameter*: è possibile stabilire un certo comportamento in base al range in cui si trova il parametro oppure si ha una medesima situazione per ogni simulazione effettuata?

Inoltre si vanno ad indicare possibili sviluppi e progetti futuri relativi al lavoro di questa tesi.

4.1 Conclusioni

In questa tesi è stato introdotto l'argomento delle reti VANET in ottica di Vehicular Fog Computing. Ne sono state presentate le potenzialità e le motivazioni che ci portano a volerle utilizzare. É stata inoltre fatta una particolare analogia tra i veicoli in ambito stradale e i batteri appartenenti ad una colonia, riportando gli articoli che dimostrano il corretto rapporto tra i 2 modelli.

Infine è stata proposta una possibile implementazione del modello batterico tramite tools come OMNeT++ e Veins. Dalle simulazioni parametrizzate si sono

costruiti i grafici 3.11 3.13 3.14. Questi mostrano che l'intervallo in cui si trova il parametro non modifica il comportamento della rete: queste misure, a meno di alcuni effetti di bordo dovuti alla geografia della simulazioni (come la presenza di edifici), non variano molto rispetto ai loro valori medi (i quali però offrono importanti informazioni come il numero medio di nodi all'interno dei gruppi, pari a 6).

Al contrario è stata verificata con successo la legge esponenziale riguardo alle "Small-World" che garantisce la robustezza della VANET (figura 3.12).

Inoltre, lo studio sul numero medio di hop ha portato un grafico (figura 3.15) che verifica il modello Watts-Strogatz (fondamentale per scalabilità e effettiva implementazione della rete).

4.2 Sviluppi e progetti futuri

Il lavoro riguardo all'argomento trattato può essere sviluppato maggiormente. I possibili sviluppi futuri sono molteplici, primo fra tutti la creazione di un modello di calcolo che permetta alla normalizzazione delle informazioni dei veicoli di trovarsi all'interno del range di valori ottimali per la creazione della rete.

In secondo luogo si può pensare ad un test reale del modello presentato; in questo modo sarebbe possibile avere un riscontro riguardo i problemi concreti delle reti VANET (in aggiunta a quelli teorici presentati all'interno di questo documento).

4.2.1 Uso di tecnologie in ascesa

Trovo inoltre utile indicare le tecnologie che più possono essere d'aiuto per il funzionamento delle reti VANET e per l'innovazione che esse possono portare.

Prima fra tutte l'intelligenza artificiale; in particolare, l'uso del machine learning potrebbe risultare fortemente vantaggioso: durante la formazione della rete e

la connessione vera e propria dei veicoli vengono inviate grandi quantità di dati di diverso tipo, che possono essere salvate da ogni veicolo all'interno della rete. Si ha quindi la situazione perfetta per l'uso di algoritmi di apprendimento automatico che permettano di prevedere possibili situazioni di pericolo e di emergenza.

Inoltre la tecnologia imminente del 5G potrebbe portare una svolta rispetto a questo tipo di reti. Essa, infatti, permette una velocità di trasmissione senza precedenti, di grande utilità nel caso delle VANET, data la loro grande mobilità e quindi la necessità di inviare informazioni tra veicoli rapidamente. In aggiunta col 5G si ha un rafforzamento delle potenzialità offerte da architetture P2P [28].

Ringraziamenti

Sono tante le persone da dover ringraziare al raggiungimento di un traguardo. Dopo 3 intensi anni di studio, finalmente, ecco che arriva il momento tanto atteso.

In primo luogo ritengo giusto ringraziare il mio relatore, il professore Francesco Chiti. É difficile trovare professori altrettanto disponibili, appassionati e bravi nel trasmettere idee e concetti; con Lei ogni incontro, ogni lezione e persino ogni prova orale, come aveva accennato qualche mese addietro, sono ottime "scuse" per imparare e scambiarsi opinioni. Inoltre ringrazio anche il mio co-relatore, il dottor Alessio Bonadio, che nel momento del bisogno ha saputo risolvere in fretta i problemi riscontrati.

In secondo luogo ringrazio più di tutti i miei genitori. Lo sforzo continuo, il supporto nei momenti difficili, lo sprono a non lasciar nulla al caso e a migliorarsi, la libertà lasciatami nel bilanciare studio e divertimento sono solo alcune tra le varie cose che mi hanno aiutato ad arrivare fin qui.

Come non menzionare, inoltre, la mia ragazza... Il suo appoggio e il suo sostegno mi permettono di dare il massimo anche in situazioni ardue e difficili. Inoltre la sua disponibilità e il suo altruismo consentono bellissime giornate, o semplicemente ore, di pausa dagli studi.

Infine come non ringraziare i compagni di studio e gli amici fuori dal contesto universitario? Insieme abbiamo passato momenti indimenticabili, tra risate, battute e pensieri riguardo al nostro futuro il tempo è volato e lo studio è stato meno

impegnativo e più leggero

Concludendo, grazie a chiunque mi abbia, bene o male, aiutato a raggiungere una tappa fondamentale di un percorso così importante.

Stiven Metaj, 11 Ottobre 2018

Bibliografia

[1] S. Winner, Hermann abd Hakuli, F. Lotz, and C. Singer, "Basic information, components and systems for active safety and comfortr," 2016.

[2] R. Pusca, Y. Ait-Amirat, A. Berthon, and J. M. Kauffmann, "Modeling and simulation of a traction control algorithm for an electric vehicle with four separate wheel drives," vol. 3, pp. 1671–1675 vol.3, 2002.

[3] P. Dahiya and K. Dalal, "State-of-the-art in vanets: The core of intelligent transportation system," vol. 10, pp. 27–39, 01 2017.

[4] B. R. Chang and H. F. Tsai, "Applying visual image and satellite positioning for fast vehicle detection and lane marking recognition," pp. 518–521, Sept 2009.

[5] M. Bisignano, G. D. Modica, O. Tomarchio, and L. Vita, "P2p over manet: a comparison of cross-layer approaches," pp. 814–818, Sept 2007.

[6] R. de Oliveira Schmidt and M. A. S. Trentin, "Manets routing protocols evaluation in a scenario with high mobility manet routing protocols performance and behavior," pp. 883–886, April 2008.

[7] M. Saini, A. Alelaiwi, and A. El Saddik, "How close are we to realizing a pragmatic vanet solution? a meta-survey," vol. 48, pp. 1–40, 11 2015.

[8] A. Rahim, I. Ahmad, Z. Khan, M. Sher, M. Shoaib, A. Javed, and R. Mahmood, "A comparative study of mobile and vehicular adoc networks," 10 2018.

[9] K. A. Khaliq, A. Qayyum, and J. Pannek, "Prototype of automatic accident detection and management in vehicular environment using vanet and iot," pp. 1–7, Dec 2017.

[10] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys Tutorials*, vol. 19, pp. 2322–2358, Fourthquarter 2017.

[11] H. Hong, "From cloud computing to fog computing: Unleash the power of edge and end devices," pp. 331–334, Dec 2017.

[12] M. Sookhak, F. R. Yu, Y. He, H. Talebian, N. S. Safa, N. Zhao, M. K. Khan, and N. Kumar, "Fog vehicular computing: Augmentation of fog computing using vehicular cloud computing," *IEEE Vehicular Technology Magazine*, vol. 12, pp. 55–64, Sept 2017.

[13] L. Bedogni, "Progettazione, modellazione, analisi e sviluppo di protocolli mac in ambito veicolare," 2010-2011.

[14] M. T. Garip, P. Reiher, and M. Gerla, "Ghost: Concealing vehicular botnet communication in the vanet control channel," pp. 1–6, Sept 2016.

[15] K. Mehta, P. R. Bajaj, and L. G. Malik, "Fuzzy bacterial foraging optimization zone based routing (fbfozbr) protocol for vanet," pp. 1–10, Nov 2016.

[16] M. Bhatt, S. Sharma, A. K. Luhach, and A. Prakash, "Nature inspired route optimization in vehicular adhoc network," pp. 447–451, Sept 2016.

[17] T. Q. de Oliveira, T. C. Pessoa, A. R. Cardoso, and J. C. Júnior, "Wchord: A hybrid and bio-inspired architecture to peer to peer networks," pp. 353–358, Oct 2011.

[18] F. Chiti, E. Dei, and R. Fantacci, "Bio-communities comminications paradigms for vehicular social networks," pp. 31–48, 2018.

[19] M. Hasan, E. Hossain, S. Balasubramaniam, and Y. Koucheryavy, "Social behavior in bacterial nanonetworks: challenges and opportunities," *IEEE Network*, vol. 29, pp. 26–34, Jan 2015.

[20] L. Canzian, K. Zhao, G. C. L. Wong, and M. van der Schaar, "A dynamic network formation model for understanding bacterial self-organization into micro-colonies," *IEEE Transactions on Molecular, Biological and Multi-Scale Communications*, vol. 1, pp. 76–89, March 2015.

[21] A. Varga and R. Hornig, "An overview of the omnet++ simulation environment," pp. 60:1–60:10, 2008.

[22] D. Krajzewicz, J. Erdmann, M. Behrisch, and L. Bieker, "Recent development and applications of SUMO - Simulation of Urban MObility," *International Journal On Advances in Systems and Measurements*, vol. 5, pp. 128–138, December 2012.

[23] S. G. Kwak and J. H. Kim, "Central limit theorem: the cornerstone of modern statistics," *Korean J Anesthesiol*, vol. 70, pp. 144–156, Apr 2017. 28367284[pmid].

[24] D. S. Bassett and E. Bullmore, "Small-world brain networks revisited," 2016.

[25] C. Teuscher, "Small-world power-law interconnects for nanoscale computing architectures," vol. 1, pp. 379–382, July 2006.

[26] F. Cornelias and S. Gago, "Deterministic small-world graphs and the eigenvalue power law of internet," pp. 374–379, May 2004.

[27] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, vol. 393, pp. 440 EP –, Jun 1998.

[28] A. A. Khan, M. Abolhasan, and W. Ni, "5g next generation vanets using sdn and fog computing framework," pp. 1–6, Jan 2018.