Università degli studi
di Modena e Reggio Emilia

Facoltà di Scienze Matematiche Fisiche e Naturali
Corso di Laurea Magistrale
in Informatica

Tecniche Communication-saving per l'Acquisizione Dati nel Sistema di Trasporto Intelligente Pegasus

Marcello Pietri

Tesi di Laurea

Relatore:
Prof. Riccardo Martoglia

Anno Accademico 2009/2010

Ringraziamenti

Questo è un altro importante passo
nel cammino della vita.
Ringrazio tutti coloro che
mi hanno permesso di compierlo.

Parole chiave

Communication-saving
Pegasus
V2I
V2V

Indice

Elenco delle figure

Introduzione

I problemi relativi al traffico delle nostre città, caratterizzati dall'aumento del consumo di carburante, dall'inquinamento acustico e ambientale, dall'elevato tasso d'incidenti e dalla conseguente generazione di congestioni, sono problemi ben noti che comunemente deteriorano la qualità di vita delle persone. Lo sviluppo di nuovi concetti del trasporto e della mobilità sono stati promossi dal 7th UE Framework Programme, con l'obiettivo di sviluppare innovative ed efficaci iniziative, riunendo tutti gli elementi di un trasporto pulito, a basso consumo energetico, sicuro ed intelligente.

L'Intelligent Transportation System (ITS) definisce le tecnologie tese ad integrare l'Information and Communications Technology (ICT) alle infrastrutture dei trasporti e dei veicoli, al fine di aumentare la sicurezza, e di ridurre l'inquinamento, i tempi di trasporto, il numero di incidenti e i consumi di carburante. L'ITS nasce quindi dalla necessità di gestire i problemi causati dalla congestione del traffico attraverso una sinergia delle nuove tecniche informatiche per la simulazione, il controllo in tempo reale e le reti di comunicazione.

Nell'ambito ITS è inserito il progetto PEGASUS, il cui obiettivo è sviluppare un sistema di trasporto intelligente realizzando una piattaforma infotelematica per fornire soluzioni di mobilità per una gestione del traffico efficiente ed efficace. Questo progetto nasce dalla collaborazione di diverse aziende, tra cui Octotelematics e Meta System, e dalla collaborazione di diversi gruppi di ricerca universitari tra cui ISGroup, presso il quale è stato svolto il seguente lavoro di ricerca.

Nel contesto così descritto, ogni veicolo appartenente al sistema è equipaggiato con un *On-Board Unit* (OBU), ovvero un'unità di bordo che comunica tramite GPRS con un centro di controllo, e via rete WiFi con le altre unità di bordo nelle vicinanze. La comunicazione mediante GPRS tra OBU e centro di controllo è denominata V2I, ovvero *Vehicle to Infrastructure*, mentre la comu-

nicazione via rete WiFi tra le varie unità di bordo è denominata V2V, ovvero *Vehicle to Vehicle*.

Considerando che nella realtà sono circa settecentomila i veicoli circolanti dotati di OBU, e se ne prevedono almeno un milione in tempi brevi, il numero di comunicazioni e la quantità di dati scambiati risultano ostativi. L'obiettivo delle tecniche *communication-saving* è appunto quello di ridurre il traffico di dati sia tra le varie unità di bordo che tra unità di bordo e centrale operativa.

Come nel caso della compressione dati, vi possono essere tecniche *lossy*, ovvero con perdita d'informazione, che sfruttano le ridondanze nell'utilizzo dei dati, oppure *lossless*, ovvero senza perdita d'informazione, che sfruttano le ridondanze nella codifica del dato. Nel caso delle tecniche *communication-saving* saranno utilizzate principalmente metodologie *lossy*, mentre procedimenti *lossless* potranno essere adottati nella fase immediatamente successiva per comprimere i dati generati dalle prime.

Nello scenario così descritto è inserito questo progetto di ricerca, i cui obiettivi principali sono la riduzione delle comunicazioni V2I mediante l'utilizzo di tecniche *communication-saving* e mediante strategie di aggregazione V2V. L'implementazione sarà sviluppata in modo da essere portabile sull'architettura reale di ogni OBU e la fase di analisi permetterà di scegliere le migliori tecnologie in base al relativo contesto.

Scendendo nei particolari, per analizzare e valutare questi processi di invio e ricezione di dati, e per studiare la formazione di *cluster* tra OBU, è stato necessario progettare e sviluppare un simulatore, poiché non sarebbe stato possibile utilizzare i mezzi fisici necessari per questo scopo, sia per questioni pratiche che per questioni economiche.

La validazione dei risultati ottenuti sarà quindi sostenuta dalla validità dei dati in input, i quali saranno generati da un simulatore di traffico professionale della PTV Vision chiamato Vissim, operante su scenari pressoché identici alla viabilità reale.

Lo sviluppo e l'implementazione delle tecniche *communication-saving* effettuato consentirà la portabilità del codice sull'architettura reale di ogni OBU in modo agevole e veloce, mentre la dettagliata analisi sperimentale dei dati otte-

nuti dalla simulazione permetterà di decretare quali di queste tecniche saranno effettivamente utilizzabili e anche di decidere quali saranno le migliori per i servizi richiesti nei differenti scenari urbani del nostro paese.

La trattazione dei suddetti argomenti in questo documento è suddivisa in due parti: la prima riporta il caso di studio, mentre la seconda riporta il progetto l'implementazione e l'analisi sperimentale, a sua volta seguita dalle conclusioni e dagli sviluppi futuri.
La struttura dettagliata riflette quindi la seguente indicizzazione:

- nel capitolo 1 si presentano le tematiche e le problematiche che verranno affrontate nel corso della tesi;
- nel capitolo 2 si descrivono le tecnologie utilizzate per l'acquisizione dei dati sul traffico e per lo sviluppo del simulatore;
- nel capitolo 3 si procede alla progettazione del simulatore secondo le specifiche SRS e il linguaggio di modellazione UML;
- nel capitolo 4 si implementa quanto delineato nel capitolo 3.1 con il progetto del software;
- nel capitolo 5, infine, si analizzano i dati ottenuti dalle simulazioni e si deducono gli aspetti chiave per questo progetto.

Parte I

Stato dell'arte

Capitolo 1

Introduzione al problema

I problemi da affrontare nello sviluppo di questo progetto riguardano principalmente la mancanza di dati reali sul traffico e un ambiente di simulazione dove poter testare le tecniche *communication-saving* e le strategie V2V proposte.

Partendo dai documenti e dalle specifiche presenti in letteratura, si prosegue presentando i passi che hanno portato alla risoluzione di queste problematiche.

Considerando che questo lavoro di tesi è parte integrante del progetto PEGASUS [29] ed è stato sviluppato per ISGroup [6] secondo le specifiche del documento [9], nei paragrafi successivi si presentano questi aspetti, rendendo l'informazione il più eguale possibile all'originale.

In particolare, in questo capitolo, nella sezione 1.1 sono inizialmente presentati gli obiettivi principali dell'intero progetto, poi, man mano, scendendo nel dettaglio, nella sottosezione 1.1.1 si introduce il lavoro affidato all'ISGroup [6] presentando il documento chiave [9], che a sua volta risulta diviso in ulteriori quattro sezioni. Nella sottosezione 1.1.2 si dettaglia il centro di controllo, mentre nella sottosezione 1.1.3 si introduce il lavoro che verrà svolto in questa tesi. Proseguendo nel documento, questo descrive la memorizzazione dati, riportata alla sottosezione 1.1.4, e trae alcune considerazioni riportate nella sottosezione 1.1.5. Nella sezione 1.2 si sviluppano e si dettagliano gli obiettivi e le tematiche descritte nella sottosezione 1.1.3, essendo questa la parte del documento [9] riguardante questo lavoro di tesi. In dettaglio, nella sottosezione 1.2.1 si espongono gli obiettivi delineati riguardanti il V2I, mentre nella sottosezione 1.2.2 si espongono quelli riguardanti il V2V, per giungere infine alla sottosezione 1.2.3 che presenta le considerazioni sul lavoro da intraprendere.

1.1 Il progetto Pegasus

Figura 1.1: Logo del progetto PEGASUS

[29] Le aree urbane e periurbane del paese sono sempre più congestionate dal traffico con conseguente aumento degli incidenti, dei rallentamenti, dell'inquinamento atmosferico e acustico, del numero degli automobilisti alla ricerca di parcheggi non sempre disponibili, o che devono affrontare vincoli/restrizioni all'accesso ai centri storici. L'insieme di tali fenomeni ha un impatto negativo sia dal punto di vista dello sviluppo economico che sociale del paese. Obiettivo di PEGASUS (ProgEtto per la Gestione della mobilità Attraverso Sistemi infotelematici per l'ambito Urbano, per la Sicurezza di passeggeri, veicoli e merci) è, quindi, la realizzazione di una piattaforma infotelematica integrata che sia l'elemento abilitante di un approccio sistemico alla gestione sostenibile ed in sicurezza dei flussi di persone, veicoli e merci all'interno delle aree urbane. In particolare, la piattaforma si baserà sulla raccolta ed elaborazione in tempo reale dei dati provenienti dai veicoli, dalle infrastrutture e dalle loro interazioni e li sfrutterà per una migliore gestione della mobilità nelle aree urbane e periurbane.

In dettaglio, gli obiettivi previsti sono:

1. il miglioramento della mobilità in termini di miglioramento dell'accesso attraverso la riduzione della congestione e dei tempi di percorrenza:

    - La navigazione intelligente potrà essere garantita attraverso l'invio di informazioni sul traffico in tempo reale ai veicoli, consentendo così al navigatore di bordo di aggiornare continuamente il percorso ottimo. Tale servizio è reso possibile dall'elevato numero di on board unit già installate o in corso di installazione su una molteplicità di veicoli che

trasmettono ad un centro di controllo informazioni legate alla velocità media, garantendo così la conoscenza, in tempo reale, di eventuali rallentamenti o ingorghi. L'acquisizione e l'elaborazione da parte di un centro di controllo di dati provenienti dalle diverse unità e dalle infrastrutture già esistenti permetteranno di fornire un dato attendibile e puntuale, il più prossimo possibile al concetto di real traffic information system. La sempre più sentita esigenza da parte delle persone di avere informazioni sul traffico sarà soddisfatta dal sistema di smart navigation anche tramite la possibilità di fornire informazioni ulteriori in linea con i profili di interesse degli utenti e con le informazioni derivabili dalle altre aree della Piattaforma (es. *green driving*, e *urban mobility*...). Il mezzo con cui verranno fornite le informazioni sarà potenzialmente ogni dispositivo mobile (cellulare o navigatore) che deciderà di usufruire dei servizi forniti dalla Piattaforma.

2. l'incremento dell'efficienza nell'uso degli attuali mezzi di trasporto e delle infrastrutture urbane e periurbane:

   - Gestione flussi, razionalizzazione della viabilità e determinazione dei livelli di inquinamento: la piattaforma infotelematica tramite il suo approccio di sistema permetterà alle Amministrazioni di pianificare le politiche sulla mobilità con una visione di lungo termine e coerentemente con le previsioni sui livelli di intensità dei flussi di spostamento, monitorando allo stesso tempo il reale impatto inquinante della mobilità in relazione alle emissioni derivanti dall'intensità e modalità d'uso dei mezzi di trasporto, ed avendo così a disposizione potenti strumenti per lanciare politiche basate sui concetti "pay as you pollute"e "crediti d'inquinamento". L'*Urban Mobility* in particolare permetterà di sviluppare un sistema integrato di gestione di nodi e servizi strategici della città: come permessi di transito/circolazione elettronici, disponibilità dei parcheggi in tempo reale e da remoto, gestione del flusso delle merci all'interno della rete stradale cittadina. L'azione coerente e coordinata, basata su informazioni raccolte ed elaborate dalla Piattaforma, fornirà uno strumento di gestione fondamentale alle PPAA e a tutti gli attori coinvolti nella gestione della mobilità in ambito urbano per realizzare e monitorare "politiche di mobilità sostenibile".

3. l'incremento della sicurezza e riduzione degli impatti ambientali derivanti dalla mobilità veicolare:

    - L'introduzione e la diffusione del sistema eCall, dell'alert preventivo e delle reti di comunicazione "Vehicle to Vehicle" tramite la diffusione del servizio e degli standard correlati determineranno un miglioramento della sicurezza sulle strade delle aree urbane e periurbane, ma anche dei veicoli. L'impatto sociale e commerciale di questa ASII (azione strategica di innovazione industriale) è indubbio e, se ne trova una conferma di ciò anche nelle azioni di indirizzo, sostegno e promozione supportate dalla Unione Europea tramite la Commissione e le iniziative eSafety.

Il raggiungimento di tali obiettivi è stato affidato alla cooperazione di vari enti e aziende tra cui Octotelematics [28], Metasystem [26], Università di Bologna [39], etc.., e chiaramente all'ISGroup dell'Università degli studi di Modena e Reggio Emilia [6] per il quale è stato svolto questo lavoro di tesi.

Nello specifico, come descritto in [9], il lavoro affidato all'ISGroup [6] è spiegato nelle prossime sezioni. Nella sezione 1.1.1 si introducono le nozioni necessarie per comprendere il documento, nella sezione 1.1.2 si descrive l'architettura del centro di controllo, nella sezione 1.1.3 si descrivono le politiche di acquisizione dati e le possibili tecniche *communication-saving*, nella sezione 1.1.4 si descrive la memorizzazione dei dati e, infine, nella sezione 1.1.5 si presentano le conclusioni relative al lavoro da intraprendere.

1.1.1 Il lavoro affidato a ISGroup

[6, 9] I problemi relativi al traffico nelle nostre città, ovvero l'aumento di consumo di carburante e l'inquinamento (inteso sia come rumore che come emissioni, soprattutto a causa di stop e ripartenze), l'aumento dei tassi di incidenti e la conseguente ulteriore congestione proprio a causa di questi ultimi, sono problemi ben noti che comunemente deteriorano la qualità di vita delle persone. I concetti di sviluppo del settore dei trasporti e della mobilità sono infatti stati promossi nel 7° Framework Program dell'UE, con l'obiettivo di sviluppare iniziative innovative ed efficaci, riunendo tutti gli elementi di un trasporto pulito, a basso consumo energetico, sicuro ed intelligente. Il sostegno alla mobilità urbana di persone e merci in un territorio è così diventato una delle grandi sfide dell'ICT di diverse aree di ricerca, come GIS [36], networking [5], operation research [25], reti di sensori senza fili [47], e telecomunicazioni [22].

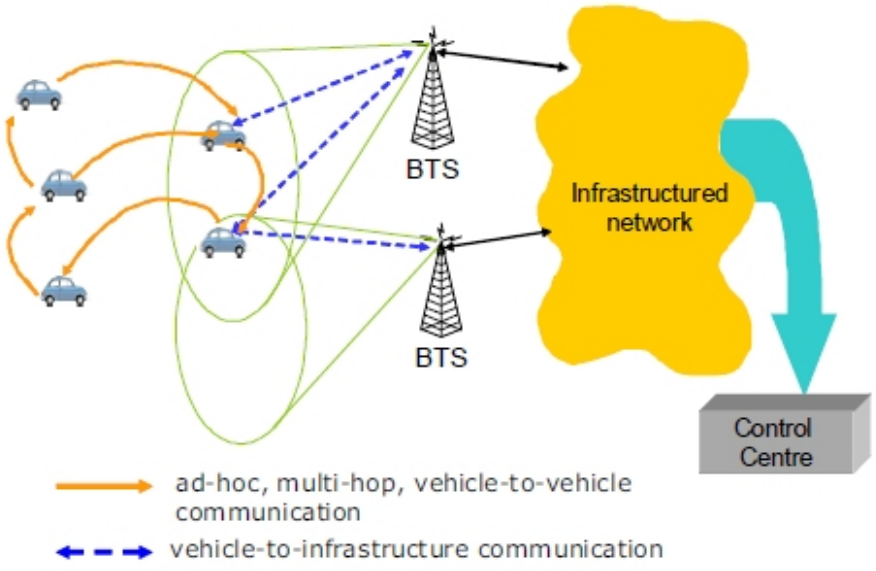

Figura 1.2: Scenario di riferimento del progetto PEGASUS

In questo contesto, il progetto PEGASUS [29] si propone di impiegare sistemi per l'infomobilità in grado di fornire soluzioni per una gestione efficiente ed efficace del traffico. Lo scenario di riferimento è illustrato in Figura 1.2.

I veicoli sono equipaggiati con i dispositivi basati su sensori chiamati On-Board Unit (OBU) che inviano uno stream di dati ricavati da sensori (ad esempio, velocità media, decelerazione improvvisa, etc.) al centro di controllo 1.1.2.

La comunicazione dei dati è effettuata in due fasi: 1) i veicoli sono dinamicamente auto-organizzati in un sistema di comunicazione a cluster V2V-based 2) I cluster heads inviano in broadcast i dati raccolti verso le Stazioni Radio Base (BTS) che compongono una rete con infrastruttura collegata al centro di controllo (comunicazione V2I 1.2.1). Il centro di controllo raccoglie, integra e analizza la grande quantità di dati provenienti dalle OBU, gestisce eventi di interesse (EOI) (Es. incidenti, ingorghi) e produce in tempo reale mappe con punti di interesse (POI) (ad esempio distributori di benzina, parcheggi, cinema, ristoranti, etc.), che vengono a loro volta re-distribuite alle OBU a secondo delle esigenze dell'utente e della posizione. Il sistema intelligente di navigazione dell'OBU sfrutta queste informazioni per fornire agli utenti finali i vari servizi per migliorare la mobilità: la prevenzione degli ingorghi e degli incidenti, i percorsi alternativi, le condizioni meteo sulla strada, la disponibilità di parcheggio, le stazione di servizio più economiche, etc. L'obiettivo principale del progetto PEGASUS [29] è quindi quello di fornire un sistema di trasporto intelligente (*Intelligent Transportation System*), che fornisce informazioni attendibili e tempestive per migliorare la sicurezza e l'efficienza dei veicoli e dei flussi di merci.

Una delle grandi sfide di PEGASUS [29] è la gestione dell'immenso flusso di dati che proviene dai veicoli (vedi 1.1.3). Il centro di controllo ha l'incarico di memorizzare in tempo reale questo flusso che deve anche essere accessibile e manipolato in modo efficiente per rispondere prontamente alle richieste degli utenti. Scalabilità, modularità, e capacità di gestione dei dati geografici, sono quindi i requisiti essenziali che caratterizzano il centro di controllo.

Il lavoro affidato all'ISGroup [6] è quindi lo sviluppo e la gestione di un Data Stream Management System (DSMS) e soprattutto l'acquisizione degli enormi volumi di dati inviati dalle OBU che dovrebbero beneficiare di tecniche *communication-saving* (vedi 1.2). Per consentire poi una gestione flessibile ed estensibile di punti di interesse (POI) e di eventi d'interesse (EOI), il sistema dovrebbe astrarre le caratteristiche specifiche dei diversi scenari. A questo scopo, si propone un livello ontologico dove POIs e EOIs sono concettualizzati e resi indipendenti dal flusso di dati grezzi provenienti dalle OBU e dalle procedure di manipolazione di quest'ultimi. La proposta è un'architettura GIS DSMS a due livelli che soddisfa i requisiti di cui sopra. Nei prossimi paragrafi si dettagliano le scelte effettuate: la sezione 1.1.2 presenta l'architettura del Centro di Controllo, compresa una breve descrizione delle ontologie POI ed EOI. L'acquisizione dei

dati e le tecniche *communication-saving*, tema centrale di questa tesi di laurea, sono descritte nella sezione 1.1.3, mentre la memorizzazione dei dati è trattata nella sezione 1.1.4. Infine, la sezione 1.1.5 presenta alcuni confronti altri studi e trae conclusioni su questo progetto.

1.1.2 Centro di controllo

Nel progetto PEGASUS [29], il Centro di controllo è responsabile della gestione e dello sfruttamento degli stream di elementi provenienti dai veicoli allo scopo di fornire dei servizi di infomobilità (navigazione intelligente, mobilità urbana, sicurezza) agli utenti. Il Centro di Controllo è quindi composto da due moduli principali: il Data Stream Management System e il Service Module, come mostrato nella Figura 1.3. Il Service Module interagisce con il DSMS per ottenere informazioni che utilizzerà per rispondere alle richieste di servizio degli utenti attraverso il Communication Manager; quest'ultimo si avvale di politiche di trasmissione efficienti ed efficaci, per esempio fornendo informazioni solo per veicoli in una determinata regione nella quale si è verificato un evento.

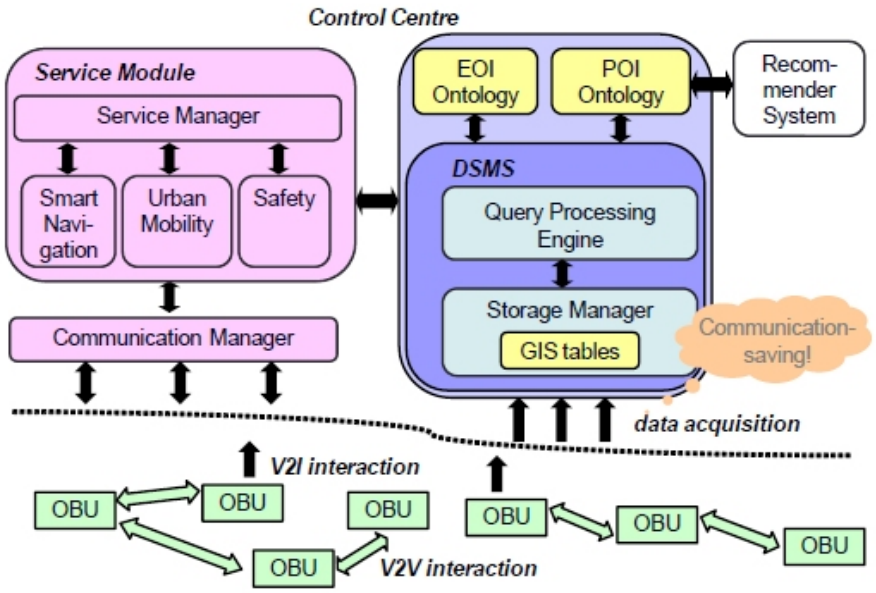

Figura 1.3: Architettura del Centro di Controllo PEGASUS

In questa fase ci si concentra principalmente sul DSMS che è il responsabile dell'acquisizione degli stream di elementi, della loro memorizzazione e della

loro manipolazione. Allo scopo di soddisfare i requisiti di scalabilità necessari per lo scenario PEGASUS [29], il DSMS deve adottare politiche di risparmio della comunicazione per l'acquisizione dei dati, nonché meccanismi flessibili per l'archiviazione dei dati, come un meccanismo flessibile che distingua i dati più recenti, i quali dovrebbero essere disponibili in tempo utile, e perciò necessitano di allocazione nella memoria principale, dai dati storici che possono essere invece memorizzati su memoria secondaria e caricati in cache quando necessario.

Il DSMS si avvale di ontologie per la gestione dei POI e degli EOI. Il disaccoppiamento del DSMS dal livello delle ontologie rafforza l'indipendenza del sistema dalle soluzioni implementative che possono essere adottate, ad esempio, per l'identificazione di eventi e dalla loro gestione che può essere cablata per specifici scenari di riferimento. In altre parole, il livello dell'ontologia permette una gestione flessibile ed estensibile dei punti di interesse e degli eventi introducendo un livello astratto in cui essi sono concettualizzati e resi indipendenti dalla stream di dati grezzi provenienti dalle OBU e agevolando l'introduzione di nuovi concetti (nuove tipologie di eventi o di punti di interesse) che possono essere creati e popolati con nuove istanze di dati.

L'ontologia POI opera con dati storici ed è popolata mediante lo stream di dati provenienti dalle OBU e da ciò che descrive il comportamento degli utenti, ad esempio registra i ristoranti dove l'utente si è fermato una volta in passato. Più precisamente, l'ontologia è inizializzata con un numero di concetti (ad esempio ristorante, parcheggio, etc) importati dalle mappe a disposizione. Quindi, i concetti sono popolati sulla base della storia delle OBU e possono essere definiti nuovi punti di interesse.

L'obiettivo finale è di supportare un sistema collaborativo di raccomandazioni [1], che sfrutti anche le tecniche di profilazione dell'utente (per esempio [27]), per gestire e fare rating dei POI così da includere nelle mappe real-time distribuite alle OBU informazioni POI mirate all'utente.

L'ontologia EOI modella una serie di eventi di interesse (per esempio incidenti, ingorghi), che sono caratterizzati dalla presenza di specifiche condizioni di traffico. Gli eventi sono rilevati sfruttando i dati raccolti dalle OBU, per esempio la tendenza di misure quantitative specifiche come la velocità e accelerazione dei veicoli, e da altre fonti di informazioni come modelli di simulazione e mappe del traffico. Basandosi su tali informazioni, gli eventi possono essere rilevati mediante l'applicazione tecniche comunemente utilizzate come il Dynamic Probabilistic

Models (DPM) e, in particolare, Hidden Markov Models (HMM) (come in [32]). In particolare, l'ontologia descrive gli eventi e le regole che disciplinano le loro generazione. Un evento è quindi rilevato quando i dati raccolti dalle OBU soddisfano le condizioni specificate nell'ontologia. La rilevazione di un evento scatena l'esecuzione delle query basate su eventi ad essi associate. Queste query monitorano ad alta frequenza l'area dell'evento, finché questo non viene risolto con l'obiettivo di fornire agli utenti servizi di infomobilità. Nello stesso modo in cui inizia un evento, la sua risoluzione viene rilevata in base alle condizioni specificate nella corrispondente ontologia e provoca l'arresto di tutte le query associate all'evento.

Una descrizione più dettagliata del DSMS e del processo di acquisizione dei dati sarà fornita nella sezione 1.1.4.

1.1.3 Acquisizione dati

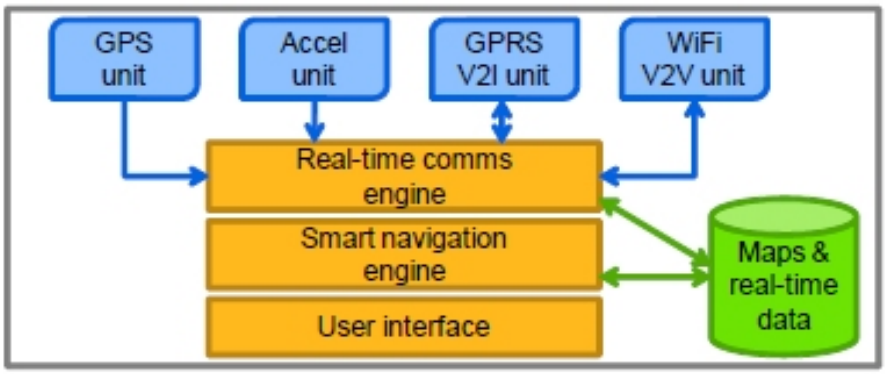

Figura 1.4: Architettura delle On-Board Unit (OBU)

Nell'idea del sistema, ognuno dei veicoli partecipanti è equipaggiato con un dispositivo OBU, il quale è responsabile della raccolta, con strumenti di interazione V2V, dei dati che devono essere inviati al Centro di Controllo (interazione V2I). L'OBU è un dispositivo di elaborazione mobile piccolo e intelligente (vedi Figura 1.4) che acquisisce i dati attraverso unità GPS e accelerometri, ed effettua comunicazioni in tempo reale attraverso la tecnologia GPRS (per V2I) e WiFi (per V2V); tutte le unità di I/O interagiscono con il real-time communication engine, che gestisce tutte le acquisizioni di dati e le operazioni di comunicazione e su cui si focalizzerà la presente sezione. Il dispositivo dispone inoltre di una

flash-memory dove sono memorizzate le mappe statiche della rete stradale, insieme alle informazioni acquisite in tempo reale dal Centro di controllo e/o da veicoli vicini; questa memoria è acceduta dal motore di navigazione intelligente al fine di supportare il guidatore con i servizi richiesti. I dati che vengono acquisiti e gestiti da un OBU sono:

- posizione, velocità, tempo di viaggio e altri dati acquisiti dal modulo GPS (utili per i servizi di traffico in tempo reale);
- accelerazioni ricavate dall'accelerometro (utilizzato per il rilevamento di assistenza di emergenza ad esempio in caso di incidente);
- Notifiche POI / EOI e classificazione POI come notificate da parte dell'utente.

Le OBU aggiornano il Centro di Controllo in tempo reale con opportune notifiche POI / EOI (o di incidenti) e con stream continui di dati derivati dal GPS. Dal momento che i costi di comunicazione GPRS sono ancora significativamente alti, ogni OBU segue un'innovativa strategia ibrida di comunicazione, dove le comunicazioni tra veicoli V2V non tentano di rimpiazzare completamente le comunicazioni V2I (come nella maggior parte della letteratura disponibile [7, 11, 21], uno scenario ancora infattibile per il nostro paese), ma sono invece sfruttate per ridurre il carico sulle comunicazioni V2I. Diverse tecniche di risparmio per la comunicazione V2I e V2V sono scelte e combinate dinamicamente sulla base degli specifici requisiti e condizioni del Centro di Controllo, consentendo al sistema di sfruttare il meglio dei due mondi, al fine di ridurre al minimo i costi globali e massimizzare l'utilità e la tempestività dei servizi offerti.

Tecniche in-node V2I: Mentre i dati POI/EOI possono essere semplicemente inviati a seguito delle richieste del guidatore, senza specifiche elaborazioni intra-nodo, diverse politiche di aggiornamento del server sono disponibili per la trasmissione dei dati derivati dal modulo GPS. Le politiche basate sul campionamento consentono al Centro di Controllo di tenere traccia della posizione di ogni veicolo nel tempo, ad esempio per i servizi assicurativi e di emergenza: oltre al *simple sampling*, che invia i dati a intervalli di tempo/distanza percorsa regolari (ad esempio ogni minuto o ogni 2 km), il *map-based sampling* permette di scegliere comunicazioni più intelligenti sulla base della posizione del veicolo sulla mappa (ad esempio, aggiornamenti frequenti a livello di strada in navigazione

un'area urbana, tassi più elevati in caso di andatura regolare autostradale). Le politiche di *information-need*, d'altra parte, si rivelano preziose per ridurre al minimo le comunicazioni in tempo reale dei servizi di informazione sul traffico, presupponendo che uno dei principali obiettivi del Centro di Controllo è in genere mantenere una stima sufficientemente precisa degli attuali tempi di percorrenza medi dei diversi segmenti di strada (dalle Autostrade alle strade normali). Nella politica *deterministic information-need*, ogni OBU riceve dinamicamente in broadcast una velocità v_b dal Centro di Controllo per ogni segmento di interesse e, per ogni segmento completato, trasmette la velocità media misurata v_m se e solo se $\mid v_b - v_m \mid$ supera una determinata soglia T [11]. Indipendentemente da questa regola, nella politica *probabilistic information-need*, ogni veicolo trasmette le informazioni con una data probabilità p, che può essere aggiornata dal Centro di Controllo al fine di garantire una determinata confidenza nel calcolo della velocità media [3]. Gli autori [6, 9] prevedono che tali politiche potranno coesistere e che saranno rese disponibili attraverso una selezione dinamica. Infine, per l'interazione V2I, è stato pianificato di adattare alcune tecniche promettenti che sono comunemente sfruttate in reti di sensori wireless, come *packet merging* ([46] inviare un pacchetto grande è meno caro che spedire più pacchetti piccoli) e *linear regression* [15] (sfruttando eventualmente le notevoli quantità di ridondanza nelle letture da un veicolo nel tempo).

Tecniche V2V: Nella nostra visione, i veicoli sfruttano il canale di comunicazione gratuito WiFi, se disponibile, organizzando se stessi in cluster e aggregando i propri dati, così da minimizzare le comunicazioni V2I. Questa *self-organization* è fondamentale in un ambiente wireless molto dinamico, mobile e ad alto tasso di *fault*, nel quale la perdita di dati e le collisioni sono molto comuni e il sistema di comunicazione potrebbe facilmente bloccarsi nelle cosiddette "storm broadcast". Similmente a [7], vicini di viaggio di un segmento sono dinamicamente auto-organizzati in un cluster, formando così un sistema di comunicazione V2V clustering-based-multi-channel. I Cluster Members (CMs) comunicano ai Cluster Heads (CHs) del cluster d'appartenenza, e, nel nostro caso, i CHs comunicano i risultati al Centro di Controllo in modalità V2I (opzionalmente può anche eseguire comunicazioni inter-cluster per ridurre al minimo ulteriori comunicazioni). Intra-cluster le comunicazioni WiFi sono quasi immediate e consentono alle OBU una reazione rapida in caso di emergenza (per esempio, una notifica di incidente EOI potrebbe essere istantaneamente trasmessa ai CMs, consenten-

do una reazione rapida, e il CH potrebbe inviare una singola notifica di EOI aggregati al Centro di Controllo). Inoltre, a fini del monitoraggio del traffico, l'esecuzione di protocolli *dynamic distributed aggregation* consentono di eseguire delle funzioni d'aggregazione per stimare misure utili all'interno di un cluster: tali protocolli, come il conteggio dinamico (ad esempio per il numero di veicoli) e la media distribuita (ad esempio per la velocità media) [21], sono essenziali nel nostro scenario, dal momento che, a differenza della maggior parte delle soluzioni d'aggregazione, essi ne' presumono connettività sufficiente a stabilire una infrastruttura di routing, né presumono che i frequenti fallimenti potenziali degli host siano visibili.

1.1.4 Memorizzazione dati

Dal punto di vista della gestione dei dati, lo scenario applicativo considerato nel progetto PEGASUS [29] può essere assimilato ad un'applicazione di streaming data-intensive con requisiti temporali e spaziali. A tal fine, si immagina un GIS DSMS temporale dotato di un linguaggio di interrogazione SQL-like per l'acquisizione e l'accesso a stream di dati tempo-geo-spaziali. Uno dei principali vincoli del progetto PEGASUS è che gli elementi dello stream devono essere mantenuti anche dopo l'elaborazione real-time in quanto il Centro di Controllo dovrà essere in grado di elaborare non solo query real time (comunemente denominate query continue) ma anche query ad-hoc o one-shot ed, eventualmente, analisi OLAP che potrebbero essere effettuate sul sistema, anche dopo l'acquisizione dei dati. Pertanto, non si può adottare una soluzione standard DSMS di gestione dei dati in cui la memorizzazione dei dati è in genere strettamente accoppiata con il query processing [1, 4, 2]: i record sono acquisiti solo se necessari per soddisfare le richieste e conservati esclusivamente per un breve periodo di tempo o eliminate direttamente dal sistema, a meno che il flusso di query richieda esplicitamente una memorizzazione persistente. Al contrario, il GIS DSMS che si propone si fonda su un'architettura a due livelli dove gli elementi dello stream vengono estratti dallo stream della sorgente in input, processati e memorizzati in un contenitore per poi poter essere ulteriormente estratti per l'elaborazione delle query. Per facilitare la comprensione dell'approccio proposto, modelliamo questa situazione attraverso due operatori (Produttore e Consumatore) e uno store tra di loro [7]. Tale store contiene la tabella OBU che ha logicamente una riga per ogni report delle OBU, con una colonna per ogni attributo che le OBU

sono in grado di produrre:

obu(id,time,position,velocity,acceleration,travel_time,. . .,poi,poi_type, . . .,eoi,. . .)

Anche se si è imposto lo stesso schema sugli elementi prodotti da ogni OBU, alcuni valori degli attributi potrebbero essere NULL. Per esempio, NULL sono i valori di default per gli attributi POI ed EOI, a meno che i conducenti riportino informazioni utili su punti o eventi di interesse interagendo in modo esplicito con il navigatore OBU. Inoltre, all'interno dello store vengono memorizzate le mappe della rete stradale insieme alle tabelle:

- poi(id, position, type, . . .) contenente i punti di interesse
- eoi(id, position, type, . . .) contenente gli eventi di interesse attuali.

È importante notare che, mentre obu è una relazione stream contenente tuple tempo-varianti [4], tutte le altre tabelle sono convenzionali relazioni memorizzate.

Q1: SELECT * FROM observations SAMPLE PERIOD 10s	Q2: ON EVENT crash(eoi.position): SELECT o.velocity FROM observations AS o, eoi WHERE distance(eoi.position, o.position) <100m SAMPLE PERIOD 1s
Q3: SELECT time, COUNT(*) from obu WHERE intersect(position,s) GROUP BY SCALE(time AS HOUR) SAMPLE PERIOD 1 h	Q4: CREATE STORAGE POINT rest hst AS (SELECT o.id, o.time, p.id, o.rating FROM obu AS o, POI AS p WHERE o.poi=p.id AND o.type='restaurant')

Tabella 1.1: Esempi di Query

Il produttore effettua regolari inserzioni append-only di nuovi record OBU nella tabella obu. Le osservazioni arrivano ad intervalli di campionamento ben

definiti specificati attraverso la clausola SAMPLE PERIOD del linguaggio di interrogazione. Per esempio, la query continua Q1 riportata nella Tabella 1.1 specifica che ogni OBU dovrebbe riferire la sua osservazione (contenuta nella tabella virtuale observation) una volta ogni 10 secondi. Q1 è una query permanente che viene eseguita continuamente, a meno che non sia esplicitamente fermata. In alternativa, l'esecuzione delle query può essere limitata per un periodo di tempo specifico con la clausola FOR (ad esempio SAMPLE PERIOD 5s FOR 1h).

A volte potrebbe essere necessario modificare il normale processo di acquisizione dei dati, ad esempio quando si verifica un particolare evento. A tal fine, si supportano query basate sugli eventi. Per esempio, la query Q2 in Tabella 1.1 potrebbe essere utilizzata per monitorare la velocità dei veicoli nei pressi di un incidente. Gli eventi di interesse (ad esempio incidente, coda, ecc) sono definiti nell'ontologia EOI. Quando un evento viene rilevato, la tupla corrispondente viene aggiunto alla tabella eoi e tutte le query associate a tale evento vengono attivate. Il sistema esegue ognuna di queste query per il periodo dell'evento, ovvero l'esecuzione termina quando la tupla è eliminata dalla tabella eoi. In generale, le query basate sugli eventi sono essenziali in quanto permettono al sistema di eseguire operazioni specifiche solo quando alcune condizioni esterne si verificano. Per esempio, un servizio di notifica di incidente è supportato al meglio da una query basata sugli eventi eseguita sulla tabella obu, che identifica tutti i veicoli che entrano in un segmento vicino ad un incidente. Infatti, una notifica immediata del sinistro a questi veicoli permetterebbe di uscire dalla superstrada e di evitare la congestione derivante [5].

Ogni query riferita alla tabella obu è un Consumatore nel senso che utilizza i risultati delle query che sono state generate dal Produttore. Le query continue sulla tabella obu seguono l'usuale semantica stream-based: sono sliding-window query che in modo incrementale producono nuove risposte considerando solamente gli oggetti che sono arrivati nell'ultimo periodo di tempo. Per esempio, la query Q3 fornisce uno stream di valori rappresentanti il numero di veicoli che viaggiano sul segmento ogni ora. Il SAMPLE PERIOD argomento della query Q3 specifica sia che la query fornisce aggiornamenti ogni ora sia che ad ogni esecuzione le tuple utilizzate per calcolare il record di aggregazione appartengono tutte al periodo di tempo tra l'ora corrente e quella precedente. In alternativa, è possibile specificare un periodo di tempo differente utilizzando l'operatore

WINDOW. Ad esempio, aggiungendo la clausola WINDOW 2h, la query Q3 produrrà ancora aggiornamenti ogni ora, ma il periodo di tempo di riferimento per i dati non è più un'ora, ma due ore.

Infine, analogamente a molti sistemi di streaming [1, 4, 2], proiezioni e/o trasformazioni di tuple dalla tabella di stream possono essere conservate in materialization point. Tali materialization point accumulano un piccolo buffer di dati che può essere utilizzato in altre query. Per esempio, la query Q4 è una query one-shot che registra i ristoranti dove gli utenti si sono fermati in passato. Questo tipo di query è sfruttata per popolare l'ontologia POI, che è la responsabile del mantenimento della storia del comportamento delle OBU rispetto ai POI visitati.

I report dei veicoli di solito contengono dati con riferimenti geografici e temporali. Pertanto, si è particolarmente interessati a implementare predicati che i GIS comunemente supportano per una analisi spaziale dei dati geografici. Per esempio, Q2 contiene il predicato spaziale distance, che restituisce la distanza tra la posizione dell'incidente e la posizione del veicolo, entrambi rappresentati come punti mentre Q3 usa il predicato intersect per selezionare il numero di veicoli su un segmento. Inoltre, obu è una relazione tempo-variante. Con riferimento alla letteratura temporale [20], il timestamp time di ogni tupla, che rappresenta il momento in cui la posizione è stata trasmessa, è un valore di tipo valid time. Così come il caso spaziale, il supporto ai costrutti delle query temporali è essenziale per un accesso temporale allo stream di dati. Per esempio, l'operatore TSQL2 Scale usato nella query Q3 trasforma il valore time fino ad una granularità di un'ora.

Quando una query è eseguita dal GIS DSMS temporale, il sistema deve fornire i risultati della query che consistono in un singolo risultato, in caso di query ad-hoc o in uno stream di risultati, in caso di query continue. In linea con i principi di design dei DSMS affermati in [7], si è immaginato un sistema dove lo Storage Manager è completamente separato dal Query Processor Engine (QPE). In questo modo, si può implementare un meccanismo di memorizzazione che coadiuva il motore di elaborazione delle query nella loro esecuzione e ottimizzazione e che offre supporto per ridurre i costi di gestione dello storage (riduzione dei tempi di risposta e del consumo di memoria). Al tempo stesso, il QPE può prescindere da questioni specifiche di accesso ai dati e concentrarsi esclusivamente sulla produzione di query-plan efficienti per l'espressivo linguag-

gio di interrogazione brevemente introdotto in precedenza. Più precisamente, lo Storage Manager fornirà le funzionalità di accesso ai dati richiesti e attuerà un meccanismo di archiviazione scalabile che supporti in modo efficiente la grande varietà di esigenze di memorizzazione che emergono dall'applicazione di trasporto intelligente considerata nel progetto PEGASUS [29]. In seguito sono state descritte brevemente le principali esigenze di memorizzazione. In primo luogo, a differenza dei DSMS standard, i dati non scadono mai, cioè non usciranno mai dal sistema. Ciò significa che lo Storage Manager deve essere in grado di gestire una grande quantità di dati. Allo stesso tempo, alcuni dei servizi che vorremmo sostenere, come l'allerta d'incidenti, hanno fabbisogni real-time. Questi servizi di solito si affidano a query continue che possono specificare finestre di dimensioni diverse sugli stessi dati. Pertanto, lo Storage Manager dovrebbe garantire l'accesso rapido ai valori dei dati più recenti. Le query basate sugli eventi necessitano di memorizzare gli eventi per un determinato periodo di tempo e di introdurre meccanismi di sincronizzazione sui contenuti memorizzati. Infine, è noto che l'elaborazione di predicati non convenzionali, come quelli spaziali e temporali, è particolarmente costosa in termini di tempo. In questo contesto, lo Storage Manager dovrebbe fornire alcune strutture di indicizzazione per accelerare i tempi di risposta. Analogamente ad altri DSMS [1, 4], le tuple della tabella obu richieste dalle operazioni con finestra saranno fondamentalmente mantenute in strutture dati in memoria principale, come code circolari [1] o liste collegate [7]. Ogni finestra scorrevole sarà associata a due puntatori che delimiteranno l'elenco delle tuple che non sono state ancora elaborate. Ogni volta che una tupla è stata consumata da tutte le finestre scorrevoli in corso diventa un elemento "storico". Gli elementi storici saranno mantenuti in strutture dati in memoria secondaria. In questo contesto, invece di spostare un elemento storico alla volta, si cercherà di ottimizzare l'operazione di swapping introducendo approcci lazy di rimozione che muoveranno più elementi contrassegnati dalla memoria principale alla memoria secondaria. Infine, si definiranno indici in memoria principale per supportare in modo efficiente diversi tipi di predicati.

Analogamente a [7], lo Storage Manager metterà a disposizione del QPE un'interfaccia di accesso consistente in operazioni di accesso ai dati per-elemento e per-attributo come read e update. Tali operazioni saranno utilizzate per costruire il query-plan. In questo modo, il QPE attuerà vari approcci d'ottimizzazione delle query, dal raggruppamento di query continue simili [2], allo sfrut-

tamento dei pattern di aggiornamento delle query continue in esecuzione [12] e alla schedulazione degli operatori [4].

1.1.5 Note e considerazioni

I Sistemi di Trasporto Intelligenti presentano sfide che coprono vari settori di ricerca, quali i GIS, le reti di sensori wireless e le telecomunicazioni, solo per citarne alcuni. Recenti studi negli Stati Uniti dimostrano che l'attuale gamma di tecnologia WiFi (100-300m in scenari all'aperto) accoppiata con una penetrazione di mercato del 3%-10% delle OBU sono sufficienti a garantire una comunicazione efficace V2V [11]. In Italia, il numero delle passate generazioni (comunicazione solo V2I) di veicoli attrezzati con OBU è in costante aumento e ha raggiunto di recente 750.000 unità[1] installate (quasi il 2% di penetrazione, vedi). Una transizione graduale verso la nostra innovativa strategia ibrida di comunicazione V2I/V2V, appare non solo possibile ma anche uno scenario promettente, dove V2V abbatterebbe significativamente la comunicazione V2I, attualmente molto elevata nei costi. In particolare, abbiamo intenzione di adeguare le politiche avanzate di risparmio della comunicazione, come le dinamiche di aggregazione e le soluzioni di auto-organizzazione, che sono state impiegate con successo in diversi scenari di UE e Stati Uniti. Per la prima volta, tali politiche saranno combinati e rese disponibili attraverso una selezione dinamica sulla base dei requisiti del Centro di controllo e su un ITS italiano.

A conoscenza degli autori [6, 9] tuttavia, non esistono proposte che sono esplicitamente dedicato ai problemi di gestione dei dati delle ITSs. L'unico lavoro che si concentra su informazioni sul traffico è [5], che però specifica solo un benchmark per il DSMS. Le aree di ricerca più legate al contesto dell'applicazione PEGASUS [29] sono quelle relative ai DSMS e alla gestione di query event-driven. Negli ultimi anni, c'è stata una vasta quantità di lavoro nel campo della gestione degli stream. Diverse ricerche hanno costruito prototipi di DSMS (ad esempio, Aurora [1], STREAM [4], NiagaraCQ [2]), ognuno proponendo i propri modelli di dati e di query e le proprie soluzioni di elaborazione delle query continue. Inoltre, molti altri lavori (ad esempio [12]) approfondiscono il problema dell'elaborazione di query continue. Tuttavia, tutti questi approcci hanno un accoppiamento stretto tra elaborazione delle query e la gestione della

[1] http://traffico.octotelematics.it

memorizzazione, dovuto all'integrazione della gestione dello storage all'interno del motore di elaborazione degli stream. Questa soluzione di gestione dati non si addicono al nostro contesto di ricerca in cui gli stream di dati devono essere conservati oltre la loro elaborazione in tempo reale. Per quanto a nostra conoscenza, [7] è l'unico documento che discute la possibilità di un sistema loosely-coupled. In questo documento [9], è stata adottata questa soluzione e si sono discusse in dettaglio le problematiche relative ad un archivio dati contenente elementi di uno stream che non scadono mai.

Una grande quantità di letteratura è disponibile in materia di individuazione e gestione di eventi in diversi campi di ricerca come l'RFID [32, 42], reti di sensori [24], DSMS [2], query continue [23] e sistemi publish/subscribe [3]. Per quanto riguarda le funzionalità event-driven del sistema descritto in 1.1, gli autori [6, 9] si sono ispirati ai lavori [24, 23]. Diversamente da questi lavori però, sono stati considerati gli eventi come un concetto di alto livello formalmente definito in una ontologia degli eventi ed è stato modellato il rapporto tra una tale ontologia e il DSMS sottostante.

Infine, il linguaggio di query che è stato riportato nella sezione 1.1.4, principalmente attraverso esempi, trae ispirazione dal linguaggio TinyDB [24], a cui è stata aggiunta la clausola WINDOW.

1.2 L'ambito di ricerca

Considerato che la bozza di partenza di questo lavoro di tesi è quanto descritto nella sezione 1.1.3, nei successivi paragrafi 1.2.1 e 1.2.2 si procedono a dettagliare le scelte effettuate nella realizzazione degli obiettivi preposti e gli ulteriori obiettivi derivati da questi.

Si precisa che in questo lavoro di tesi non è stato richiesto il raggiungimento tutti gli obiettivi proposti in 1.1.3, ma sono stati concordati parte di questi, dato che tutto il progetto avrebbe una durata stimata di circa tra anni.

1.2.1 V2I - Vehicle to Infrastructure

Per quanto riguarda il V2I (Vehicle To Infrastucture), oltre a quello già detto nel documento [9] e riportato nella sezione 1.1.3, con particolare riferimento agli altri obiettivi descritti [2], ovvero:

- ridurre il carico sulle comunicazioni V2I utilizzando tecniche per il risparmio di dati
- indicare dopo un'attenta analisi (vedi capitolo 5) le migliori strategie *communication-saving* utilizzabili in pratica
- ridurre il carico sulle comunicazioni V2I utilizzando comunicazioni V2V

ciò che si è potuto dedurre sono stati gli ulteriori seguenti obiettivi:

- riuscire a implementare le tecniche descritte in modo da rendere il codice portabile sull'architettura reale senza troppi disagi
- cercare e/o creare nuove tecniche *communication-saving* eventualmente utilizzabili
- confrontare tra loro pregi e difetti delle diverse tecniche

[2] Elencati in ordine di importanza decrescente

Le tecniche

Così come indicato da [9] con riferimento a quanto riportato in 1.1.3, le tecniche in primo luogo sviluppabili, in ordine di importanza e fattibilità decrescenti, sono le seguenti:

- *simple sampling*, che invia i dati a intervalli di tempo/distanza percorsa regolari (ad esempio ogni minuto o ogni 2 km)
- *map-based sampling*, che permette di scegliere comunicazioni più intelligenti sulla base della posizione del veicolo sulla mappa
- *deterministic information-need*, dove ogni OBU riceve dinamicamente in broadcast una velocità v_b dal Centro di Controllo per ogni segmento di interesse e, per ogni segmento completato, trasmette la velocità media misurata v_m se e solo se $\mid v_b - v_m \mid$ supera una determinata soglia T [11]
- *probabilistic information-need*, dove ogni veicolo trasmette le informazioni con una data probabilità p, che può essere aggiornata dal Centro di Controllo al fine di garantire una determinata confidenza nel calcolo della velocità media [3]
- *linear regression* [15], sfruttando eventualmente le notevoli quantità di ridondanza nelle letture da un veicolo nel tempo

Considerando che non si può ancora disporre di un centro di controllo, la tecnica *probabilistic information-need* non è possibile svilupparla, mentre tramite alcune modifiche è eventualmente possibile implementare una o più versioni di *deterministic information-need* (vedi progetto (3) e implementazione (4, 4.2.2)).

Così come per le due tecniche precedenti, non avendo a disposizione le mappe che normalmente sarebbero disponibili sull'architettura dell'OBU (vedi Figura 1.4), per quanto riguarda il *map-based sampling* si procederà utilizzando quanto messo a disposizione da Vissim [40] (Vedi 2.1) e da Google con Google Geocoding API [13] (Vedi 2.5).

Per quanto riguarda invece *linear regression*, non dovrebbe essere necessario

creare modelli e kernel particolarmente complessi come in [15], soprattutto, se vediamo una strada come unione di tanti tratti, otteniamo all'incirca un gruppo di segmenti e possiamo quindi interpolarli con una semplice retta. Chiaramente utilizzando funzioni più complesse si potrebbero ottenere risultati migliori, ma bisogna anche tener conto del costo computazionale che un OBU può sopportare, molto inferiore a quello di un normale personal computer.

Sempre a differenza di [15] i dati su cui si deve lavorare non provengono da una rete di sensori, ma bensì dalla storia passata (che verrà definita *history*) del singolo veicolo, cioè tutti i dati compresi tra un intervallo di tempo passato e uno più recente. Chiaramente anche questo ha un impatto sulle performance dell'architettura reale dell'OBU, in quanto memorizzare dati richiede memoria.

Comunicazione

Per quanto riguarda la comunicazione, intesa come scambio di messaggi tra OBU e Centro di Controllo (V2I), dato che una stima plausibile della dimensione reale media di un messaggio servirebbe a stimare ulteriori componenti del sistema (ad esempio vedi 1.1.4), si è cercato di definire il contenuto di questi messaggi in modo, chiaramente univoco, ma anche il meno ridondante possibile in termini di Bytes. Questa codifica avvantaggia notevolmente una eventuale compressione a posteriore dei dati.

Compressione dati

La codifica implementata in 4.2.8 è stata studiata per permettere algoritmi di compressione e/o codifica in stile Huffmann dato che vi sono alcuni caratteri che hanno molta più frequenza di altri. Un breve studio sulla compressione dei messaggi ha portato ad evitare algoritmi come Lempel Ziv e similari (ZIP, RAR, etc.) per i singoli messaggi in quanto la dimensione degli headers risulta proibitiva (Vedi appendice 6.2). Nonostante questo non si nega nessuna possibilità di ulteriori compressioni a posteriori, sia a livello software che hardware, soprattuto se s'inviano cluster di messaggi.

1.2.2 V2V - Vehicle to Vehicle

Come per il V2I, con particolare riferimento alla sezione 1.2.1, anche per quanto riguarda il V2V (Vehicle To Vehicle), oltre a quello già detto nel documento [9]

e riportato nella sezione 1.1.3, sempre con particolare riferimento agli obiettivi descritti [3], ovvero:

- stimare la necessaria quantità percentuale di veicoli necessari per rendere possibile un efficiente utilizzo delle comunicazioni V2V
- utilizzare comunicazioni e strategie V2V per ridurre il carico sulle comunicazioni V2I

ciò che si è potuto dedurre sono stati gli ulteriori seguenti obiettivi:

- ottenere stime il più verosimili possibile alla realtà
- definire delle strategie e delle politiche di clustering tra veicoli (ed eventualmente tra cluster)

Si prosegue con la descrizione più dettagliata che ha portato ai passi successivi di questo lavoro, presentando le scelte effettuate in merito alle tecniche, alla comunicazione e alla compressione dati.

Tecniche

Come descritto in [9] e riportato nella sezione 1.1.3 di questo documento, con particolare riferimento a [7], le tecniche da adottare prevedono l'utilizzo di CH (Cluser Heads) e CM (Cluser Members).

I veicoli che montano un OBU e sono marcati come CH dirigono il gruppo, ovvero fungono da coordinatori e/o da punto di riferimento per le altre OBU ed eventualmente sono gli unici che comunicano le informazioni, sempre relative al gruppo, al Centro di Controllo (comunicazione V2I per V2V).

Numerosi esempi di politiche, protocolli di comunicazione e algoritmi per la formazione di questo tipo di gruppi, sono citati in letteratura (vedi ad esempio [7, 21]); in questo lavoro si proporrà un semplice algoritmo per l'aggregazione delle OBU, prendendo spunto da questi altri articoli.

[3] Elencati in ordine di importanza decrescente

Comunicazione

Per quanto riguarda la comunicazione, non si ci vuole soffermare troppo su questo aspetto dato che, considerando le performance dei nodi, che sono equipaggiati con un sistema WiFi 802.11, ovvero tempi di latenza compresi tra uno e tre millisecondi e velocità di trasmissione quasi sempre superiori ad un mega bit al secondo [43, 44], è necessario trasferire quantità di dati che richiedono performance inferiori a quelle date (vedi analisi sperimentale 5 e paragrafi relativi 5.3 5.4).

Compressione dati

Così come per il V2I (vedi sezione 1.2.1) le tecniche e gli algoritmi utilizzati sono i medesimi. L'unica nota differente potrebbe derivare dalla definizione di un nuovo formato d'invio, comunque già previsto dalla codifica implementata (vedi 4.2.8).

1.2.3 Note e considerazioni

Da questo quadro sono emerse le scelte progettuali visibili nel capitolo 3 e di conseguenza all'implementazione (capitolo 4). La scelta degli scenari, che devono essere appunto il più possibili reali, ha portato alle scelte effettuate in 5.1, mentre la parte di analisi, sia V2I (sezione 5.3) sia V2V (sezione 5.4), ha confermato e talvolta smentito ciò che la letteratura indicava, come descritto ulteriormente infine.

Nel capitolo successivo (2) si prosegue con la descrizione delle tecnologie studiate ed eventualmente utilizzate per lo sviluppo del progetto e la sua implementazione.

Capitolo 2

Le tecnologie utilizzate

Come già introdotto dai capitoli precedenti, tralasciando Java [18] il linguaggio di programmazione scelto, orientato agli oggetti, creato da James Gosling e altri ingegneri di Sun Microsystems, la cui piattaforma di programmazione è fondata sul linguaggio stesso, sulla *Java Virtual Machine* (o JVM) e sulle API Java, lo sviluppo di questo progetto ha richiesto l'utilizzo di alcune tecnologie che verranno dettagliate nei paragrafi successivi.

Nella sezione 2.1 è descritto Vissim [40] un simulatore di traffico professionale; nella sezione 2.2 Simjava [35] è riportato lo studio fatto per l'eventuale supporto che questo package avrebbe potuto fornire nello sviluppo del simulatore; nella sezione 2.3 JCoord [19] è presentato questo package utilizzato per la conversione di coordinate geografiche; nella sezione 2.4 viene riportato l'utilizzo fatto del parser DOM [8] all'interno del progetto, con particolare riferimento all'utilizzo di Google Geocoding API [13] (vedi 2.5 ultima sezione di questo capitolo).

2.1 Vissim

In questa sezione si descrive il simulatore di traffico Vissim [40], in primo luogo, nella sezione 2.1.1, con una presentazione presa da [31] [1], poi, nella sezione 2.1.2, si presenta l'esperienza diretta derivata dall'uso di questo software.

2.1.1 L'introduzione di TPS PTV

VISSIM [40] è un modello di microsimulazione dinamica della circolazione stradale che fa parte della linea di prodotti PTV Vision [31]. É il più potente strumento disponibile per la simulazione multimodale dei flussi di traffico, comprendendo varie categorie veicolari: auto, veicoli pesanti, bus, tram, treni, cicli, motocicli e pedoni.

Figura 2.1: Vissim screenshot - esempio 3D

La sua flessibile struttura di rete permette all'utente di modellizzare qualsiasi tipo di infrastruttura di trasporto e differenti comportamenti di guida dei conducenti.

Applicazioni tipiche di VISSIM

VISSIM può essere utilizzato per trovare soluzione a molteplici problemi di simulazione del traffico pubblico e privato. Le applicazioni più comuni includono:

- Studi di pianificazione
- Analisi e confronto di diversi scenari progettuali (nuove infrastrutture di trasporto o modifica degli schemi di circolazione esistenti)
- Analisi degli interventi in corrispondenza di nodi: incroci a raso regolati da segnaletica o semaforizzati, rotatorie, incroci a livelli sfalsati

[1] http://www.tpsitalia.it/software/ingegneria_traffico/vissim.php

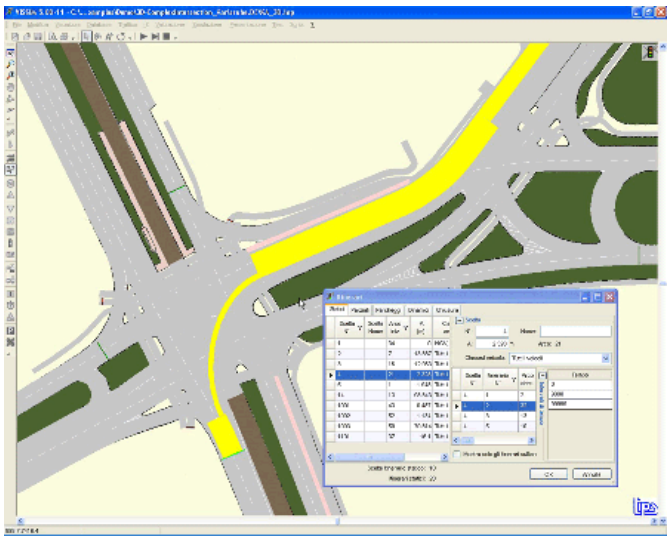

Figura 2.2: Vissim screenshot - Modellizzazione itinerari

- Valutazione ed ottimizzazione del traffico in reti con impianti semaforici attuati e coordinati
- Valutazione dell'impatto di nuovi sistemi di trasporto (LRT, Tram, ecc.) in reti stradali urbane
- Interventi di traffic calming
- Analisi degli attraversamenti ferroviari
- Valutazioni di impatto ambientale
- Analisi dei Sistemi di Trasporto Intelligente (ITS)
- Studi di traffico nelle aree landside e airside aeroportuali
- Simulazione pedonale per lo studio dei fenomeni legati all'affollamento in aree urbane e all'interno di strutture inclusi piani di evacuazione

Qual è la particolarità di VISSIM?

Il sistema ad interfaccia aperta garantisce compatibilità con software esterni. La possibilità di definire geometricamente archi e connessioni in modo dettagliato e

Figura 2.3: Vissim screenshot - Simulazione multimodale

l'elevata risoluzione dei movimenti veicolari (1/10s) consentono una rappresentazione realistica del comportamento di veicoli sulla rete. Presente sul mercato dal 1992, VISSIM [40] sta diventando lo standard di riferimento per i software di microsimulazione; un'intensa ricerca e un elevato numero di utenti in tutto il mondo lo collocano ai vertici fra i prodotti di questo tipo. Inoltre, la linea di prodotti PTV Vision [31] è stata la prima ad integrare la microsimulazione con la pianificazione strategica dei trasporti e la modellazione della domanda di trasporto.

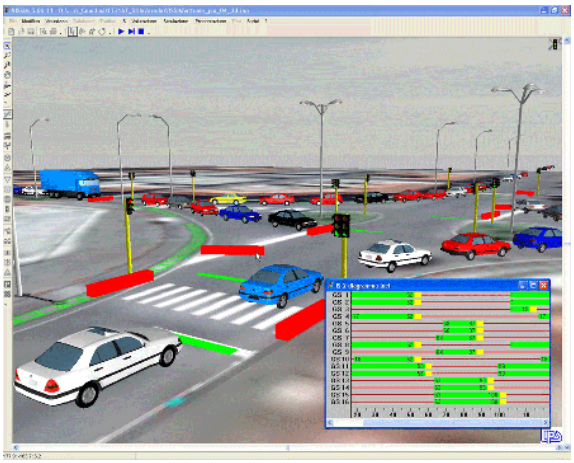

Figura 2.4: Vissim screenshot - Impianto semaforico

Rete di trasporto

VISSIM viene utilizzato per lo studio di reti di trasporto di qualsiasi dimensione, dalle singole intersezioni ad intere aree metropolitane. Consente la modellizzazione delle diverse tipologie di strada, dalle autostrade alle strade pedonali e piste ciclabili. Permette inoltre di modellizzare reti per semplici sistemi di trasporto definendo liberamente ulteriori caratteristiche geometriche e funzionali.

Volumi di traffico

VISSIM offre la possibilità di definire un numero illimitato di tipi di veicolo. Permette infatti di modellizzare: auto, veicoli pesanti, veicoli equipaggiati con sistemi di ricerca itinerario, bus, treni, cicli, motocicli, ed anche velivoli.

Figura 2.5: Vissim screenshot - Pannelli a messaggio variabile, priorità del trasporto pubblico, ingresso ad un parcheggio

Trasporto pubblico

VISSIM viene utilizzato da tempo come supporto alle analisi sul trasporto pubblico; ad esempio per lo studio di servizi di linea con autobus, servizi di linea metropolitana e terminal di scambio.

VISSIM per il trasporto pubblico permette di modellizzare itinerari, diversi tipi di veicolo, corse, fermate, diversi tipi di fermata e tempi di sosta alle fermate. L'utente ha la possibilità di scegliere tre modalità diverse per la definizione dei tempi di sosta alle fermate.

Controllo semaforico

VISSIM permette la modellizzazione di intersezioni regolate da segnali di precedenza, segnali di stop o da impianti semaforici. A differenza di altri software di

simulazione permette inoltre la modellizzazione di qualsiasi altro tipo di dispositivo di controllo. La regolazione semaforica del traffico può avvenire seguendo modalità diverse. In particolare possono essere definiti impianti semaforici a ciclo fisso e impianti semaforici attuati dal traffico.

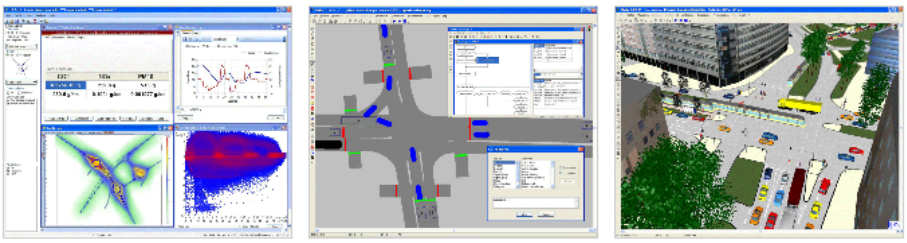

Figura 2.6: Vissim screenshot - Emissioni degli inquinati, controllo semaforico, modellazione 3D

Moduli Addizionali

- Assegnazione dinamica: attraverso l'utilizzo di matrici O/D, consente la realizzazione di una simulazione sulla rete di trasporto senza dover ricorrere all'implementazione manuale di tutti gli itinerari possibili all'interno della rete rappresentata.
- VAP (Vehicle Actuated Programming) e VisVAP (Visual VAP): il modulo VAP consente la modellizzazione di impianti semaforici attuati dal traffico.
- Modellazione 3D: questo pacchetto include il VISSIM 3D Modeller (V3DM), tool di modellazione 3D che consente la creazione di nuovi oggetti dinamici (es. veicoli) ed oggetti statici (es. edifici). Il V3DM è principalmente utilizzato per l'importazione in VISSIM di modelli 3D di veicoli realizzati in formato DWG o 3DS e aggiungere informazioni relative alla simulazione (es. posizione assi, luci di stop, indicatori di svolta etc.).
- Emissioni: sono disponibili diverse configurazioni di moduli per il calcolo delle emissioni inquinanti. Tali moduli differiscono in funzione delle componenti inquinanti calcolate (CO, CO2, NOX, HC, SO2, particolato, etc.) e del livello di risoluzione spazio-temporale.

- API: il pacchetto API rende disponibili tre interfacce di programmazione per l'utente, tutte in forma di dll: interfaccia di controllo semaforico; interfaccia modello di guida esterno; interfaccia modello di emissioni.

2.1.2 L'utilizzo effettuato

Considerando le possibilità di questo software, gli innumerevoli utilizzi possibili, nonché la particolare difficoltà nell'apprendere una così vasta gamma di *features* (almeno 1000 pagine di manuali in lingua inglese e/o tedesca) e soprattuto dato l'uso che si prefiggeva di fare, si ci è "limitati" ad utilizzare le mappe di dimostrazione presenti (vedi paragrafi 5.1.2 5.1.3 5.1.4) e a modificare quella fornita dall'Università di Bologna [39] (vedi sezione 5.1.1) che avrebbe altresì dovuto fornire una versione completa di tutto, veicoli e semafori compresi, cosa che purtroppo non è stata possibile (vedi figura 2.7).

Figura 2.7: Vissim screenshot - Mappa del centro di Bologna

2.2 SimJava

Simjava [35] simjava è un pacchetto di simulazione per Java che lavora per processi ed è basato su eventi a intervalli discreti, simile a Jade's Sim++, con servizi di animazione.

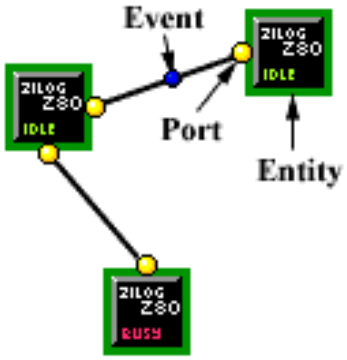

Figura 2.8: Simjava - A simulation layout

Una simulazione simjava è un insieme di entità ognuna eseguita nel proprio *thread*. Queste entità sono collegate tra loro da porte e possono comunicare tra loro mediante l'invio e la ricezione di "oggetti evento". Una classe del sistema centrale controlla tutti i *thread*, incrementa il tempo di simulazione, e smista gli eventi. Il progresso della simulazione è registrato attraverso dei messaggi appositi prodotti dalle entità ed è salvato in un file.

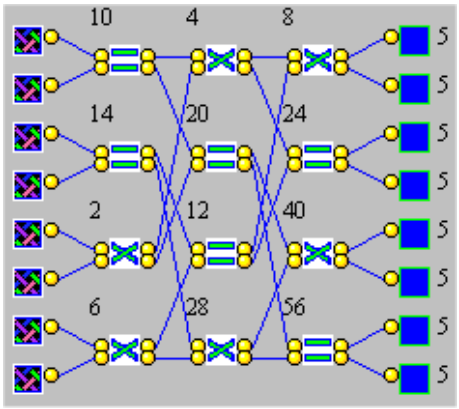

Figura 2.9: Simjava - A simple omega switching network

Purtroppo l'altro numero di OBU, che sarebbero divenuti le entità di simjava, avrebbe provocato numerosi problemi sia per l'alto numero di *thread* attivi (si parla di più di diecimila per il V2I e altrettanti per il V2V), sia per il numero di connessioni (più di diecimila per il V2I e più di cento milioni 100.000.000 per il V2V), mentre, da alcuni test personalmente effettuati, emerge che il limite massimo si aggira sulle cinquemila entità, circa un quarto del valore minimo richiesto.

Fatte queste considerazioni, pur avendo acquisito nozioni ed avendo testato questa tecnologia, essa è stata infine scartata.

2.3 JCoord

JCoord [19] è un package Java che implementa conversioni tra OSGB (Ordnance Survey of Great Britain), UTM (Universal Transverse Mercator) e latitudine/-longitudine.

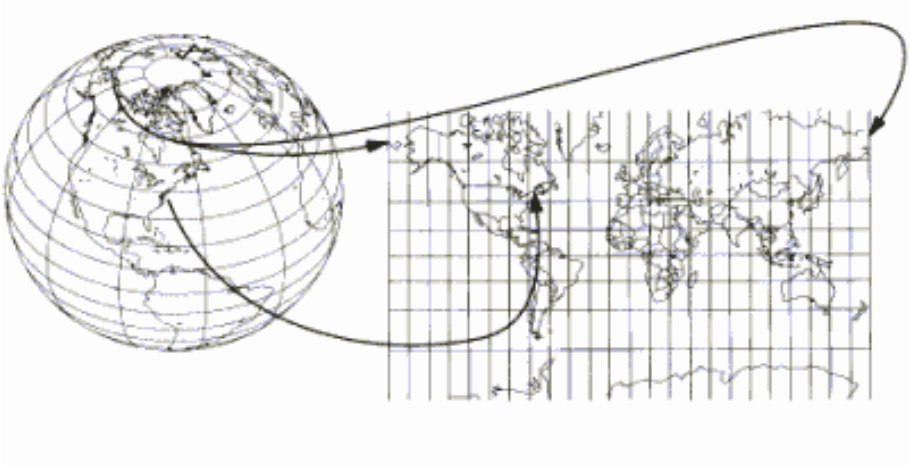

Figura 2.10: JCoord - conversione latitudine/longitudine in UTM

Questo package è stato utilizzato per risparmiare l'implementazione della conversione delle coordinate UTM in latitudine/longitudine, in quanto Vissim [40] utilizza le prime.

L'implementazione del calcolo della distanza tra due punti latitudine/longitudine è stata comunque effettuata (vedi sezione 4.2.7) anche se presente in JCoord, dato che, molto probabilmente, l'architettura dell'OBU non supporterà codice Java, quindi si sarà impossibilitati ad usare questo package.

2.4 DOM e Xerces

Il Document Object Model (DOM) [8], letteralmente modello a oggetti del documento, è una forma di rappresentazione dei documenti strutturati come modello orientato agli oggetti, è indipendente dalla piattaforma e dal linguaggio. I principali formati gestiti sono HTML, XHTML e XML. Gli aspetti di DOM, ovvero i suoi elementi "Elements", possono essere manipolati con la sintassi del linguaggio di programmazione in uso (nel nostro caso Java [18]).

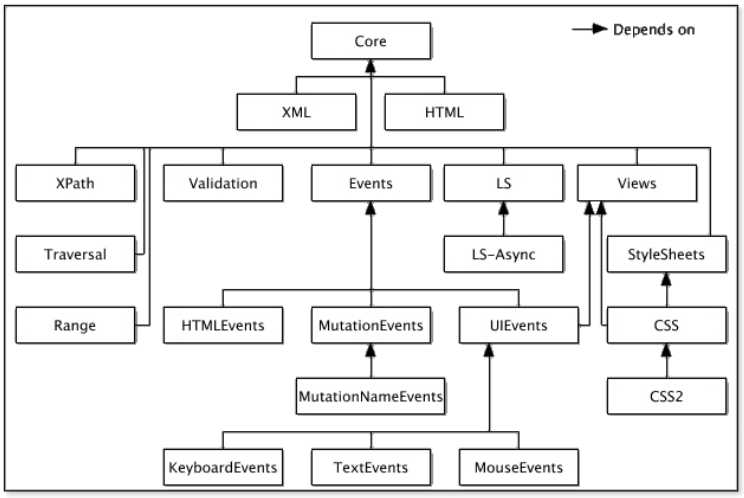

Figura 2.11: DOM e Xerces - architettura DOM (da W3C)

DOM è lo standard ufficiale del W3C [41] per la rappresentazione di documenti strutturati in maniera da essere neutrali sia per la lingua che per la piattaforma. DOM è inoltre la base per una vasta gamma delle interfacce di programmazione delle applicazioni; alcune di esse sono standardizzate dal W3C.

L'implementazione per Java del parser DOM è Xerces [45]. Il suo utilizzo in questo progetto ha riguardato l'estrapolazione di dati derivanti da Google Geocoding API (vedi sezione 2.5).

2.5 Google Geocoding API

Il *geocoding* è il processo di conversione da indirizzo (ad esempio "1600 Amphitheatre Parkway, Mountain View, CA") a coordinate geografiche (ad esempio latitudine 37.423021 e longitudine -122.083739), le quali possono essere usate per posizionarsi in una mappa. Il Google Geocoding API [13] fornisce servizi di *geocoding* tramite una richiesta HTTP; questo servizio permette anche di compiere l'operazione inversa, da coordinate ad indirizzo, conosciuta come *reverse geocoding*.

Figura 2.12: Google Geocoding API - Web services

L'output in XML, interpretato con Xerces e DOM (vedi sezione 2.4), e le notevoli informazioni aggiuntive, qui (*) successivamente presentate, hanno portato alla creazione di tecniche *Map Based* "intelligenti" (vedi sezione 4.2.2).

(*) Informazioni aggiuntive eventualmente disponibili come presentate in [13]:

- *street_address* indicates a precise street address.
- *route indicates* a named route (such as "US 101").
- *intersection* indicates a major intersection, usually of two major roads.

- *political* indicates a political entity. Usually, this type indicates a polygon of some civil administration.
- *country* indicates the national political entity, and is typically the highest order type returned by the Geocoder.
- *administrative_area_level_1* indicates a first-order civil entity below the country level. Within the United States, these administrative levels are states. Not all nations exhibit these administrative levels.
- *administrative_area_level_2* indicates a second-order civil entity below the country level. Within the United States, these administrative levels are counties. Not all nations exhibit these administrative levels.
- *administrative_area_level_3* indicates a third-order civil entity below the country level. This type indicates a minor civil division. Not all nations exhibit these administrative levels.
- *colloquial_area* indicates a commonly-used alternative name for the entity.
- *locality* indicates an incorporated city or town political entity.
- *sublocality* indicates an first-order civil entity below a locality
- *neighborhood* indicates a named neighborhood
- *premise* indicates a named location, usually a building or collection of buildings with a common name
- *subpremise* indicates a first-order entity below a named location, usually a singular building within a collection of buildings with a common name
- *postal_code* indicates a postal code as used to address postal mail within the country.
- *natural_feature* indicates a prominent natural feature.
- *airport* indicates an airport.
- *park* indicates a named park.

- *point_of_interest* indicates a named point of interest. Typically, these "POI's are prominent local entities that don't easily fit in another category such as "Empire State Building" or "Statue of Liberty".
- *post_box* indicates a specific postal box.
- *street_number* indicates the precise street number.
- *floor* indicates the floor of a building address.
- *room* indicates the room of a building address.

Come indicato in [9] e riportato nella sezione 1.1.3, la tecnica *Map Based* potrebbe essere interpretata in vari modi differenti. Il primo e più gettonato sarebbe quello per cui l'OBU invia informazioni quando cambia strada, ovvero, per quanto riguarda i dati forniti da Google Geocoding API [13], quando cambia il parametro *street_address*. Un altro metodo potrebbe riguardare il raggiungimento di una posizione segnata come *intersection*, etc., anche dipendentemente dalle richieste in corso.

Purtroppo il limite di questa tecnologia, oltre l'eventuale intoppo dato dalla connessione ad internet, che difficilmente supera il mega bit per secondo e ritarda notevolmente i tempi di simulazione, riguarda il limite nel numero di connessioni (e quindi richieste) verso il servizio, fissato in 2500 nell'arco di 24 ore [13]. Utilizzando una simulazione apposita, anche se brevissima e con pochissime OBU (vedi riferimenti in 5.1), ad esempio con durata di sessanta secondi, cento OBU e con campionamento a mezzo secondo, otterremmo circa dodicimila richieste, circa cinque volte tanto il limite imposto.

Queste limitazioni hanno portato all'estrapolazione di ulteriori dati dal simulatore di traffico (vedi 2.1) che hanno permesso l'implementazione (vedi 4.2.2) di tecniche simili a quelle che utilizzavano il servizio di *geocoding*.

Da notare che, nell'architettura reale dell'OBU (vedi figura 1.4), le mappe con le relative informazioni, probabilmente molto simili a quelle fornite da Google Geocoding API [13], sono disponibili senza le limitazioni viste, quindi i principi applicati alle tecniche sviluppate saranno perfettamente validi.

Note e considerazioni sul caso di studio

Dal capitolo 1 sono emersi gli obiettivi principali che hanno guidato questo lavoro di tesi e che hanno permesso di svolgerlo con fluidità nel suo complesso, mentre dal capitolo 2 si sono potute definire le tecnologie che hanno portato alla costruzione dell'ambiente utilizzato.

Avendo così appreso il problema nella sua interezza ed avendo acquisito le necessarie conoscenze per risolverlo, si prosegue con lo sviluppo pratico dell'ambiente, in particolare con il progetto del software (capitolo 3), tramite alcuni aspetti SRS [37] e con il linguaggio di modellazione UML [38]. Nel capitolo 4 s'implementa ciò che è stato documentato e descritto nel capitolo 3, mentre nel capitolo 5 si presentano i risultati ottenuti dall'analisi sperimentale, a supporto o meno delle teorie espresse in precedenza. Infine si traggono le conclusioni generali e si definiscono i possibili aspetti futuri legati al lavoro svolto.

Parte II

Progetto, implementazione e analisi sperimentale

Capitolo 3

Analisi e progetto del software

In ingegneria del software, la progettazione, detta anche progetto o disegno, è una fase del ciclo di vita del software. Sulla base della specifica dei requisiti prodotta dall'analisi, il progetto definisce come tali requisiti saranno soddisfatti, entrando nel merito della struttura che dovrà essere data al sistema software che deve essere realizzato.

In questa fase si utilizzeranno alcune delle specifiche fornite dal SRS [37] per l'analisi e il linguaggio di modellazione UML [38] per il progetto.

Non essendo il software[1] da produrre destinato ad un pubblico, ma dovendo simulare il comportamento di un'architettura reale in movimento, saranno tralasciati alcuni scenari comunemente descritti in questa fase di progetto e ne saranno riportati altri comunemente non descritti.

[1] Da ora in poi definito come "simulatore di OBU" o "simulatore"

3.1 L'analisi dei requisiti

Il primo passo dell'analisi è quello dunque di confrontarsi con l'architettura reale dell'OBU visibile in figura 1.4 ed enfatizzata a livello di collegamenti in figura 3.1.

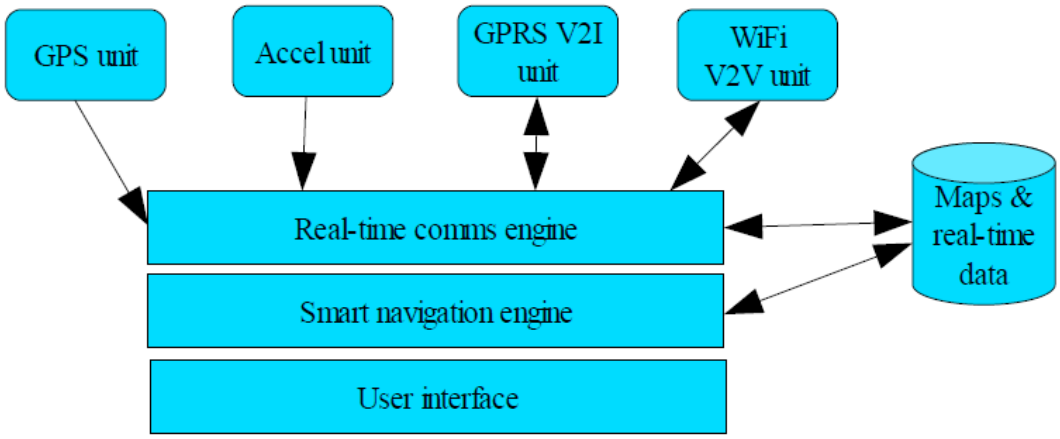

Figura 3.1: Architettura di un OBU

Questa però deve anche tener conto delle reali esigenze della simulazione di seguito elencate:

- Il simulatore deve leggere i dati dai file generati dalla simulazione del traffico
- Il simulatore deve analizzare i dati ed estrarre le informazioni relative ad ogni veicolo.
- Il simulatore, in base a parametri configurabili (Es. una percentuale di tutti i veicoli), deve decidere quante OBU devono essere istanziate.
- Il simulatore deve aggiornare, utilizzando i dati della simulazione del traffico, le singole OBU fino a che la simulazione non termina
- Le OBU devono comunicare, eventualmente, tra loro e con la centrale, con diverse politiche
- I risultati ottenuti dalle simulazioni devono essere presentati in forma intellegibile

Da queste specifiche è stato possibile definire il primo modello del software di simulazione, riportato in figura 3.2.

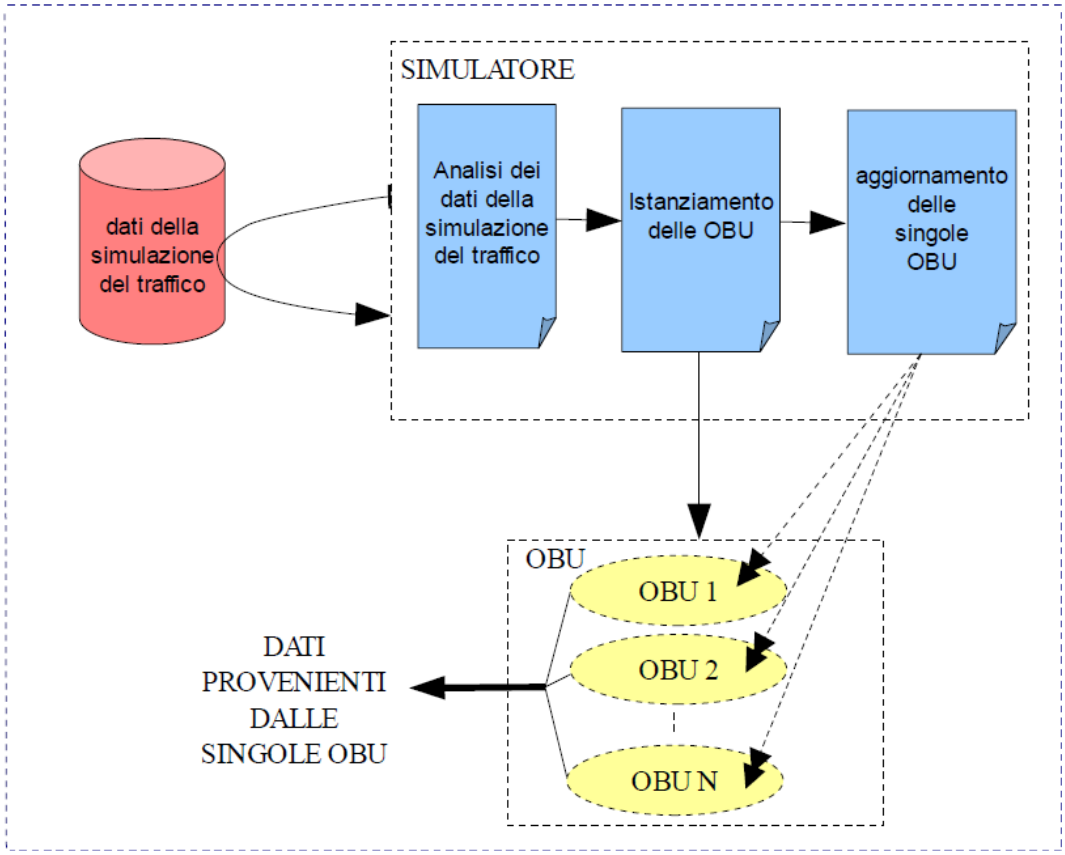

Figura 3.2: Bozza di architettura del simulatore

Il vincolo di poter riutilizzare il codice scritto all'interno dell'architettura reale decade per tutto ciò che non riguarda l'architettura stessa dell'OBU, infatti, gli "ambienti" di simulazione devono essere nettamente separati.

Non si richiedono particolari vincoli prestazionali, al di fuori del poter eseguire una simulazione, anche con quantità di dati intorno ai cinquecento mega bytes, in tempi non proibitivi (qualche ora), su architetture (PC) moderne.

Come si può notare, non vi sono particolari costrizioni da parte dei commit-

tenti, soprattuto perché l'interesse riguarda il fine, ovvero i risultati, e non il mezzo con cui si ottiene, ovvero il simulatore.

Alcuni dei vincoli, infatti, sono stati qui aggiunti per permettere ad un futuro utilizzatore e/o sviluppatore sia di comprendere il funzionamento del software che dare la possibilità di estenderne i contenuti senza particolari problemi.

A tal fine si richiede uno stile di scrittura del codice il più possibile conforme a quanto riportato nel libro "The Java Language Specification" [17], che dovrà altresì essere documentato.

Al fine di non divulgarsi inutilmente su particolari statistici e pratici insignificanti, si assume che la comunicazione GPRS sia sempre fattibile e il numero di veicoli della simulazione sia pari al numero di OBU (100%) se non espressamente specificato. Quest'ultimo parametro deve altresì essere modificabile in fase di esecuzione, in modo da permettere ad un futuro utilizzatore di selezionare le percentuali che riterrà più realistiche nella simulazione di traffico fornite.

3.2 Il progetto

Partendo dai requisiti della sezione 3.1, nella sezione 3.2.1 si presenta il caso d'uso, mentre nella sezione successivo 3.2.2 si presenta l'*activity diagram* come da specifiche UML [38].

Considerando che la progettazione del software è un processo ciclico, ovvero il software subisce continue evoluzioni che derivano da nuovi requisiti pratici, nella sezione 3.2.3 si presenta la prima versione sviluppata con i *class diagram* del linguaggio UML [38] seguita, nella sezione 3.2.4, dalla versione definitiva sviluppata nel medesimo modo.

3.2.1 Il caso d'uso

Dati i requisiti visibili alla sezione 3.1, essendo il software da sviluppare dedicato solo ed eventualmente ad utenti esperti, l'unico caso d'uso presentabile è quello visibile in figura 3.3.

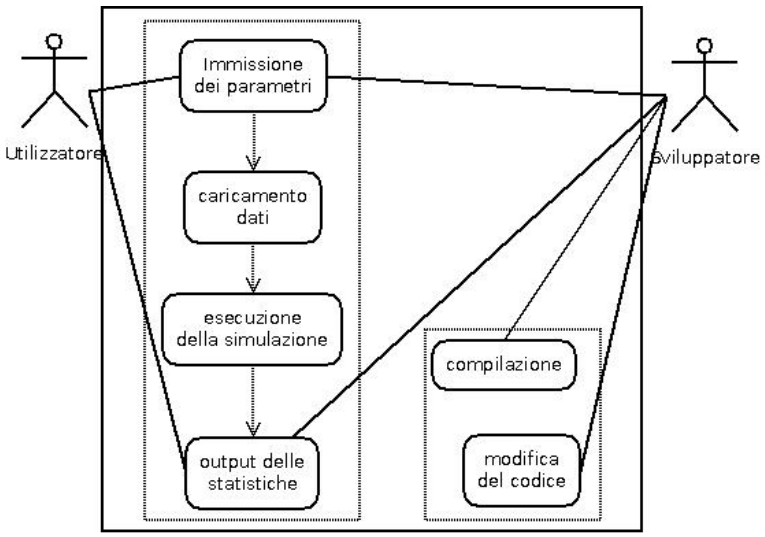

Figura 3.3: Use Case diagram

3.2.2 Il diagramma delle attività

In questa sezione, in figura 3.4 si presenta l'*activity diagram* che descrive il comportamento generale del software, che pur essendo stato concepito per permettere l'esecuzione a *threads* non li implementa in quanto tutto può essere svolto in modo seriale.

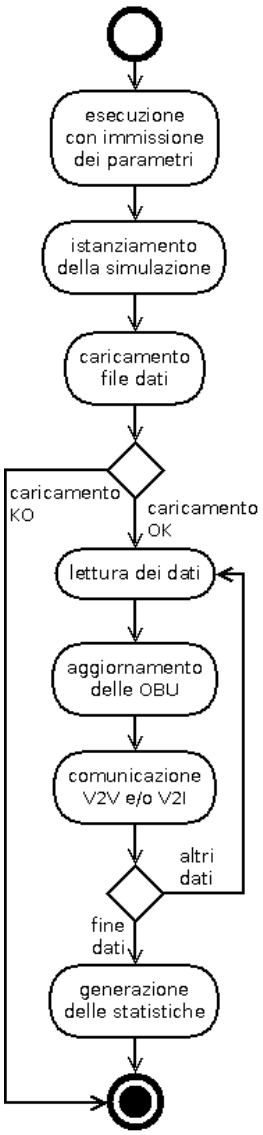

Figura 3.4: Activity Diagram

3.2.3 Il prototipo

Dallo schema generale dell'architettura del simulatore (vedi figura 3.2) e dallo schema dell'architettura dell'OBU (vedi figura 3.1) è possibile abbozzare una prima stesura dei *class diagram* [38] che comporranno man mano il simulatore. La modellazione ha inizio con il diagramma delle classi dell'architettura dell'OBU (figura 3.5).

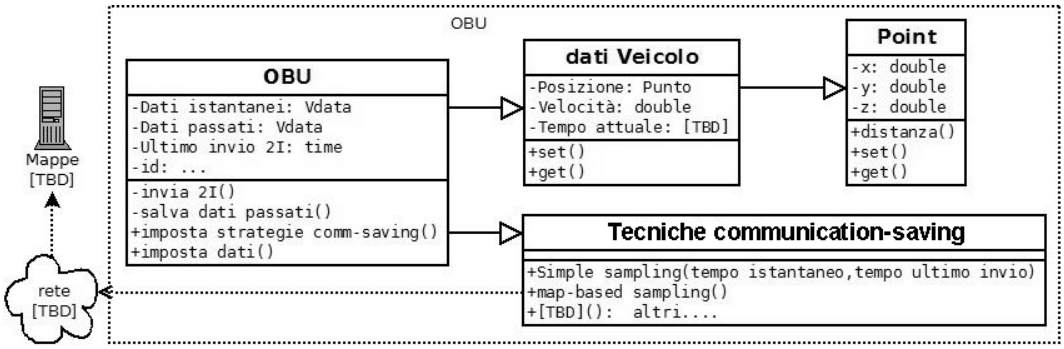

Figura 3.5: Class diagram - bozza architettura OBU

Il package OBU è composto principalmente da quattro classi:

1. la classe OBU che simula l'OBU reale
2. la struttura dati "dati Veicolo" (o "Vdata") che permette una facile estensione di eventuali altri dati, oltre alla pratica maneggevolezza di quelli già definiti (Es. Point)
3. "Point" che consente le operazioni tra punti nello spazio euclideo a 1, 2 e 3 dimensioni (Es. latitudine x, longitudine y, altitudine z)
4. "Tecniche communication-saving" che permettono all'OBU di sapere se, quando e come inviare i dati. Queste tecniche saranno ulteriormente descritte e sviluppate in questo documento (vedi sezione 4.2.2).

Non si esclude l'eventuale aggiunta di altre classi o la modifica di quelle appena definite in corso d'opera, compatibilmente con quanto descritto, per fare fronte ad eventuali problemi pratici. Tali modifiche saranno nuovamente documentate e descritte tramite ulteriori *class diagram*.

Per eseguire la simulazione è necessario definire anche la parte che si occupa del caricamento dei dati e della gestione delle OBU. Questo modulo, chiamato "OBUsim", è riportato nel diagramma delle classi in figura 3.6 insieme alla restante architettura del simulatore. In questo scenario è previsto anche l'eventuale ausilio del package SimJava [35].

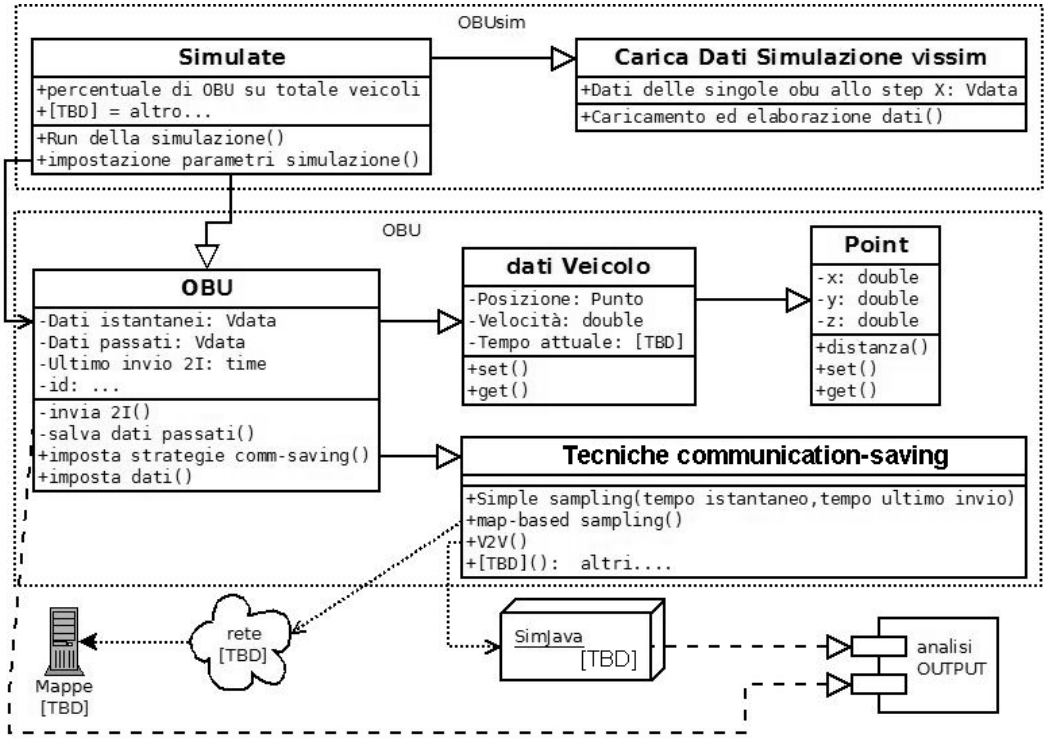

Figura 3.6: Class diagram - bozza architettura generale del simulatore

Lo scenario di simulazione "OBUsim" è composto principalmente da due parti:

1. la classe "Carica dati simulazione vissim" che si occupa del caricamento dei dati prodotti dalla simulazione nelle strutture dati "dati Veicolo"

2. la classe Main che permette di eseguire la simulazione dopo aver impostato i parametri richiesti.

I dati prodotti dall'OBU simulata verranno salvati per una successiva elaborazione (grafici e altri strumenti) così come per i dati prodotti dalle tecniche per V2V, eventualmente supportate dal package Simjava [35] ("analisi OUTPUT").

Anche in questo caso non si esclude l'eventuale aggiunta di altre classi o la modifica di quelle appena definite in corso d'opera, compatibilmente con quanto descritto, per fare fronte ad eventuali problemi pratici. Tali modifiche saranno nuovamente documentate e descritte tramite ulteriori *class diagram*.

3.2.4 La versione definitiva

La versione definitiva del progetto, mostrata nel *class diagram* in figura 3.7, dimostra la complessità raggiunta dal simulatore, al quale sono stati aggiunti e modificati classi, package, etc. in corso d'opera. La versione presentata è l'ultima stesura ed è stata realizzata prima della versione definitiva del software.

Come visibile in diagramma, la struttura di massima è stata preservata mentre le restanti parti sono state ampliate e modificate; l'utilizzo del package SimJava [35] è stato scartato come motivato nella sezione 2.2.

In dettaglio, al package OBU sono state aggiunte le classi "StraightLine2D", "GPS2Dconverter", "Sdata" e sono state modificate tutte le restanti ad eccezione di "Point".

Il package OBUsim è passato da due a otto classi, modificando anche le due già presenti. La classe "Main" è stata separata dalla classe "Simulate", il gruppo formato da "GetSimData" e "SimData" deriva dal caricamento dei dati della simulazione, in particolare "SimData" è visto come oggetto che contiene tutti i parametri caricati. Le classi "V2Vsim" e "I" hanno permesso rispettivamente la simulazione V2V, che necessita di dati condivisi, e la simulazione V2I che necessita della centrale. La classe "DecompressData" è a supporto della classe "I", mentre "GenUtil" contiene funzioni di utilità generiche.

I due nuovi package introdotti, "MapQuest"e "Test", sono rispettivamente di ausilio alla classe "Techniques"e alle classi "V2Vsim"e"I".

"MapQuest" s'interfaccia a Google Geocoding API [13], servizio descritto nella sezione 2.5, tramite protocollo HTTP, mentre la classe "SureStrArray"contiene i dati ricevuti nella sua apposita struttura.

Il package "Test"contiene classi per il testing dell'applicazione e la classe "Debug"che oltre a fornire strumenti per il salvataggio dei dati, come richiesto nella sezione 3.1, contiene anche le direttive per il *tracing* e il *debug* del programma stesso.

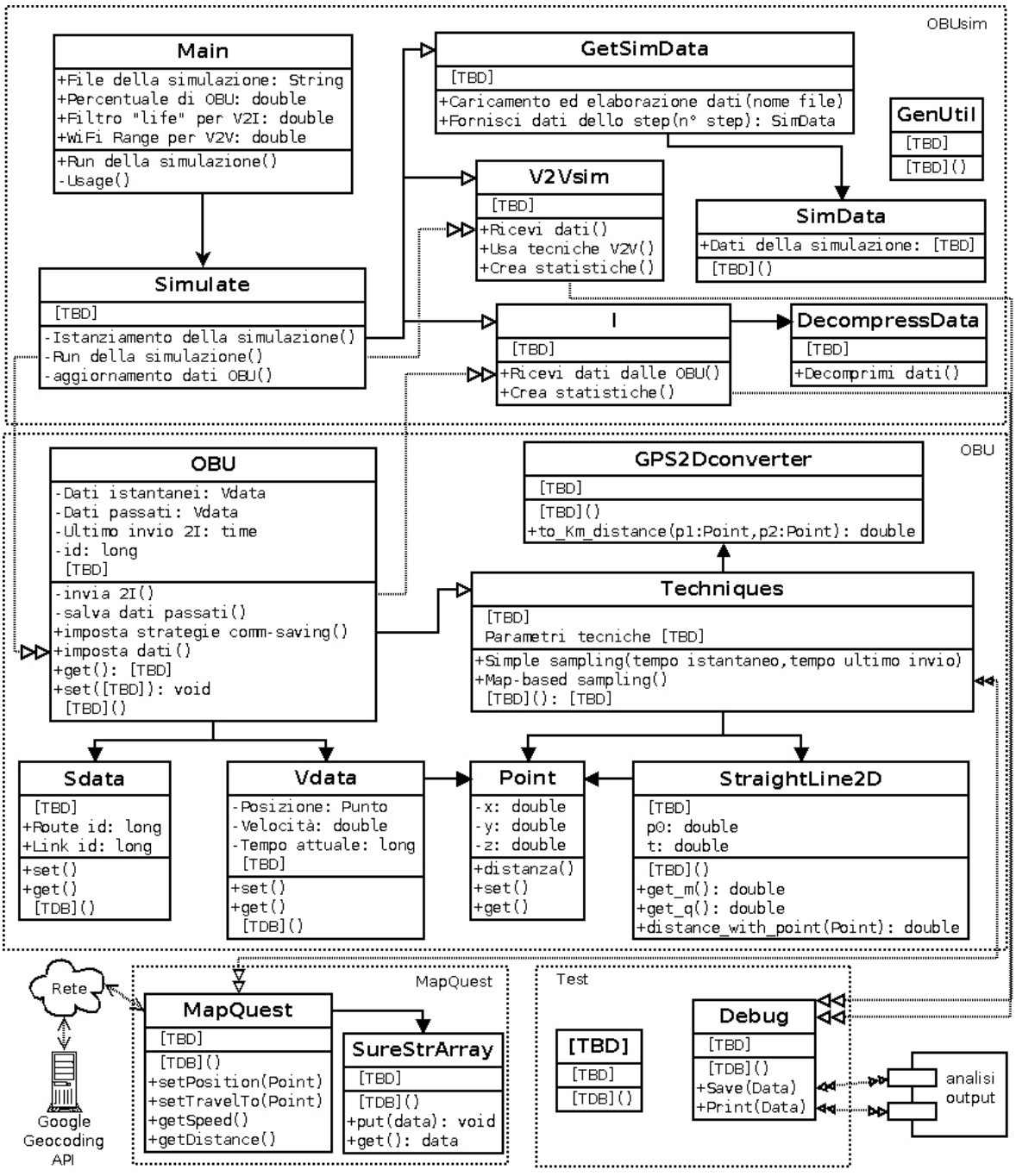

Figura 3.7: Class diagram - architettura generale del simulatore

Capitolo 4

L'implementazione

Come già accennato in precedenza, la scelta del linguaggio di programmazione per lo sviluppo del simulatore di OBU è ricaduta su Java [18], sia per motivi legati alla portabilità su diversi sistemi operativi, che per la programmazione ad alto livello, ma soprattuto per la somiglianza con il linguaggio C, il quale sarà probabilmente il linguaggio supportato dall'architettura reale dell'OBU.

Con questo non si vuole assolutamente dire che il codice potrà essere preso e portato direttamente, ma le modifiche da apportare saranno comunque minime e legate al solo package "OBU" che, appunto, sarà l'unica parte di codice da esportare.

Nei prossimi paragrafi saranno presentati i diagrammi UML [38] dei package e delle classi che rispecchiano fedelmente la struttura del codice.

Il simulatore è sviluppato su quattro package descritti nei paragrafi relativi, "OBUsim", sezione 4.1, che dirige la simulazione e definisce gli oggetti con cui interagiscono le OBU, "OBU", sezione 4.2, che implementa la struttura definita dell'OBU e le relative tecniche *communication-saving*, "Maps", sezione 4.3, che fornisce l'interfacciamento con Google Geocoding API [13], e, infine, "Test", sezione 4.4, che fornisce gli strumenti di I/O per le statistiche, il tracing e il debug, oltre a contenere i test di tutte le classi e le funzioni del simulatore.

4.1 Il package OBUsim

Questo package, che contiene al suo interno la classe Main (sezione 4.1.1), ha come scopo l'esecuzione della simulazione tramite la classe "Simulate" (sezione 4.1.2), fornendo tutti gli "oggetti" e i "dati" necessari al suo compimento.

Per oggetti s'intende l'infrastruttura o centro di controllo, simulata dalla classe "I" (sezione 4.1.7), le metodologie per la gestione del V2V, simulate dalla classe "V2Vsim" (sezione 4.1.8), la decompressione dei dati, effettuata dalla classe "DecompressData" (sezione 4.1.6), e, infine, altre funzioni di utilità generiche, implementate nella classe "GenUtil" (sezione 4.1.5).

Per dati necessari s'intendono invece le strutture dati, implementato dalla classe "SimData" (sezione 4.1.4) e il processo di acquisizione, implementato dalla classe "GetSimData" (sezione 4.1.3).

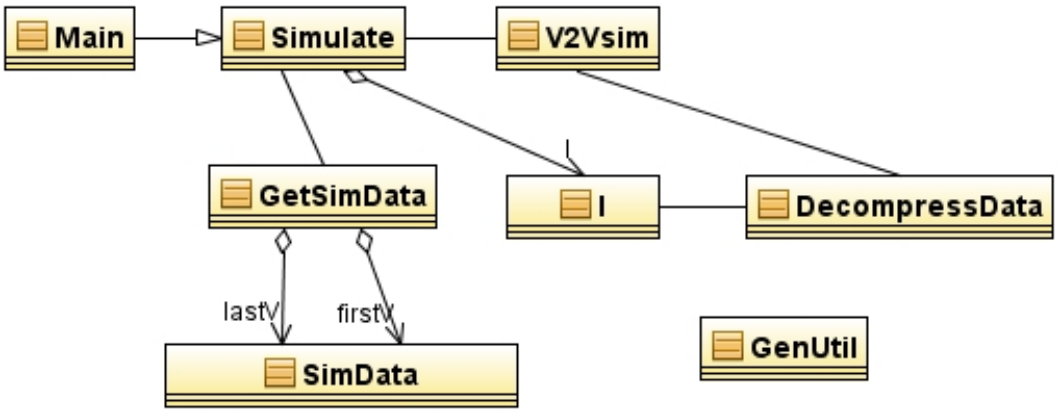

Figura 4.1: Class diagram definitivo - Package OBUsim

4.1.1 La classe Main

L'unico scopo di questa classe è eseguire il programma tramite la funzione "main" ed eventualmente avvisare l'utente nel qual caso immettesse dei parametri errati tramite la funzione "printusage".

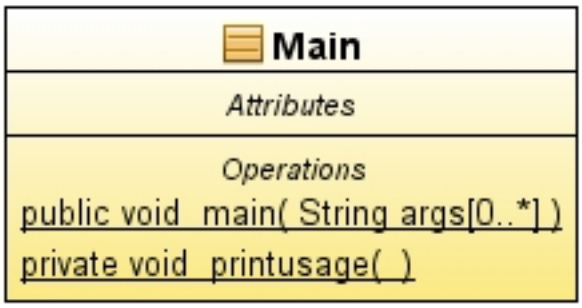

Figura 4.2: Class diagram definitivo - Classe Main

I parametri richiesti per eseguire la simulazione sono di due tipi, indispensabili e opzionali.

Il nome del file che contiene i dati della simulazione del traffico appartiene al primo tipo, mentre la percentuale di veicoli che montano un OBU, il filtro per escludere le OBU con tempo di vita inferiore a tale valore e il raggio di copertura dell'antenna WiFi, appartengono al secondo tipo. Nel qual caso i valori dei parametri opzionali non venissero inseriti o non fossero corretti, il programma utilizzerebbe quelli di *default*.

Per avviare la simulazione la funzione "main" istanzia una classe "Simulate" (vedi sezione 4.1.2) con i parametri di cui sopra.

4.1.2 La classe Simulate

Ricevendo i parametri necessari dalla classe "main", questa classe può procedere all'inizio della simulazione tramite la funzione "startSimulation"; altresì questa classe si occupa dell'istanziamento delle OBU e della relativa infrastruttura "I" che viene resa statica e pubblica.

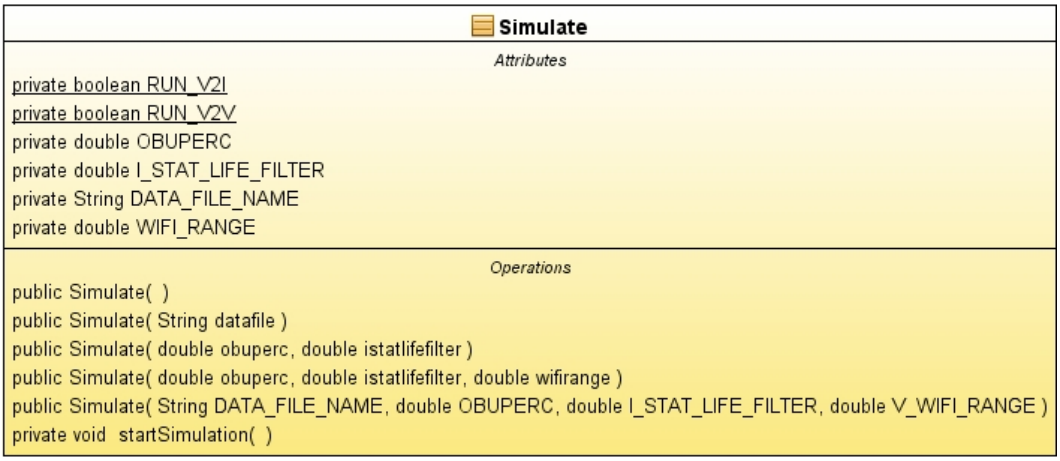

Figura 4.3: Class diagram definitivo - Classe Simulate

Il caricamento dei dati è affidato alla classe "GetSimData" (vedi sezione 4.1.4), mentre l'aggiornamento avviene sempre qui dato che gli "oggetti" "OBU" e "V2Vsim" sono creati in questo punto.

Se un futuro sviluppatore volesse modificare il comportamento delle simulazioni dovrebbe intervenire a questo livello, ad esempio impostando le tecniche *communication-saving* di ogni OBU appena dopo la creazione di queste, oppure anche a *runtime*, sempre per mezzo delle funzioni che la classe "OBU" mette a disposizione (vedi sezione 4.2.1).

4.1.3 La classe GetSimData

Come intuibile dal nome, lo scopo di questa classe è quello di fornire funzioni per il caricamento dei dati derivanti dalla simulazione del traffico.

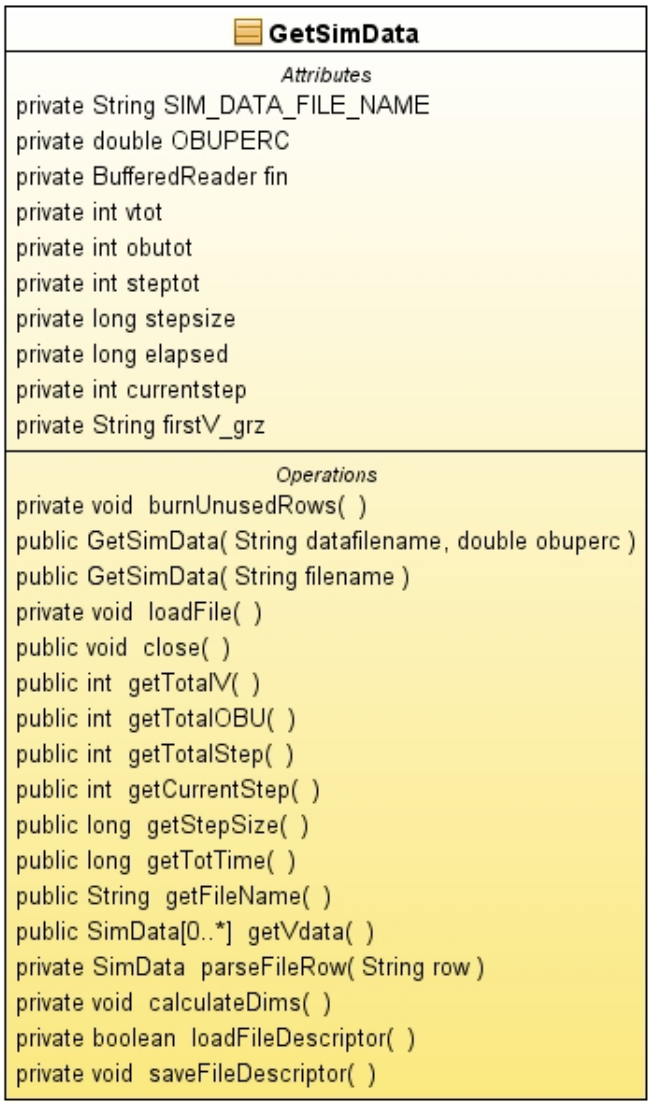

Figura 4.4: Class diagram definitivo - Classe GetSimData

Il costruttore accetta in input il nome del file ed eventualmente la percentuale di OBU sul totale di veicoli, nel qual caso questo parametro mancasse o fosse errato si assume che tutti i veicoli ne abbiano una (100%).

Tramite le funzioni pubbliche di tipo *get*, come "getTotalOBU", "getTotalV", etc., si possono reperire i dati richiesti.

Il processo di acquisizione, all'istanziamento della classe, non avviene in un unico passo, ma, in linea di massima, secondo l'iter seguente:

1. il costruttore, dopo aver verificato i parametri, richiama la funzione "loadFile"
2. la funzione "loadFile" richiama la funzione "loadFileDescriptor"
3. la funzione "loadFileDescriptor" controlla se c'è già un file descrittore valido, se c'è ed è errato lo rinomina, se c'è ed è valido imposta i parametri
4. il controllo ritorna alla funzione "loadFile"; se i parametri non sono stati impostati scorre tutto il file, li imposta e richiama "saveFileDescriptor", altrimenti, se sono stati impostati correttamente, salta il passo successivo
5. la funzione "saveFileDescriptor" salva il nuovo descrittore del file
6. la funzione "loadFile" apre il file con i dati della simulazione che ora può essere letto passo per passo, ovvero possono essere lette tutte le informazioni in un dato istante temporale che dipendono esclusivamente dal campionamento della simulazione, tramite la funzione "getVdata".

Le motivazioni che hanno spinto al caricamento per passi (o *step*) sono sostanzialmente due: la prima è l'eventuale alto consumo di memoria RAM necessario a caricare un intero file di dati della simulazione, che può arrivare e tranquillamente superare i cinquecento mega byte; la seconda riguarda la possibilità di interrompere la simulazione allo *step* voluto, ad esempio immettendo un nuovo parametro "durata", poi chiudendo il file tramite la funzione "close".

L'utilizzo della funzione "close" non è necessario nel qual caso si leggessero tutti i dati disponibili, ovvero si richiamasse la funzione "getVdata" per tutti gli *step* definiti e ottenibili dalla funzione "getTotalStep".

4.1.4 La classe SimData

Questa classe è stata definita al solo scopo di fornire una struttura dati che se fosse mancata avrebbe reso illeggibile e più complicato il codice delle classi "Simulate" (vedi sezione 4.1.2) e "GetSimData" (vedi sezione 4.1.3).

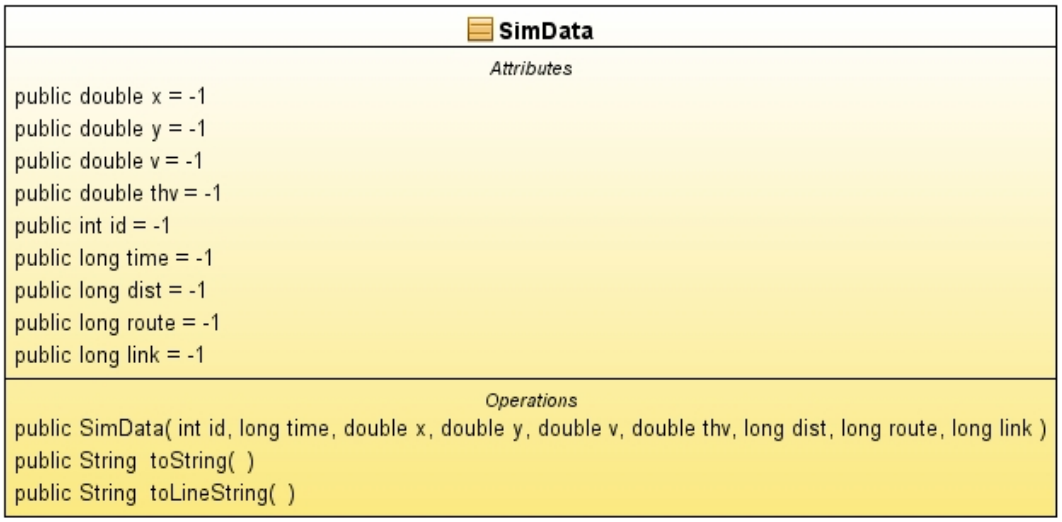

Figura 4.5: Class diagram definitivo - Classe SimData

Tralasciando le funzioni di tipo *toString*, questa classe rispecchia una semplice struttura dati *struct* del linguaggio C.

struct SimData{
double x,
double y,
...
};

4.1.5 La classe GenUtil

Questa classe contiene alcune funzioni di utilità generale. Tutte le funzioni, ad eccezione "hash_sha" che è utilizzata solo all'interno del package corrente (vedi sezione 4.1), sono richiamate più volte all'interno di tutti gli altri package.

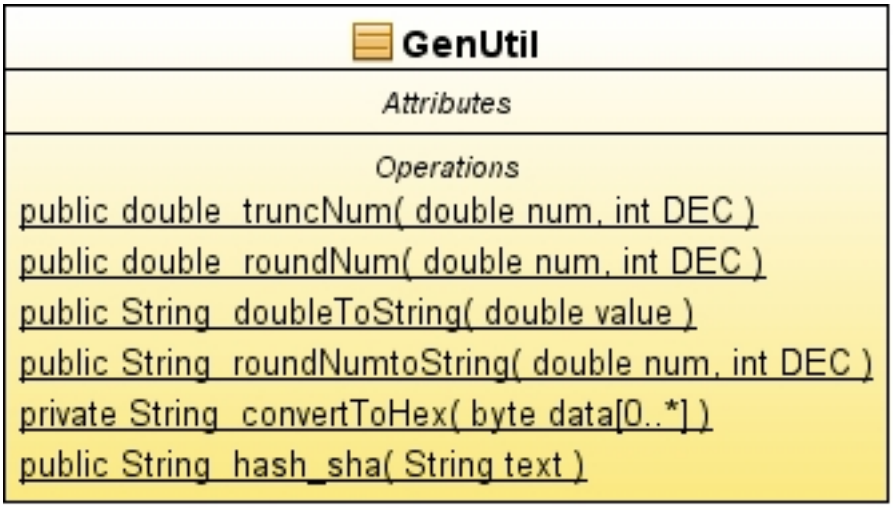

Figura 4.6: Class diagram definitivo - Classe GenUtil

Da considerare che, anche se queste funzioni sono utilizzate dal package OBU, al momento della codifica in altri linguaggi, esse non diverranno più necessarie in quanto diverranno disponibili altri formalismi del linguaggio, ad esempio in C per arrotondare un numero basterà utilizzare le funzioni *round* dell'*header math.h*.

4.1.6 La classe DecompressData

Questa classe, la cui unica funzione accessibile pubblicamente degna di note è "decompress", si occupa, appunto, tramite tal funzione, di decomprimere messaggi provenienti dalle OBU e compressi secondo i criteri di cui alla sezione 4.2.8.

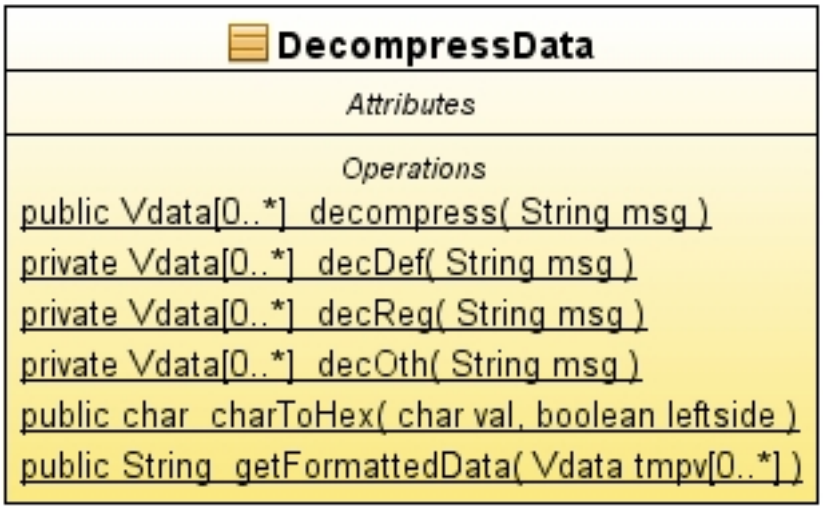

Figura 4.7: Class diagram definitivo - Classe DecompressData

Si precisa che l'implementazione delle funzioni di questa classe, o l'equivalente, dovrà essere presente nel centro di controllo nel momento in cui si adottasse la codifica creata.

4.1.7 La classe I

La classe "I", dove "I" sta per *Infrastructure*, ovvero il centro di controllo (vedi sezione 1.1.2), si occupa della simulazione di quest'ultimo per quanto riguarda l'invio di dati da parte delle OBU, elaborando anche le statistiche che saranno utilizzate per l'analisi V2I della sezione 5.3.

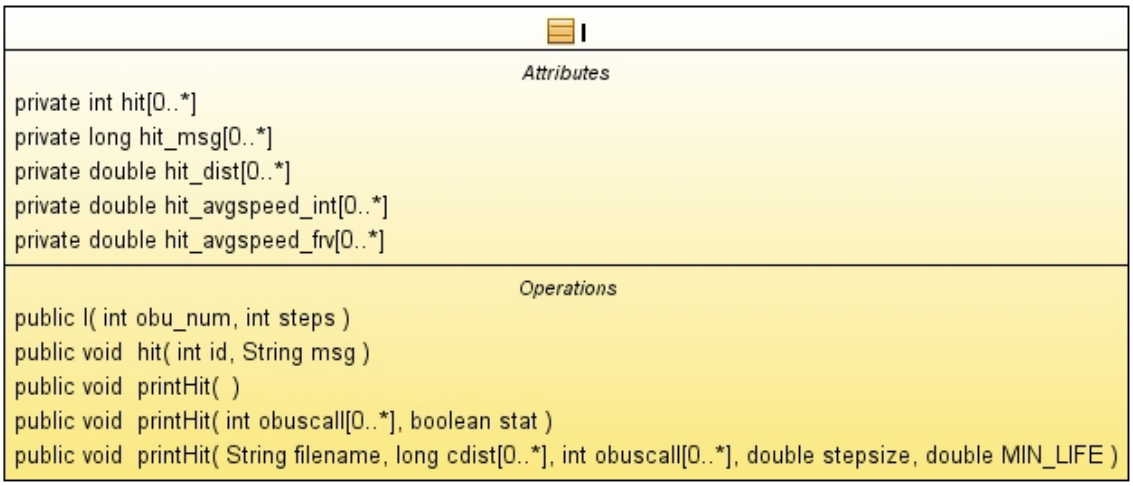

Figura 4.8: Class diagram definitivo - Classe I

Il costruttore richiede come parametri il numero di OBU e il numero di step della simulazione per allocare le strutture dati che conterranno i messaggi inviati dalle OBU tramite la funzione "hit".

4.1.8 La classe V2Vsim

Come nel caso della classe "I" (vedi sezione 4.1.7), questa classe simula l'iterazione V2V tra le varie OBU elaborando anche le statistiche che saranno utilizzate per l'analisi V2V della sezione 5.4. A differenza di "I" però, i dati relativi allo step devono essere aggiornati come in "OBU" (vedi sezione 4.2.1) per mezzo della funzione "update".

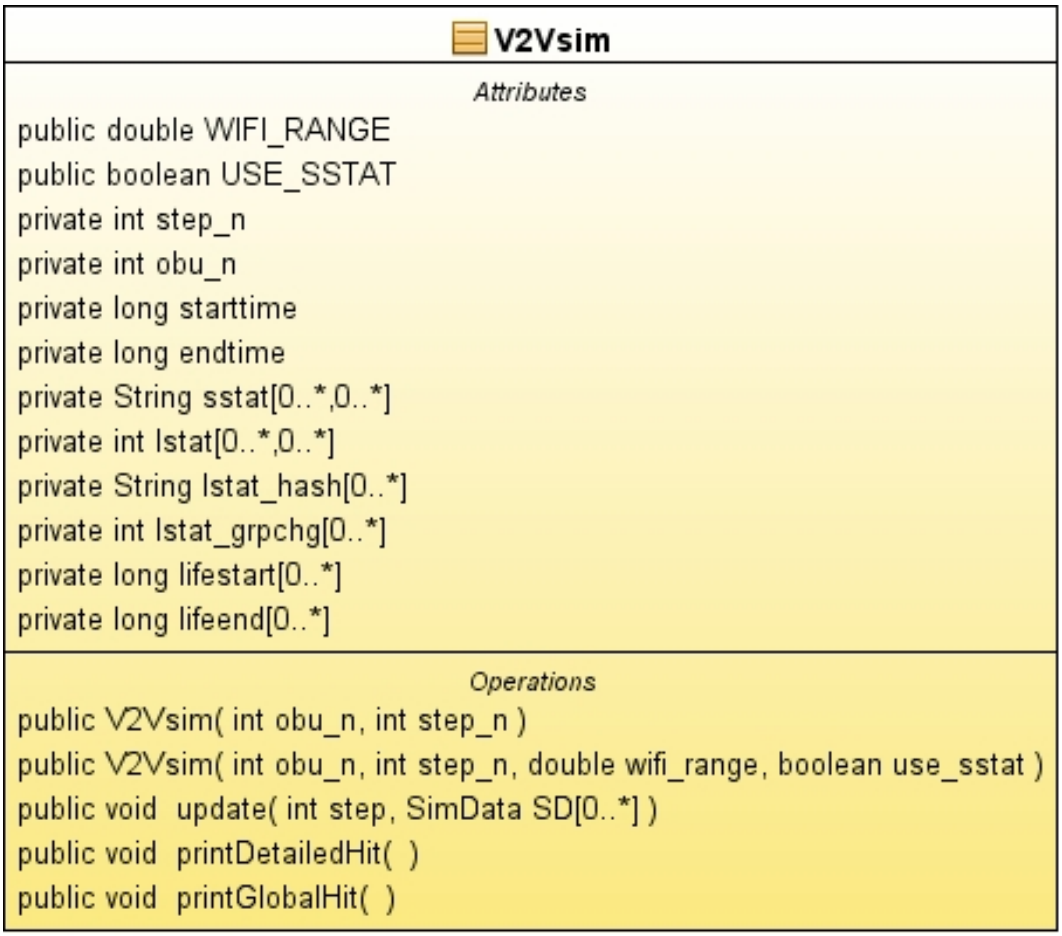

Figura 4.9: Class diagram definitivo - Classe V2Vsim

La decisione di non integrare le strategie V2V all'interno dell'OBU stessa deriva dal fatto che queste non sono ancora state testate completamente e anche dal fatto che le OBU dovrebbero conoscere le informazioni l'una dell'altra per potersi aggregare.

Una futura implementazione, in termini di sviluppo, dovrà seguire le orme di quanto fatto per il V2I, ovvero i dati dovrà inviarli l'OBU "in etere", che verrà simulato appunto da questa classe, mentre le funzioni per le strategie e gli algoritmi di aggregazione dovranno essere inclusi nell'OBU stessa.

Per maggiori dettagli si faccia riferimento all'appendice 6.1 e alle Conclusioni e sviluppi futuri al termine di questo documento.

4.2 Il package OBU

Questo package, che contiene al suo interno la classe "OBU" (sezione 4.2.1), ha come scopo la simulazione dell'architettura reale dell'OBU stessa (vedi figura 3.1), fornendo tutti gli "oggetti" e i "dati" necessari al suo compimento.

Per oggetti s'intendono la classe "Point" (sezione 4.2.5) e "StraightLine2D" (sezione 4.2.6), che implementano rispettivamente le funzioni relative ai punti nello spazio a uno, due e tre dimensioni e alle rette nello spazio a due dimensioni.

Per dati s'intendono invece le strutture dati implementate dalle classi "Sdata" (sezione 4.2.4) e "Vdata" (sezione 4.2.3), che semplificano rispettivamente la gestione dei dati della simulazione e del veicolo e mettono a disposizione funzioni per il loro corretto utilizzo.

Per quanto riguarda invece le classi "Techniques","GPS_2Dconverter" e "CompressData" (paragrafi 4.2.2, 4.2.7 e 4.2.8), queste altro non sono che estensioni della classe OBU, così fatte per suddividere meglio, anche a livello comprensivo, le diverse specializzazioni. In particolare la classe "Techniques" implementa le tecniche *communication-saving* fino ad ora descritte.

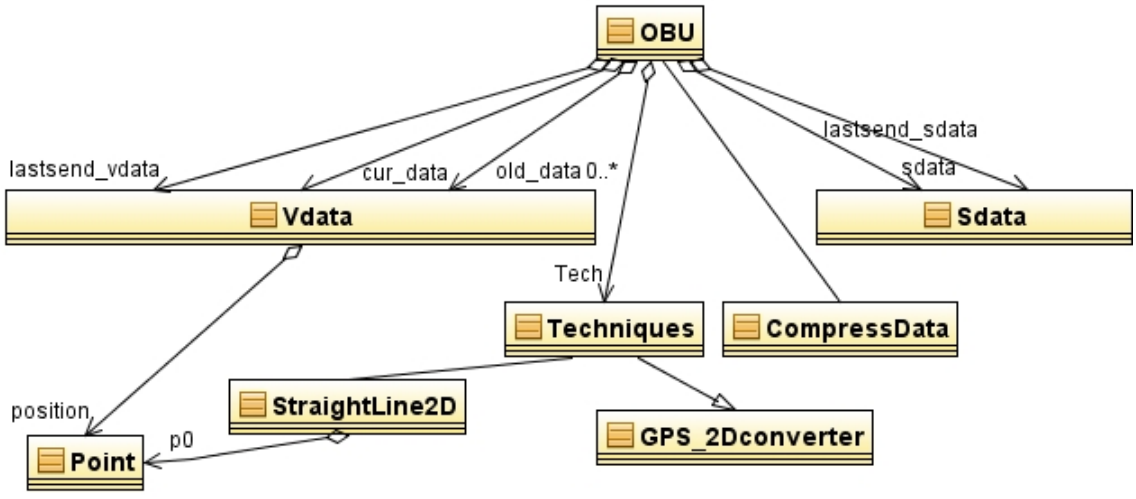

Figura 4.10: Class diagram definitivo - Package OBU

4.2.1 La classe OBU

Già dal nome della classe, che riprende il nome del package, ne si può già intuire il contenuto, ovvero il "*core*" stesso del pacchetto.

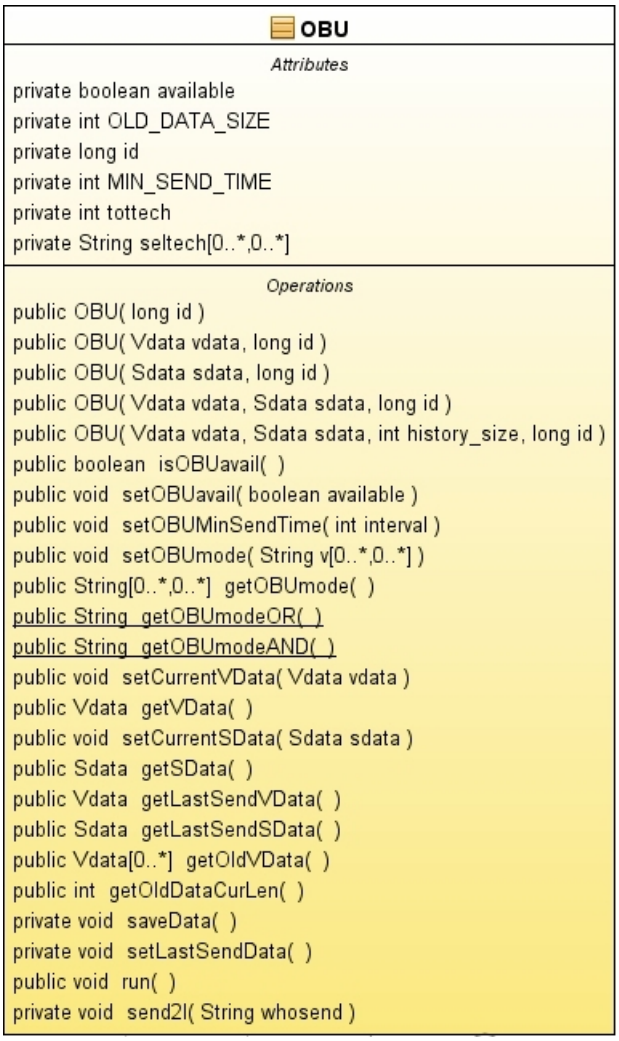

Figura 4.11: Class diagram definitivo - Classe OBU

Tramite diversi tipi di costruttori, come visibile dal class diagram in figura 4.11, si può configurare la classe direttamente all'istanziamento. Unico parametro indispensabile è i'identificativo (id) dell'OBU, che, come nella realtà, dovrà essere univoco. Procedendo nella descrizione, si dettagliano le funzioni implementate in questa classe.

La funzione "setOBUavail" permette di attivare e disattivare l'OBU, mentre "isOBUavail" ne restituisce lo stato.

La funzione "setOBUMinSendTime" imposta il tempo minimo tra una comunicazione e la successiva, questo dovrà essere ovviamente positivo.

La funzione "setOBUmode" , che richiede in ingresso una matrice di stringhe nel formato {{*getAllowTech(x),getOBUmodeOR()*},{*...,getOBUmodeAND()*},...}

ad esempio:
"**SPACE SAMPLING and TIME SAMPLING or REGRESSION or**"

serve per impostare le tecniche che verranno utilizzate dall'OBU per il *communication-saving*. Le stringhe che compongono la matrice sono definite nella classe "Techniques" (vedi sezione 4.2.2). Le funzioni *getOBUmode* associate restituiscono informazioni sullo status attuale della modalità impostata, oltre alla definizione delle stringhe *and* e *or*.

Le funzioni di tipo *setCurrentXData*, dove X sta per S o V, impostano rispettivamente i dati della simulazione e del veicolo mentre quelle di tipo *getXData* associate restituiscono i valori correnti memorizzati nell'OBU; quelle di tipo *getLastSendXData* restituiscono l'ultimo valore inviato, infine, quelle di tipo *getOldXData* forniscono la *history* dei dati la cui lunghezza può essere definita in fase di istanziamento con il parametro "history_size".

Siccome la gestione della *history* è molto delicata, considerando anche il fatto che le dimensioni sono variabili fino al suo riempimento, questa è resa trasparente dalle funzioni "getOldDataCurLen","saveData" e "setLastSendData".

Infine, il cuore dell'OBU è la funzione "run" che utilizza le tecniche della classe "Techniques" (vedi sezione 4.2.2) secondo la modalità impostata e la funzione "send2I" per inviare i dati alla classe "I" (vedi sezione 4.1.7).

4.2.2 La classe Techniques

Come già descritto in precedenza, questa classe, che è strettamente legata alla classe "OBU" (vedi sezione 4.2.1), implementa le tecniche *communication-saving* obiettivo di questa ricerca. Data l'importanza di tutto ciò, si proseguirà con la descrizione dettagliata di ogni funzione, ovvero di ogni tecnica se si tralasciano le funzioni di tipo *tech* utilizzate come *wrapper*.

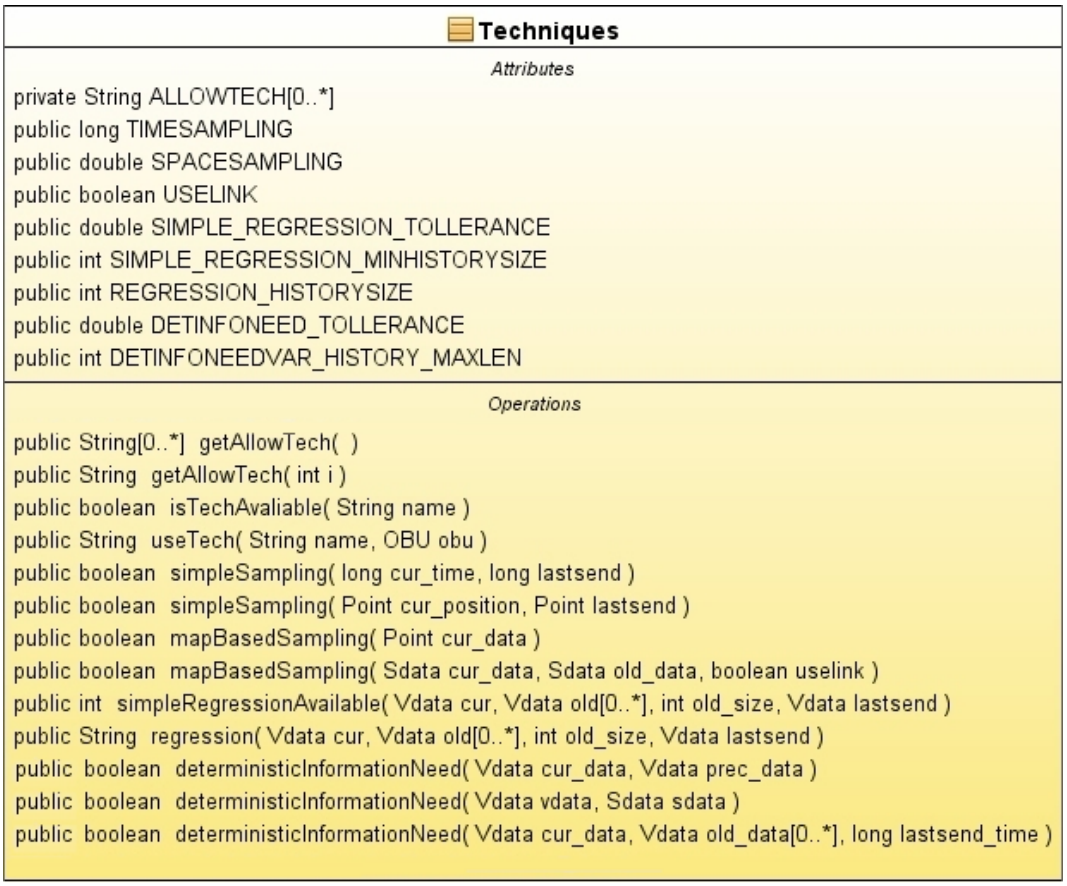

Figura 4.12: Class diagram definitivo - Classe Techniques

Simple sampling

Con questa tecnica si decide di comunicare ad intervalli regolari di tempo e/o di spazio. L'unità di misura risulta a tal fine indifferente.

Esempio di codice:

```
/** Time parameter used by simpleSampling (time) */
public long TIMESAMPLING = 2000;

/**
* Uses simpleSampling (time) technique (*) to
* determinate if informations must be send 2I.
* (*) Send if last communication time overtake
*      TIMESAMPLING [time measure unit] from now
*
* @param cur_time Current OBU data
* @param lastsend Last time OBU send information 2I
* @return true (send), false (not send)
*/
boolean SimpleSampling( long cur_time, long lastsend )
{

        if( ( cur_time - lastsend ) >= TIMESAMPLING )
                return true;

        return false;

}
```

Per quanto riguarda un utilizzo ad intervalli spaziali:

```
/** Distance parameter used by simpleSampling (distance)*/
    public double SPACESAMPLING = 1609.344;

/**
 * Uses simpleSampling (distance) technique (*) to
 * determinate if informations must be send 2I.
 * (*) Send if last communication distance overtake
 *     SPACESAMPLING [distance measure unit] from now
 *
 * @param cur_position Current OBU data
 * @param lastsend Last position in which OBU sent
    informations 2I
 * @return true (send), false (not send)
 */
boolean SimpleSampling( Point cur_position, Point lastsend
   )
{

        if( lastsend.distance( cur_position ) >=
           SPACESAMPLING )
                return true;

        return false;

}
```

La classe "Point"(vedi sezione 4.2.5) si occupa di calcolare le distanze tra due punti (1D,2D o 3D) della mappa in modo trasparente; anche in questo caso l'unità di misura non è influente. Il valore di ritorno è semplicemente un booleano con valore *true* se la comunicazione deve avvenire, *false* altrimenti.

Map-based sampling

Nel caso della tecnica *Map-based sampling*, si distingue dalla versione *on-line* e quella *off-line*. Nella prima si utilizza Google Geocoding API [13] per verificare se il veicolo si trova in un *hot-point* ad esempio un incrocio (vedi sezione 4.3.1), mentre nella seconda si verifica se il veicolo ha cambiato strada o segmento di strada. In quest'ultimo caso i dati sono presi dal simulatore di traffico.

```
/**
 * Uses mapBasedSampling technique (*) to
 * determinate if informations must be send 2I.
 * -- ONLINE VERSION -- Using MapQuest
 * (*) Send if current point is an "hot point".
 * @param cur_data Current OBU data
 * @return true (send), false (not send)
 */
public boolean mapBasedSampling( Point cur_data )
{
   Point pt;
   //#ifdef UTM_COORDS // UTM (Vissim)
         pt = new Test.Maps_fromUTM().bologna(cur_data);
   // #elsif GTM_COORDS // GTM (Realty?)
   //    pt = cur_data;
   // #else
   //    ...
   // #endif
   MapQuest Map = new MapQuest(pt);

   if( Map.isHotPoint() )
            return true;

   return false;
}
```

Dal codice si può vedere che le coordinate UTM devono essere trasformate in GTM; in una futura implementazione con linguaggio C si potrebbero utilizzare i costrutti evidenziati nei commenti.

Nella versione *off-line* (codice sottostante), si può vedere il semplice utilizzo dei dati presi dal simulatore di traffico. Se si attiva la modalità USELINK la funzione segnalerà anche i cambiamenti di segmento e non solo quelli di strada.

```
/** Needs only to change the link for sending */
public boolean USELINK = false;

/**
 * Uses mapBasedSampling technique (*) to
 * determinate if informations must be send 2I.
 * -- OFFLINE VERSION -- Using SData
 * (*) Send if car change road (or link).
 * @param cur_data Current OBU simulation data
 * @param old_data OBU simulation data at last send
 * @param uselink Using change of road and link
 * @return true (send), false (not send)
 */
public boolean mapBasedSampling( Sdata cur_data, Sdata
   old_data, boolean uselink )
{
   if( ! uselink )
       return ( cur_data.getRouteID() != old_data.
          getRouteID() );

   return ( cur_data.getRouteID() != old_data.getRouteID()
      )
          ||
          ( cur_data.getLinkID() != old_data.getLinkID() );
}
```

Entrambe le versioni del *Map-based sampling* dovranno essere adattate all'architettura reale dell'OBU e richiedere le informazioni direttamente alle mappe memorizzate su di essa (vedi figura 1.4).

Deterministic information-need

Di questa tecnica ne sono state implementate tre versioni: la prima *off-line* richiede al simulatore la velocità teorica nel tratto in percorrenza; la seconda *on-line* ha lo stesso funzionamento della prima, ma i dati sono richiesti tramite Google Geocoding API [13] (vedi sezione 4.3.1); infine, la terza verifica se la media delle n velocità trascorse si discosta più di una certa tolleranza dalla velocità attuale.

Tutte e tre le versioni si basano su differenze di velocità in quanto esse determinano un cambiamento nello stato del veicolo, ad esempio code, semafori, curve, etcetera.

```
/**
 * Uses deterministicInformationNeed technique (*) to
 * determinate if informations must be send 2I.
 * -- OFFLINE VERSION using simulator's data
 * (*) Send if current speed is different from Map speed.
 * @param vdata Current OBU Vehicle data
 * @param sdata Current OBU Simulation data
 * @return true (send), false (not send)
 */
private boolean deterministicInformationNeed( Vdata vdata,
   Sdata sdata )
{

    if( Math.abs( vdata.getSpeed() - sdata.
       getTheoricalSpeed() ) >= DETINFONEED_TOLLERANCE )
         return true;

    return false;
}
```

Dal codice si può vedere come sia immediato il calcolo per questa tecnica, che richiede solo due parametri in ingresso.

```
/** Speed tollerance parameter used by
   deterministicInformationNeed */
public double DETINFONEED_TOLLERANCE = 10;
/**
 * Uses deterministicInformationNeed technique (*) to
 * determinate if informations must be send 2I.
 * -- ONLINE VERSION using MapQuest
 * (*) Send if current speed is different from Map speed.
 * @param cur_data Current OBU data
 * @return true (send), false (not send)
 */
private boolean deterministicInformationNeed( Vdata
   cur_data, Vdata prec_data ) {
    if( prec_data == null )
        return false;

    Point toGTM = new Test.Maps_fromUTM().bologna( cur_data
       .getPosition() );
    MapQuest Map = new MapQuest( toGTM , 1 );
    //MapQuest Map = new MapQuest( cur_data.getPosition() ,
        1 ); // No UTM

    toGTM = new Test.Maps_fromUTM().bologna( prec_data.
       getPosition() );
    Map.setTravelTo( toGTM );
    //Map.setTravelTo( prec_data.getPosition() ); // No UTM

    if( Math.abs( Map.getSpeed() - cur_data.getSpeed() ) >=
        DETINFONEED_TOLLERANCE )
        return true;

    return false;
}
```

Come nel caso precedente il calcolo è lo stesso, ma le informazioni sono richieste tramite Google Geocoding API.

```
/** Speed tollerance parameter used by
   deterministicInformationNeed */
public int DETINFONEEDVAR_HISTORY_MAXLEN = 50;

/**
 * Uses deterministicInformationNeed - technique (*) to
 * determinate if informations must be send 2I.
 * (*) Send if current speed is different from average of
    old speeds.
 * @param cur_data Current OBU data
 * @param old_data OBU History data
 * @param lastsend_time Last time OBU sent informations 2I
 * @return true (send), false (not send)
 */
private boolean deterministicInformationNeed( Vdata
   cur_data, Vdata, old_data[],long lastsend_time )
{

    double avg = 0;
    int i = 0;

    final int histmax = DETINFONEEDVAR_HISTORY_MAXLEN
                         <
                        old_data.length
                         ?
                        DETINFONEEDVAR_HISTORY_MAXLEN
                         :
                        old_data.length ;

    for(; i<histmax; i++ )
        if( ( old_data[i] == null )
             ||
            ( old_data[i].getTime() < lastsend_time )
          )
            break;
        else
            avg += old_data[i].getSpeed();
```

```
    if( i != 0 )
        avg /= i;


    if( Math.abs( avg - cur_data.getSpeed() ) >=
       DETINFONEED_TOLLERANCE )
        return true;

    return false;

}
```

A differenza delle altre due tecniche di tipo *Deterministic information-need*, questa è più pesante dal punto di vista computazionale ed introduce un nuovo parametro [1] per dimensionare il numero di velocità passate su cui calcolare la media che verrà confrontata con la velocità corrente dell'OBU.

Si precisa che in caso di *history* insufficiente, ovvero un numero di dati passati inferiore a quelli richiesti, l'algoritmo utilizzerà comunque tutti quelli a disposizione.

Si presume che potendo variare i due parametri [2] si possa giungere ad un buon compromesso tra numero di informazioni inviate e reale necessità di informazioni. Tali considerazioni però spetteranno alla parte di analisi (vedi sezione 5.3.4).

[1] DETINFONEEDVAR_HISTORY_MAXLEN
[2] DETINFONEED_TOLLERANCE e DETINFONEEDVAR_HISTORY_MAXLEN

Simple regression

Questa tecnica, a discapito del nome, non implementa una vera tecnica di regressione, ma verifica se il tragitto percorso, ovvero i punti che lo definiscono, sono interpolabili a meno di una certa tolleranza da una retta.

L'equazione della retta è determinata da due punti relativi alla posizione dell'OBU, quello di inizio (sempre l'attuale) e quello di fine (variabile), all'interno della *history* di quest'ultima.

La verifica si basa sull'interpolazione (a meno di una tolleranza) di tutti gli altri punti tra questi compresi nell'intervallo di tempo trascorso.

In caso di *history* insufficiente o punti che si discostano troppo dalla retta la funzione dichiarerà l'impossibilità a procedere, mentre se i punti sono facilmente interpolabili la funzione restituisce l'indice della *history* che corrisponde al punto di fine.

```
/** Point distance tollerance parameter used by
    simpleRegressionAvailable */
public double SIMPLE_REGRESSION_TOLLERANCE = 10;

/** Minimum history dimension parameter used by
    simpleRegressionAvailable */
public int SIMPLE_REGRESSION_MINHISTORYSIZE = 10;

/**
 * Uses simpleRegressionAvailable to determinate if
 * simpleRegression technique is available on current data.
 *
 * @param cur Current OBU data
 * @param old OBU History data
 * @param old_size OBU History data (right) size
 * @param lastsend Vehicle data about last OBU's
     information sending 2I
 * @return  Oldest data index if available (-1 if not
     available)
 */
public int simpleRegressionAvailable( Vdata cur,
```

```
                                        Vdata old[],
                                        int old_size,
                                        Vdata lastsend )
{
    if( old_size <= this.SIMPLE_REGRESSION_MINHISTORYSIZE )
        return -1;

    int oldest = 0;
    for(; oldest<old_size; oldest++ ){
        if( old[oldest].getTime() < lastsend.getTime() )
            break;
    }

    if( oldest <= this.SIMPLE_REGRESSION_MINHISTORYSIZE )
        return -1;

    for( int i = (oldest-1); i>=this.
       SIMPLE_REGRESSION_MINHISTORYSIZE; i-- )
    {
        boolean avail = true;
        StraightLine2D sl = new StraightLine2D( cur.
           getPosition(), old[i].getPosition() );

        for( int j=0; j<=i; j++)
            if( sl.distanzaConPunto(old[j].getPosition())
               >= this.SIMPLE_REGRESSION_TOLLERANCE )
            {
                avail = false;
                break;
            }

        if(avail)
            return i; // index
    }

    return -1;
}
```

Regression

Il modello di regressione utilizzato è il modello di regressione lineare semplice che si ottiene supponendo una relazione lineare tra X e Y, ovvero $Y = \beta_1 + \beta_2 X + e$ dove β_1 e β_2 sono i parametri della cosiddetta retta di regressione, i quali devono essere opportunamente valutati sulla base delle osservazioni.

Il metodo utilizzato è quello dei minimi quadrati dove, per stimare i parametri β_1 e β_2 del modello di regressione si estrae un campione costituito da n coppie di valori $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$, dove $x_1, x_2, ..., x_n$ sono i valori della variabile esplicativa e $y_1, y_2, ..., y_n$ sono i valori assunti dalla variabile dipendente. Le osservazioni possono essere rappresentate in un grafico a dispersione come esemplificato nella figura 4.13.

Figura 4.13: Regressione lineare - grafico a dispersione

Per stimare β_1 e β_2 si utilizza il metodo dei minimi quadrati ordinari. Questo è un metodo non parametrico in quanto non richiede alcuna ipotesi sulla distribuzione dell'errore e quindi della variabile dipendente. Le stime sono costituite dai valori dei parametri cui corrisponde la retta che interpola al meglio i dati. A tal fine si considerano le distanze dei punti (x_i, y_i) dalla retta di regressione, ovvero gli errori di previsione (scarti) $e_i = y_i - (\beta_i + \beta_2 x_i)$ $peri = 1, 2, ..., n$.

Le stime di β_1 e β_2 sono scelte in modo tale da minimizzare le distanze dei punti (x_i, y_i) dalla retta di regressione stimata. Poiché alcune distanze sono positive e altre negative, si considera la somma delle distanze al quadrato.

Dopo alcuni passaggi algebrici (vedi [33]), che non si vogliono ulteriormente dettagliare, si ottengono infine i parametri della retta di regressione stimata, ad esempio quella in figura 4.14.

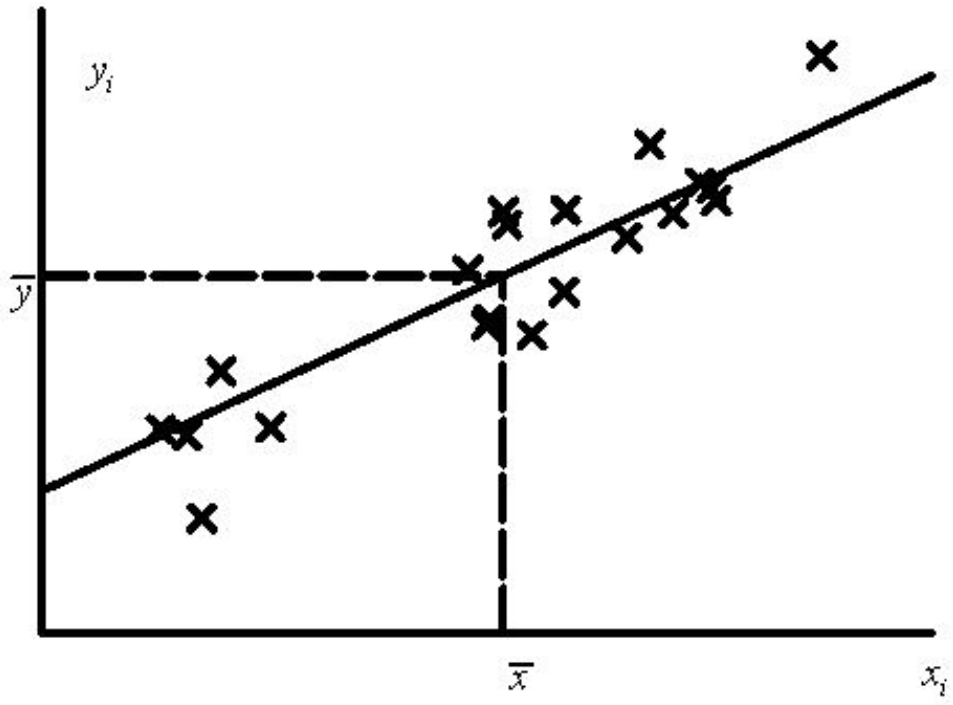

Figura 4.14: Regressione lineare - retta stimata

L'algoritmo di seguito implementa quanto appena descritto.

```
/** History dimension parameter used by regression */
public int REGRESSION_HISTORYSIZE = 10;

/**
 * Regression techinque.
 * Uses linear regression and least minimum squares.
 *
 * @param cur Current OBU data
 * @param old OBU History data
 * @param old_size OBU History data (right) size
 * @param lastsend Vehicle data about last OBU's
    information sending 2I
 * @return Straightline equation (two point) p2id;xp1;yp1;
    xp2;yp2
 */
public String regression( Vdata cur, Vdata old[], int
   old_size, Vdata lastsend )
{
    if( old_size <= this.REGRESSION_HISTORYSIZE )
        return null;

    // Find the value that preceedings last sent
    int oldest = 0;
    for(; oldest<old_size; oldest++ )
        if( old[oldest].getTime() < lastsend.getTime() )
            break;

    if( oldest <= this.REGRESSION_HISTORYSIZE )
        return null;

    double x[] = new double[ this.REGRESSION_HISTORYSIZE ];
    double y[] = new double[ this.REGRESSION_HISTORYSIZE ];
    x[0] = cur.getPosition().getX();
    y[0] = cur.getPosition().getY();
    for( int i = 1; i<this.REGRESSION_HISTORYSIZE; i++ )
    {
```

```
        x[i] = old[i-1].getPosition().getX();
        y[i] = old[i-1].getPosition().getY();
    }

    double mediax = 0, mediay = 0;
    double sommax = 0, sommay = 0;
    double varx=0, vary=0, varxy=0;

    for(int i=0; i<this.REGRESSION_HISTORYSIZE; i++)
    {
        sommax += x[i];
        sommay += y[i];
    }
    mediax = sommax/this.REGRESSION_HISTORYSIZE;
    mediay = sommay/this.REGRESSION_HISTORYSIZE;

    for(int i=0;i<this.REGRESSION_HISTORYSIZE;i++)
    {
        double f=(x[i]-mediax);
        double g=(y[i]-mediay);
        varx += f*f;
        vary += g*g;
        varxy += f*g;
    }

    double intercetta = varxy / varx;
    double pendenza = mediay - intercetta * mediax;
    //double coeff = varxy / (Math.sqrt(varx*vary) );

    StraightLine2D sl = new StraightLine2D( intercetta,
       pendenza );

    String ret = (this.REGRESSION_HISTORYSIZE - 1)+
       SPLIT_RET_VALUE;

    double yy = OBUsim.GenUtil.roundNum( sl.y_from_x( x[0]
       ) , 4 );
```

```
    if( Double.isNaN( intercetta ) || Double.isNaN(
       pendenza ) )
        yy = y[0];

    double xx = x[0];

    ret += xx+SPLIT_RET_VALUE+yy+SPLIT_RET_VALUE;

    yy = OBUsim.GenUtil.roundNum( sl.y_from_x( x[ this.
       REGRESSION_HISTORYSIZE - 1 ] ) , 4);
    if( Double.isNaN( intercetta ) || Double.isNaN(
       pendenza ) )
        yy = y[ this.REGRESSION_HISTORYSIZE - 1 ];

    xx = x[ this.REGRESSION_HISTORYSIZE - 1 ];

    ret += xx+SPLIT_RET_VALUE+yy;

    return ret;
}
```

L'appoggio alla classe "StraightLine2D" (vedi sezione 4.2.6) non risulta strettamente necessario, ma rende il codice leggermente più leggibile.

Il costo computazionale di questo algoritmo, è $O(3n)$, come si può vedere dal codice, dove si utilizzano tre cicli *for* non annidati. Questo algoritmo, dunque, può essere eseguito anche dall'architettura reale senza particolari problemi, considerando le ulteriori agevolazioni date dall'occupazione di memoria che è sempre nell'ordine di $O(n)$ e dal non utilizzo di costrutti condizionali, che "costano" di più delle normali operazioni algebriche qui utilizzate.

4.2.3 La classe Vdata

Questa classe è stata definita al solo scopo di fornire una struttura dati e le relative funzioni che permettono la gestione rapida e trasparente dei dati del veicolo, in particolare contiene le informazioni relative alla posizione (latitudine, longitudine, altitudine), alla velocità ed al tempo.

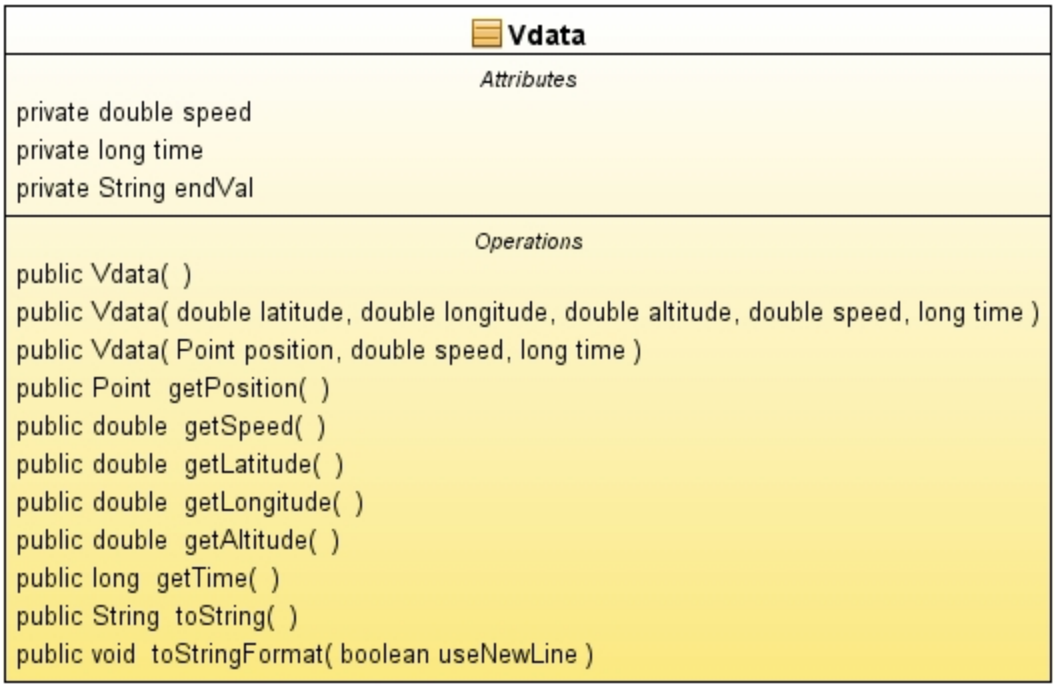

Figura 4.15: Class diagram definitivo - Classe Vdata

Utilizzando una struttura dati *struct* del linguaggio C si può facilmente emulare il comportamento di questa classe, prestando però attenzione allo sviluppo delle funzioni della classe "Point" (vedi sezione 4.2.5), che non potranno essere incluse nell'eventuale tipo di dato, ma dovranno essere sviluppate a parte.

4.2.4 La classe Sdata

Le stesse considerazioni fatte per la classe "Vdata" (vedi sezione 4.2.3) possono essere fatte per questa classe, che è stata definita al solo scopo di fornire una struttura dati e le relative funzioni che permettono la gestione rapida e trasparente dei dati ottenuti dalla simulazione di traffico, in particolare contiene le informazioni relative alla velocità teorica, al distanza effettiva percorsa e agli identificativi del segmento di strada e della strada stessa che il veicolo sta percorrendo.

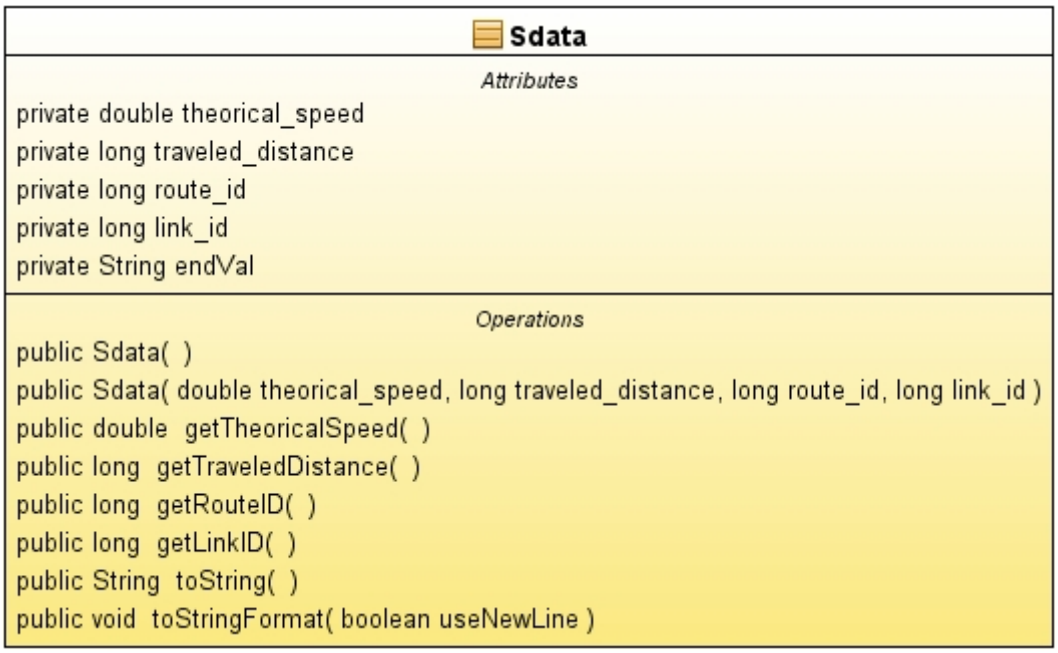

Figura 4.16: Class diagram definitivo - Classe Sdata

Sempre come nel caso precedente, utilizzando una struttura dati *struct* del linguaggio C si può facilmente emulare il comportamento di questa classe, che contiene solo dati di tipo primitivo.

4.2.5 La classe Point

A differenza delle classi precedenti, dove alcune funzioni implementate non risultavano di fondamentale importanza, qui tutto ciò che riguarda il calcolo delle distanze non può essere trascurato.

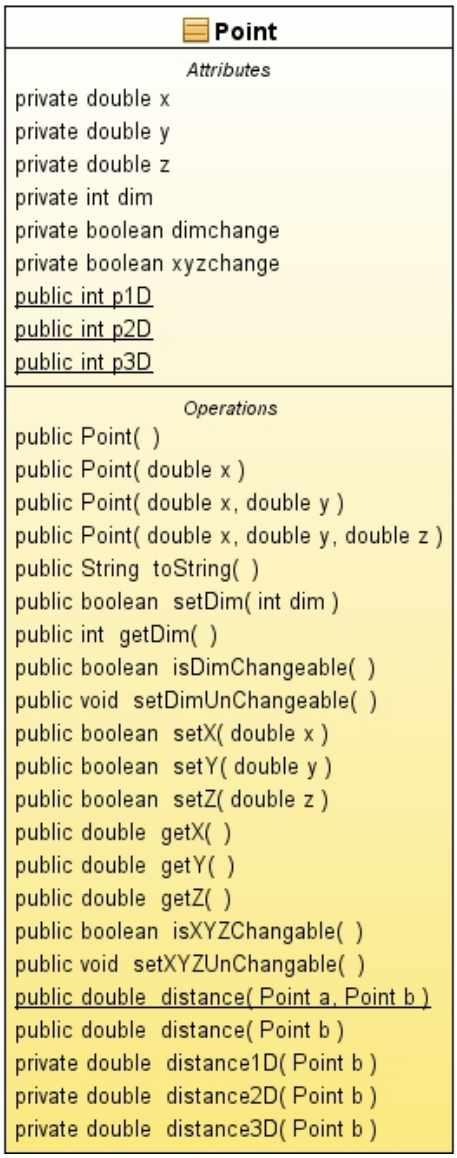

Figura 4.17: Class diagram definitivo - Classe Point

Detto ciò, i dati che essa contiene, ovvero x, y, z e le due variabili *booleane*, sono di tipo primitivo, quindi riconducibili ad una *struct* del linguaggio C.

4.2.6 La classe StraightLine2D

Questa classe, come la precedente classe "Point" (vedi sezione 4.2.5), implementa il tipo di dato "retta" e le funzioni ad essa associate, quali parallelismo, appartenenza, intersezione, distanza, etcetera.

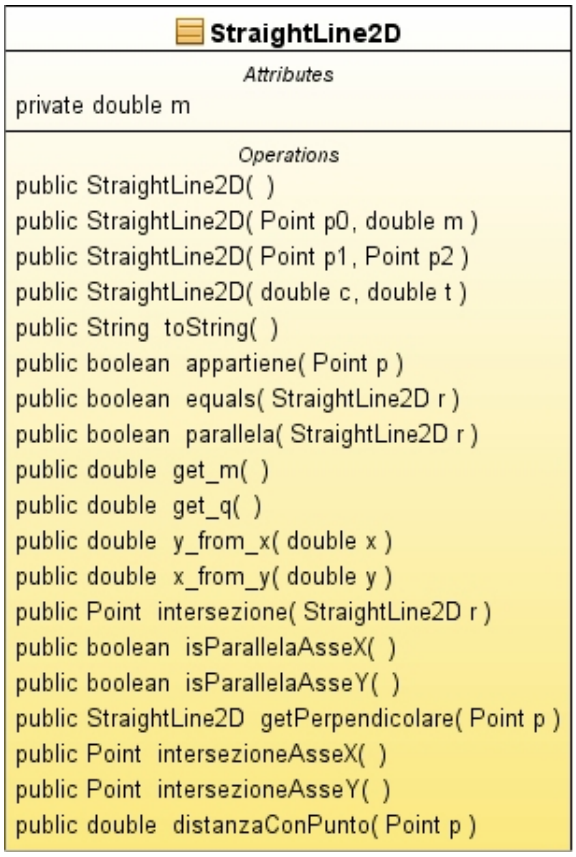

Figura 4.18: Class diagram definitivo - Classe StraightLine2D

La traduzione in linguaggio C, se necessaria, dovrà, oltre definire il tipo di dato con una *struct*, implementare dette funzioni, considerando che, all'interno della classe sono altresì contenute strutture dati complesse quali la classe "Point".

4.2.7 La classe GPS_2Dconverter

Dato che in pratica, ovvero sull'architettura reale dell'OBU (vedi figura 1.4), non potrà essere eseguito con quasi certezza codice Java [18], a sua volta l'utilizzo del package JCoord (vedi sezione 2.3) non sarà consentito, quindi, lo sviluppo di funzioni per il calcolo delle distanze tra punti (GTM) diventa di fondamentale importanza.

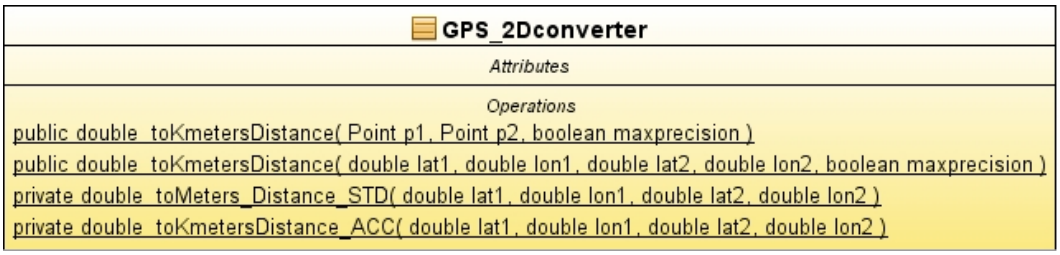

Figura 4.19: Class diagram definitivo - Classe GPS_2Dconverter

Questa classe, appunto, fornisce funzioni per il calcolo della distanza tra due coordinate GTM, dando la possibilità di specificare il livello di accuratezza (standard o accurato) voluto.

La differenza principale tra i due metodi consiste nell'utilizzo di un maggior numero di variabili in quello più accurato, cosa che consente di non trascurare le cifre decimali meno significative nei passaggi dei numerosi calcoli matematici. Ovviamente l'utilizzo di memoria risulta inferiore, anche se di pochi bytes, nel caso meno accurato.

4.2.8 La classe CompressData

Questa classe, i cui unici metodi accessibili pubblicamente sono "compress", "defaultC" e "regressionC", si occupa appunto della codifica e dell'eventuale compressione dei dati che l'OBU deve inviare alla centrale.

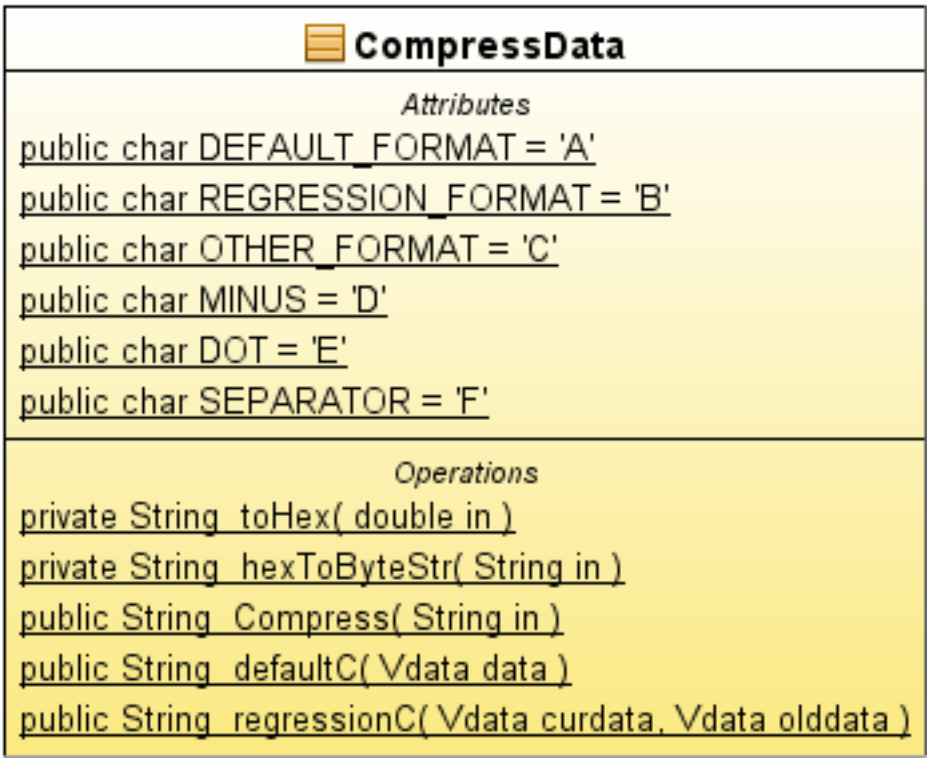

Figura 4.20: Class diagram definitivo - Classe CompressData

Non necessitando molti di caratteri al di fuori di cifre $(0, 1, 2, .., 9)$ e altri simboli quali '−' e '.', ovvero un separatore e identificatori di formato, si è riusciti ad utilizzare una codifica "esadecimale", intesa nel senso dell'utilizzo di quattro bit per rappresentare i caratteri.

Lo sviluppo di tale codifica non è indispensabile ai fini pratici della simulazione, ma risulta di fondamentale importanza per quanto riguarda lo studio del traffico di dati scambiati e dell'effettiva esigenza di tecniche *communication-saving*.

La funzione "compress" è stata utilizzata invece per verificare che la compressione in stile Lempel Ziv è controproduttiva per messaggi di questo tipo (vedi appendice 6.2), ma non si esclude la possibilità di utilizzarla in caso di aggregazione dei messaggi.

La compressione dei dati per mezzo di ulteriori codifiche in stile Huffmann dovrebbe invece essere molto produttiva, in quanto alcuni caratteri (Es. 'A', 'B', 'C') compaiono quasi sempre solo una volta all'inizio del messaggio, mentre altri (Es. 'D', 'E', 'F', '0') compaiono molto spesso.

L'eventuale studio di ulteriori codifiche e/o algoritmi di compressione è lasciato ai futuri sviluppatori, dato che non si conosce nei dettagli specifici l'effettiva architettura, non solo a livello applicativo, che potrebbe già implementare o fornire algoritmi di compressione e/o codifica ottimali.

4.3 Il package Maps

Questo package, che contiene al suo interno le classi "MapQuest" (sezione 4.3.1) e "SureStrArray" (sezione 4.3.2), ha come scopo l'interfacciamento ai servizi forniti da Google Geocoding API [13] come descritto nella sezione 2.5 e sviluppato nelle tecniche "Map-based sampling" (sezione 4.2.2), fornendo tutti gli "funzionalità" e i "dati" necessari al suo compimento.

Per funzionalità s'intende la classe "MapQuest" (sezione 4.3.1), mentre per dati s'intende invece la struttura dati implementa dalla classe "SureStrArray" (sezione 4.3.2).

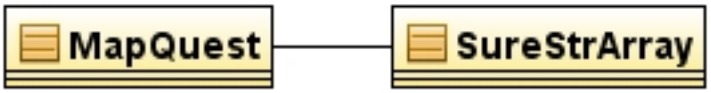

Figura 4.21: Class diagram definitivo - Package Maps

Non essendovi esigenze pratiche di conversione in linguaggi differenti, dato che non si richiede l'esportazione sull'architettura reale, all'interno delle classi questo package si utilizzano strumenti e strutture dati (quali DOM [8], Hashtable, URL, HttpURLConnection, etc.) che non sarebbero facilmente esportabili.

La scarsa quantità di memoria richiesta ha decretato l'utilizzo del *parser* DOM invece che altri strumenti di *parsing* quali SAX[3].

[3]Simple API for XML, http://www.saxproject.org

4.3.1 La classe MapQuest

Tramite i metodi pubblici di questa classe è possibile ottenere tutte le informazioni rilevanti ai fini della simulazione ricavate dal servizio *on-line* Google Geocoding API [13].

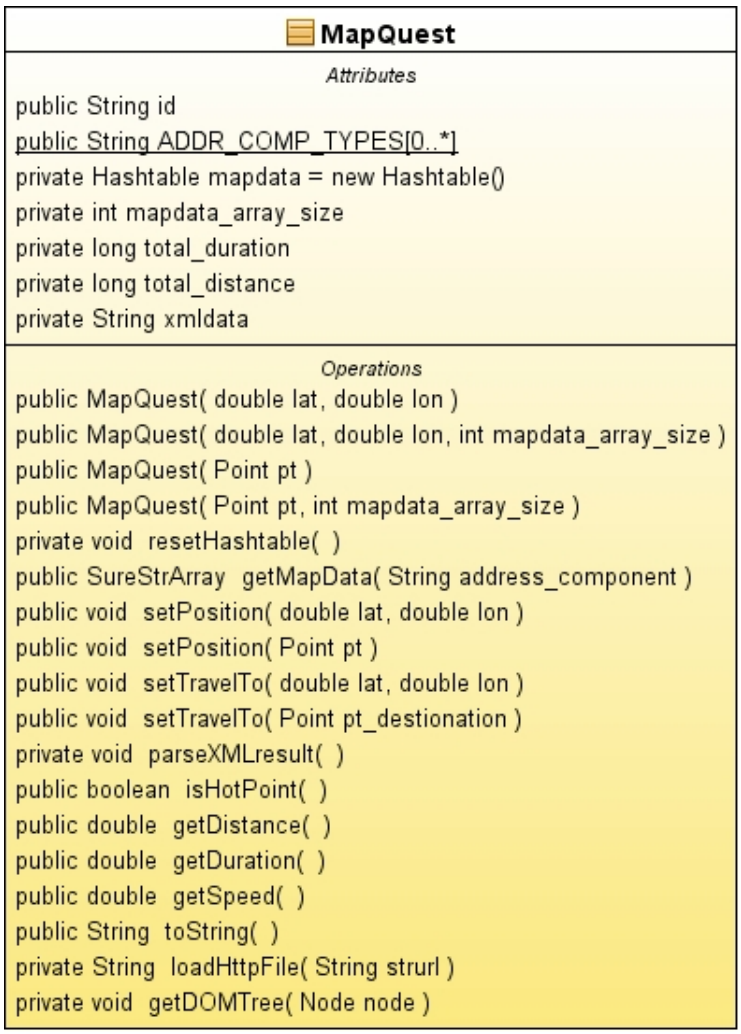

Figura 4.22: Class diagram definitivo - Classe MapQuest

Sempre per mezzo di questi metodi, è anche possibile impostare un tragitto da un punto ad un altro del globo e ottenere le informazioni a riguardo, quali distanza, velocità media e tempi di percorrenza.

4.3.2 La classe SureStrArray

Questa struttura dati è utilizzata dalla classe "MapQuest" (vedi sezione 4.3.1) per memorizzare i dati ricevuti da Google Geocoding API [13], i quali sono organizzati in una struttura XML e possono essere in numero indefinito.

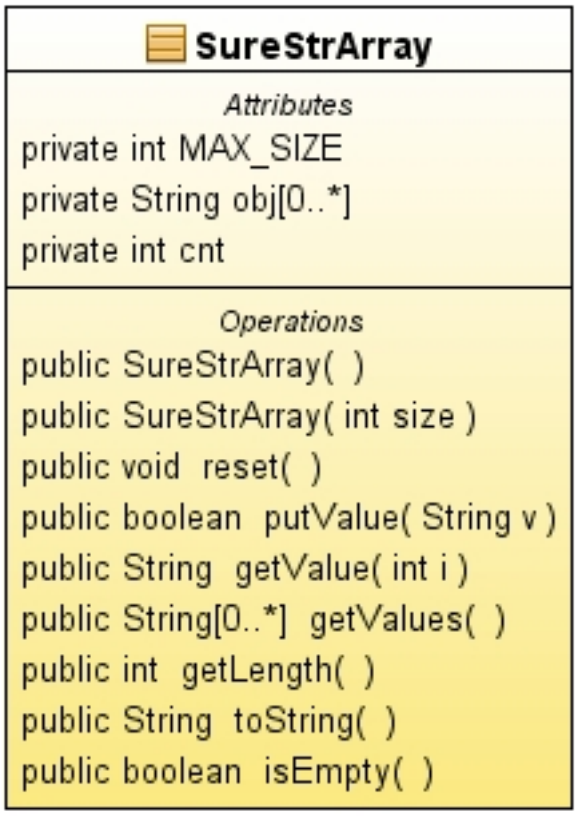

Figura 4.23: Class diagram definitivo - Classe SureStrArray

Con *SureStrArray* s'intende "Array di Stringhe Sicuro" , praticamente un'array di stringhe a dimensione prefissata nel quale i vecchi dati non possono essere sovrascritti da quelli nuovi e le operazioni sono fatte in modo da non creare problemi di *overflow* o similari, infatti, la struttura accetta in input anche una quantità di dati "infinita".

L'eventuale segnalazione di mancata memorizzazione del dato è segnalata dal valore di ritorno della funzione "putValue" in fase di immissione stessa del dato e può essere tranquillamente ignorata.

4.4 Il package Test

Come deducibile direttamente dal nome stesso, il package "Test" ha come scopo quello di contenere le classi utilizzate ai fini dell'analisi, intesa sia come analisi del codice stesso (vedi sezione 4.4.3) che come analisi dell'*output* generato dal programma (vedi sezione 4.4.2). In aggiunta a ciò è stata inserita una classe temporanea che non avrebbe altra localizzazione al di fuori di tale package, dato che è in via sperimentale (vedi sezione 4.4.1).

4.4.1 La classe Maps_fromUTM

Come già introdotto poco prima, questa classe, che contiene solo la funzione "bologna" , non dovrebbe esistere in quanto risolve solo un problema temporaneo nella trasformazione delle coordinate della mappa di bologna da UTM a GTM mediante l'ausilio del package JCoord (vedi sezione 2.3).

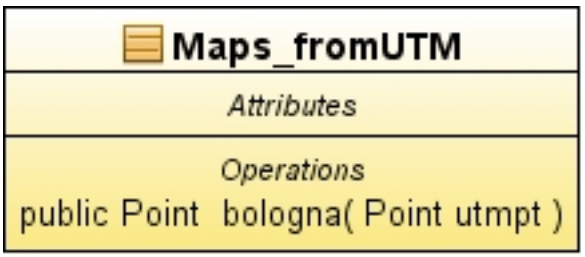

Figura 4.24: Class diagram definitivo - Classe Maps_fromUTM

Non dovendo essere implementata praticamente, come del resto quasi tutte le classi e le funzioni del package in questione (package "Test"), si è scelto di confinare la funzione in questo spazio.

4.4.2 La classe Debug

La classe "Debug" è l'unica che si occupa dell'*I/O* relativo al programma, sia per quanto riguarda lo *standard output* che per quanto riguarda lo *standard error*, che vengono mostrati, salvati su file o scartati dipendentemente dalle impostazioni effettuate.

Figura 4.25: Class diagram definitivo - Classe Debug

Il meccanismo di salvataggio dei file di *log* e dei file contenenti le statistiche calcolate, prevede l'utilizzo della data per conferire l'unicità ai nomi (questo almeno fino al 2038 d.C.).

4.4.3 Le altre classi di test

Come tutti i programmi anche questo necessita della fase di collaudo (detta anche *testing*), ovvero un procedimento utilizzato per individuare le carenze di correttezza, completezza e affidabilità delle componenti software in corso di sviluppo. Essa consiste nell'eseguire il software da collaudare, da solo o in combinazione ad altro software di servizio, e nel valutare se il comportamento del software rispetta i requisiti.

Lo scopo di questo collaudo, effettuato appunto mediante "le classi di test", è di rilevare i difetti tramite i malfunzionamenti, al fine di minimizzare la probabilità che il simulatore abbia dei malfunzionamenti nella normale operatività.

Nessun collaudo può ridurre a zero tale probabilità, in quanto le possibili combinazioni di valori di input validi sono enormi, e non possono essere riprodotte in un tempo ragionevole; tuttavia questo collaudo rende la probabilità di malfunzionamenti abbastanza bassa.

Il lungo utilizzo per l'analisi sperimentale del capitolo 5, infine, verificherà ulteriormente la correttezza del simulatore stesso.

Un esempio di test è riportato in appendice, sezione 6.3.

Capitolo 5

Analisi sperimentale

L'analisi sperimentale con l'ausilio del simulatore sviluppato, intesa anche come scelta degli scenari, dei parametri e delle tecniche su cui effettuare prove sperimentali, è il passo conclusivo e di maggior rilevanza di questo progetto.

Partendo dall'analisi acquisita dai test effettuati durante lo sviluppo del simulatore e ripercorrendo, ma solo nei primi passi, questa strada, si ci è accorti che i risultati ottenuti, oltre alla loro correttezza nella quasi totalità dei casi, salvo piccole incongruenze dovute a fattori pressoché irrilevanti al nostro scopo, erano in molti casi migliorati, sia per merito delle tecniche *communication-saving* che per merito delle altre migliorie apportate.

Considerando gli obiettivi proposti nella sezione 1.2, in parte soddisfatti dall'implementazione del capitolo 4, si prosegue con la scelta degli scenari usati alla sezione 5.1, con la scelta dei parametri alla sezione 5.2, con l'analisi V2I alla sezione 5.3 e con l'analisi V2V alla sezione 5.4. Infine, per delineare un quadro completo del lavoro svolto, si presentano le Conclusioni e gli Sviluppi Futuri al termine di questo capitolo.

5.1 Scenari usati

Il primo passo per intraprendere una corretta analisi sperimentale è quello di scegliere degli scenari che rispecchino la realtà nel modo più fedele possibile. Non sempre questo è possibile, soprattuto nel nostro caso, dato che non si è in possesso di strumenti tali da poter svolgere questo compito in tempi ragionevoli.

La creazione di uno scenario di traffico, utilizzando il simulatore Vissim [40] fornito da PTV [31], è un processo lungo e molto laborioso, che occupa anche settimane di tempo. Vi sono *tool* che permettono l'importazione di mappe reali, quali quelle create da Octotelematics [28], in tempi brevi, ma il loro utilizzo non è ne' libero ne' tanto meno sostenibile da un punto di vista economico.

Considerando i partners affiliati al progetto PEGASUS [29] (vedi [30]), si ci è rivolti all'Università di Bologna [39] per ottenere la mappa, in via di sviluppo, del centro di Bologna (vedi sezione 5.1.1). Purtroppo questa si è rivelata in parte una delusione, dato che non soddisfaceva minimamente i nostri requisiti.

Le soluzioni sono state due: la prima è stata modificare la mappa fornitaci in tempi non esorbitanti; la seconda quella di utilizzare tutti gli scenari di esempio possibili inclusi direttamente nel il simulatore di traffico Vissim [40].

Anche nel secondo caso alcuni problemi, quali la limitazione date dalla licenza ottenuta (di tipo studenti), ad esempio le dimensioni della mappa con massimo 10Km di lato, un numero di semafori non superiore a 20, etc., hanno portato alla scelta quasi forzata di alcuni degli scenari di esempio, ovvero Roma (sezione 5.1.2), Beijing (sezione 5.1.3) e Toll Plaza (sezione 5.1.4).

Nonostante tutto, i dati ottenuti sono risultati molto realistici, soprattuto se confrontati con i precedenti utilizzati in fase di test. La conferma di ciò non deriva da deduzioni fittizie, ma dalle prove sperimentali ottenute attraverso Google Geocoding API [13], anche se non inizialmente sudiate appositamente per questo scopo.

5.1.1 Bologna

Come si può vedere dalle figure 5.1 e 5.2 la mappa in Vissim riporta fedelmente la realtà; in figura 5.3 si può vedere invece lo stesso scenario della figura 5.2, ma con i veicoli in moto, come enfatizzato dai cerchi.

Figura 5.1: Scenario - Bologna dal satellite

Figura 5.2: Scenario - Bologna da Vissim, senza veicoli

Figura 5.3: Scenario - Bologna da Vissim, con veicoli

Le modifiche apportate a questa mappa, dato che l'originale presentava numerosi difetti incompatibili con una simulazione di traffico seria, riguardano principalmente i percorsi seguiti dai veicoli (che tendevano sempre a percorrere le stesse strade in circolo), l'immissione di alcuni semafori e la creazione dei veicoli stessi.

Non avendo a disposizione dati ufficiali sul numero di veicoli che percorrono abitualmente le strade di Bologna, si è cercato di immettere questi dati secondo buonsenso.

Certo tali vincoli non hanno permesso una seria analisi in campo V2V, mentre per quanto riguarda l'analisi V2I le stime risultano valide e perfettamente attendibili, in quanto tali assunzioni non sono correlate al funzionamento delle tecniche *communication-saving*.

Da queste mappe si è cercato di creare una simulazione che, scartando molte vie del centro e reindirizzando alcune vie periferiche, permettesse di studiare veicoli che viaggiassero per alcune ore, cosa solitamente non possibile avendo a disposizione mappe che non possono eccedere i bordi di un quadrato di 10Km per lato.

5.1.2 Roma

Per questo scenario si riportano parte delle informazioni del documento allegato, intitolato "variante via tiburtina, tratto Via di Casal Bruciato Ponte Mammolo, verifica mediante modello di simulazione, Relazione generale" redatto a Perugia nell'aprile del 1999.

In particolare, la descrizione dell'offerta di trasporto, gli impianti semaforici nella 1° soluzione progettuale, gli impianti semaforici nella 2° soluzione progettuale.

Il programma VISSIM consente la ricostruzione della rete stradale e della disciplina di circolazione. Con un modello simile si può tener conto dell'effettiva lunghezza dei tronchi di scambio, delle corsie d'immissione, d'uscita e di preselezione alle intersezioni, così come dei triangoli di visibilità, degli angoli fra le traiettorie conflittuali fra i veicoli e dell'ampiezza delle aree di intersezione. Nella definizione delle caratteristiche delle strade è necessario implementare: gli archi, caratterizzati da numero e modulo delle corsie, e definiti dal loro punto di inizio e di fine oltre che eventualmente da punti intermedi che ne definiscono la geometria; le connessioni fra archi per la modellizzazione dei cambi di direzione (movimenti di svolta alle intersezioni) e per la riduzione o l'aumento del numero di corsie.

Sia per gli archi sia per le connessioni bisogna specificare la velocità di percorrenza desiderata e le zone di rallentamento in prossimità di curve e restringimenti di carreggiata. In entrambi i casi va indicata non una velocità massima di progetto ma va descritta la legge di distribuzione delle velocità desiderate, distinta per le autovetture e per i veicoli pesanti. Per le intersezioni sono stati inseriti i dati relativi al modo di risoluzione dei punti di conflitto: a livelli sfalsati (nelle rappresentazioni grafiche del modello allegate alla presente relazione, le sottovie sono rappresentate in grigio scuro); a precedenza, con l'indicazione della posizione e dei valori relativi al distanziamento spaziale e temporale minimo tra i veicoli, e con limitazione sulle velocità; mediante semafori, con l'indicazione della posizione della linea d'arresto e dei riferimenti relativi all'impianto e ai gruppi semaforici (insieme delle lanterne che cambiano colore nello stesso istante).

L'elevato livello di definizione nella geometria delle strade e il rispetto delle caratteristiche dimensionali è semplificato dalla possibilità di implementare la planimetria numerica di progetto. Tale possibilità è stata inoltre sfruttata per

il caricamento della planimetria generale che agevola la lettura dei luoghi nella rappresentazione a video e nelle immagini allegate. Per la seconda soluzione progettuale nelle simulazioni effettuate per Via dei Monti Tiburtini sono state assunte carreggiate a tre corsie per senso di marcia.
- Gli impianti semaforici nella 1° soluzione progettuale:
Per gli impianti semaforici in corrispondenza dell'intersezione di Via Tiburtina con Via dei Monti Tiburtini e Via F.Fiorentini così come della rotatoria sulle complanari di Via Tiburtina vicino a P.za S.Maria del Soccorso sono stati definiti i gruppi semaforici, la durata del ciclo, delle fasi e delle interfasi, cioè i tempi di giallo e di tutto rosso.

Nella figura 5.4 viene rappresentato graficamente il ciclo semaforico (durata 50s) a quattro gruppi semaforici e 3 fasi dell'intersezione di Via Tiburtina con Via dei Monti Tiburtini e Via F.Fiorentini:

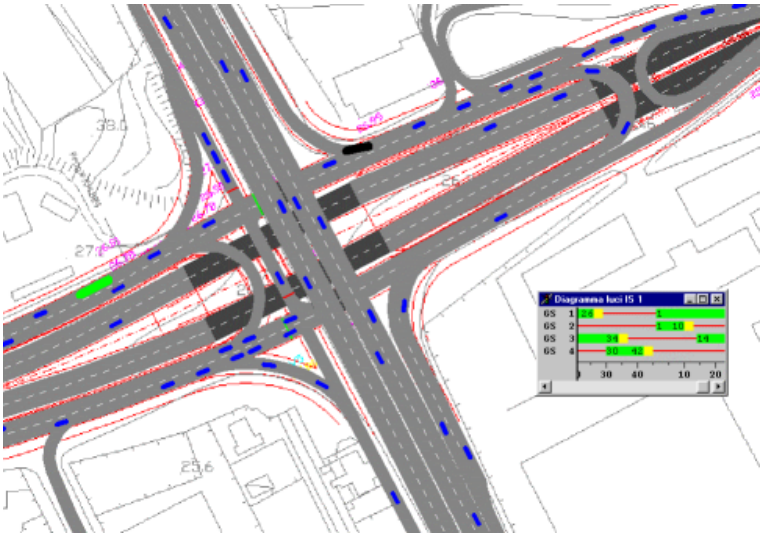

Figura 5.4: Scenario - Roma, planimetria 1

Nella figura 5.5 viene rappresentato graficamente il ciclo semaforico (durata 60s) a nove gruppi semaforici, comprensivi dei due relativi agli attraversamenti pedonali, e 3 fasi della rotatoria sulle complanari di Via Tiburtina vicino a P.za S.Maria del Soccorso:
- Gli impianti semaforici nella 2° soluzione progettuale
Per semplicità di immissione dati, gli impianti semaforici attuati dal traffico a priorità del trasporto pubblico, per l'inversione di marcia in Via Tiburtina

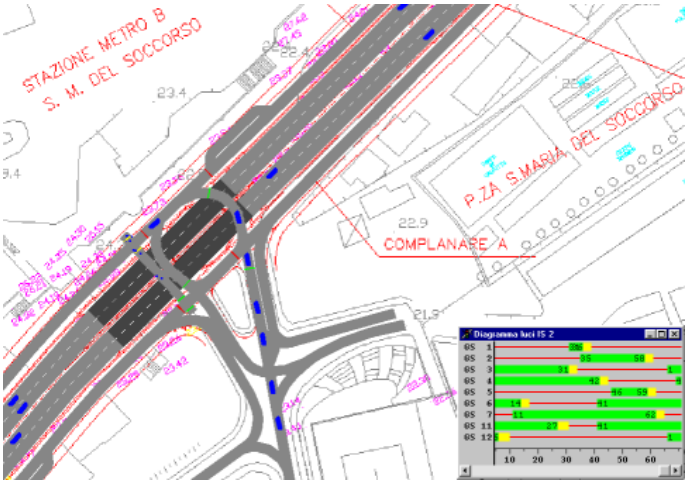

Figura 5.5: Scenario - Roma, planimetria 2

in corrispondenza delle fermate autobus, sono stati sostituiti dall'obbligo del rispetto di precedenze i cui parametri definiti sono tali da rendere del tutto equivalenti i risultati ottenuti.

Il solo impianto definito nel secondo scenario simulato è quindi quello che risolve il conflitto tra le manovre Via Tiburtina centro e Via Tiburtina GRA e tra Via Tiburtina GRA e Via Fiorentini. Nella figura 5.6 viene rappresentato graficamente il ciclo semaforico (durata 50s) a due gruppi semaforici e due fasi.

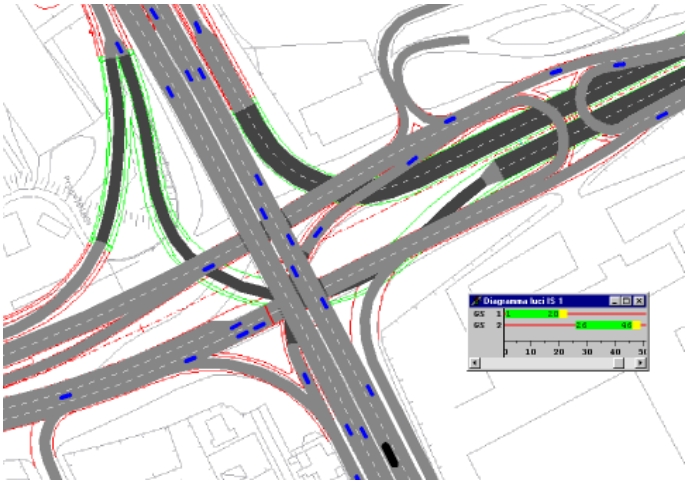

Figura 5.6: Scenario - Roma, planimetria 3

Nella figura 5.7 è possibile vedere il quadro generale, mentre nella figura 5.8 si mostra lo scenario riportato in Vissim senza veicoli.

Figura 5.7: Scenario - Roma, quadro definitivo

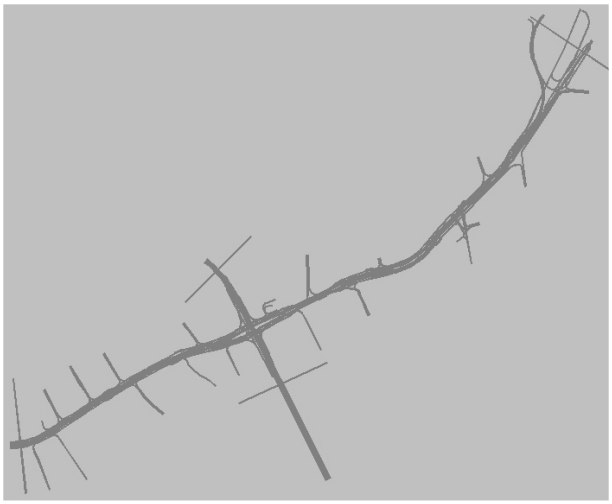

Figura 5.8: Scenario - Roma da Vissim, senza veicoli

Nelle figure 5.9 e 5.10 si mostra rispettivamente lo scenario riportato in Vissim con i veicoli in moto, e uno scenario simile ingrandito.

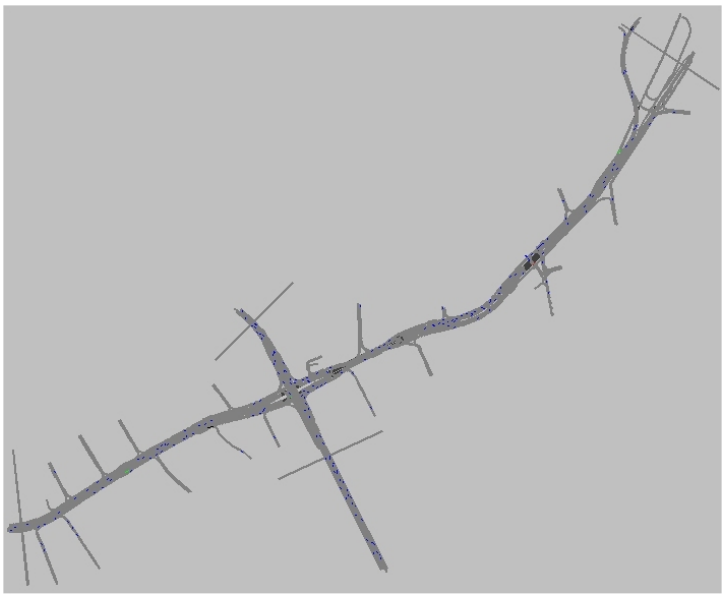

Figura 5.9: Scenario - Roma da Vissim, con veicoli

Le dimensioni di questo scenario risultano di circa 700m per il tratto breve (verticale) e 2200m per quello lungo (orizzontale).

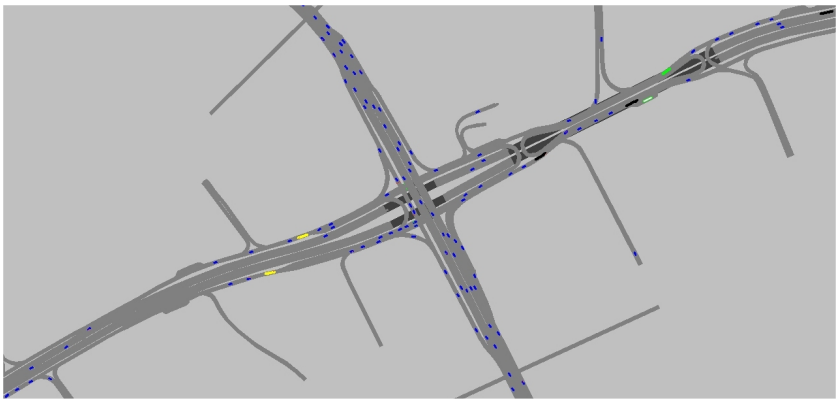

Figura 5.10: Scenario - Roma da Vissim, con veicoli (zoom)

5.1.3 Beijing

Come nel caso di precedente di Roma, questo scenario è ambientato in un incrocio a BeiJing (Pechino). Le informazioni sul traffico, i tempi semaforici, le quantità di veicoli, etc., risultano presumibilmente attendibili. Le dimensioni reali di tale incrocio sono approssimativamente di 1.2 x 1 Km.

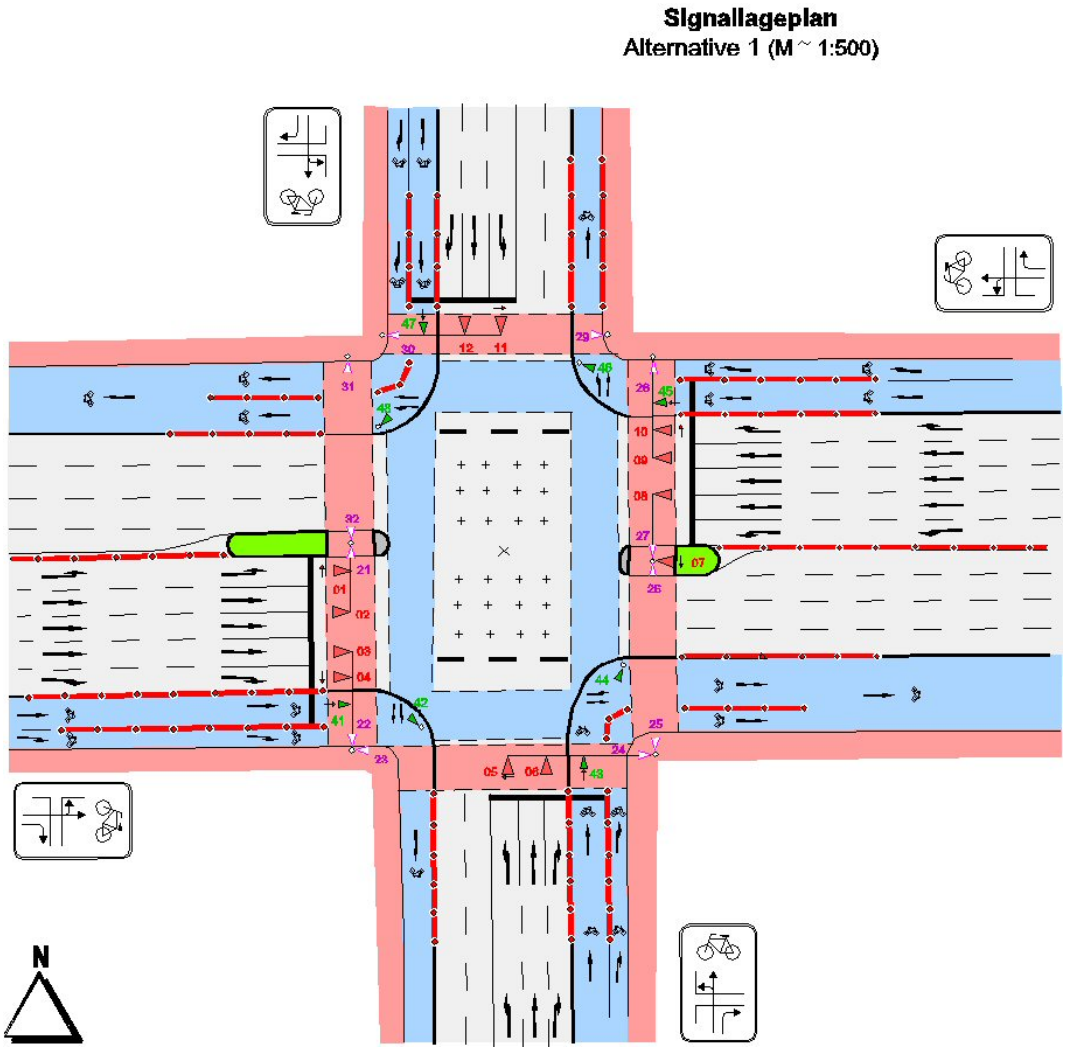

Figura 5.11: Scenario - Beijing progetto

Le figure 5.12 e 5.13 mostrano rispettivamente lo scenario di Beijing nel simulatore di traffico Vissim, e lo stesso scenario ingrandito.

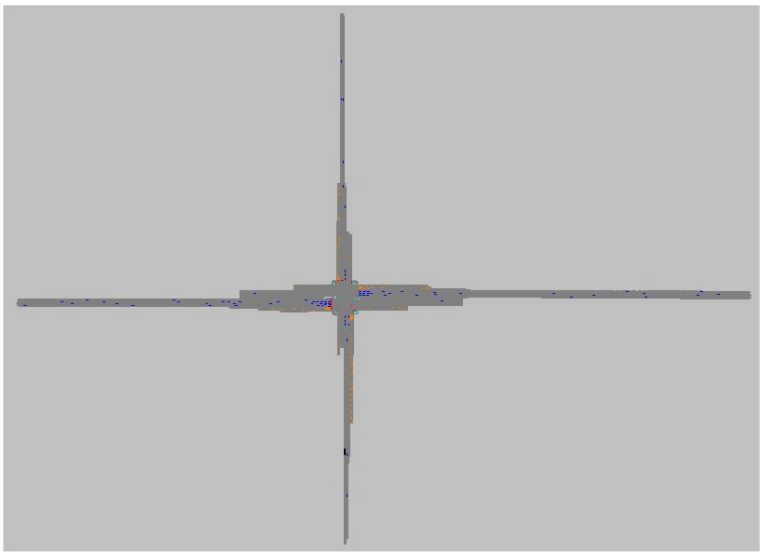

Figura 5.12: Scenario - Beijing da Vissim, con veicoli

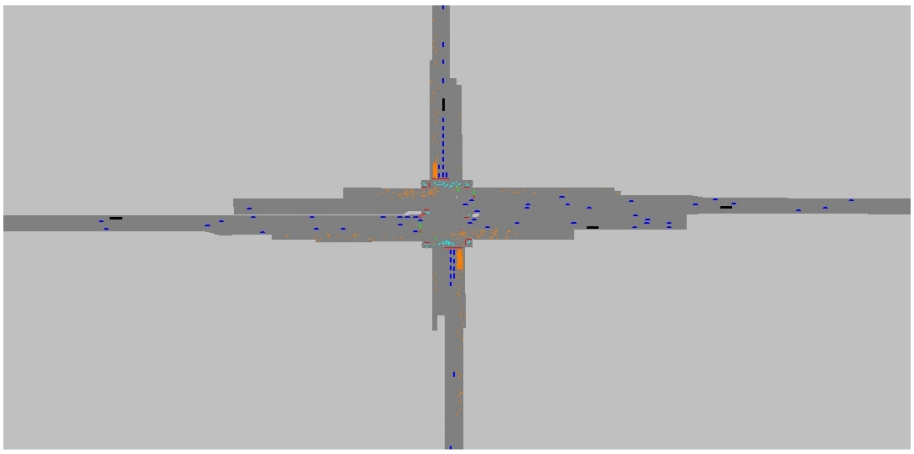

Figura 5.13: Scenario - Beijing da Vissim, con veicoli (zoom)

5.1.4 Toll Plaza

Questo scenario s'intitola "VISSIM demonstration Toll Plaza in Camden, New Jersey, USA".

Il modello presentato è stato utilizzato per analizzare le operazioni sul traffico in Benjamin Franklin Toll Plaza per i veicoli che viaggiavano verso il ponte di Delaware River e provenivano da Camden e quelli provenienti dal New Jersey verso Philadelphia in Pennsylvania.

Oltre a tutto ciò questo modello fu fatto anche per analizzare le violazioni dei conducenti che non pagavano il casello.

I volumi di traffico, le velocità, la percentuale di violazioni e tutti gli altri input inseriti in VISSIM sono dati reali.

La figura 5.14 mostra un'immagine dal satellite al tempo del progetto della simulazione.

Figura 5.14: Scenario - TollPlaza foto da progetto

La figura 5.15 mostra un'immagine dal satellite dell'anno 2010, mentre la figura 5.16 mostra lo scenario così come mostrato da Vissim durante la simulazione.

Figura 5.15: Scenario - TollPlaza da Google Maps

Figura 5.16: Scenario - TollPlaza da Vissim, con veicoli

Le dimensioni reali di questo scenario sono circa 2300 metri per il tratto più lungo, 1400 metri per quello intermedio e 430 metri per quello più corto.

5.2 Parametri usati

Essendo aree di lavoro differenti, l'analisi V2I richiede considerazioni, e quindi parametri e configurazioni, diverse dall'analisi V2V. In questa sezione si presentano le scelte effettuate in merito, giustificandole sulla base dei risultati di test preliminari e dello stato dell'arte attuale.

Da considerare che tutti i risultati possono presentare un "*life filter*" (che caso di applicazione verrà comunque segnalato) ovvero le OBU che hanno un tempo di vita inferiore a questo parametro sono scartate dalle medie.

Gli scenari utilizzati saranno gli stessi, mentre la durata della simulazione e l'intervallo di campionamento dei dati saranno variabili in entrambi i casi.

5.2.1 Parametri per l'analisi V2I

Per fissare dei parametri con cui fare analisi serve innanzitutto definire una linea guida, ovvero un parametro base a cui fare riferimento per i confronti, in modo da sapere sempre se il lavoro svolto ottiene risultati positivi, neutri o negativi.

L'altro aspetto da definire riguarda l'utilizzo dei dati trasmessi, infatti, se non si definiscono dei servizi che li utilizzano non si potrà mai sapere cosa serve e cosa non serve.

Il criterio stabilito per valutare la qualità dei dati trasmessi riguarda quindi la percentuale di errore relativa alle distanze percorse ed alle velocità, considerando ad esempio i seguenti servizi:

- di tipo assicurativo, che basano il premio sulle distanze percorse
- di tipo informativo, quali la stima dei tempi di percorrenza di un tratto stradale
- di tipo preventivo, che verificano le velocità in determinati tratti stradali.

In tutti i casi comunque, ciò che si valuta è la precisione sia delle distanze che delle velocità trasmesse rispetto quelle reali del veicolo.

Per calcolare le distanze reali si è fatto riferimento ai dati riportati dal simulatore di traffico (vedi sezione 4.2.4), ovvero la distanza percorsa dal veicolo durante la simulazione di traffico ed il suo tempo di vita. Dividendo la distanza percorsa per il tempo di vita si è ottenuta la velocità media del veicolo.

In tabella 5.1 è possibile vedere un esempio di questi dati, il cui numero complessivo, nello specifico esempio, era di 776 righe. Da considerare che ognuna delle righe di questa tabella è il riassunto della simulazione di traffico per l'OBU identificata dal parametro "id".

Tabella 5.1: Esempio dati dettagliati V2I

id	hits	steps	data len	tot dist	avg v frv	avg v int	c dist	dist diff	life	c avg v	v diff
0	163	163	3242	809,76	35,79	35,99	812	2,24	81,5	35,87	0,12
1	83	83	1655	461,66	40,5	40,54	468	6,34	41,5	40,6	0,06
2	84	84	1665	461,75	40,04	40,06	468	6,25	42	40,11	0,05
3	1200	1200	24427	3546,91	21,06	21,3	3516	30,91	600	21,1	0,2
4	55	55	1091	277,76	36,83	37,03	283	5,24	27,5	37,05	0,02
5	1200	1200	24666	2726,13	16,19	16,37	2704	22,13	600	16,22	0,15
6	1200	1200	23352	1956,41	11,66	11,75	1951	5,41	600	11,71	0,04
7	1200	1200	24593	3441,14	20,45	20,66	3414	27,14	600	20,48	0,18
8	104	104	2065	522,77	36,41	36,54	528	5,23	52	36,55	0,01
9	109	109	2172	531,85	35,48	35,46	539	7,15	54,5	35,6	0,14
10	46	46	911	243,96	38,86	39,03	250	6,04	23	39,13	0,1

Definizioni dei nomi dei campi delle tabelle:

- "id": identificativo univoco dell'OBU
- "hits": numero di comunicazioni con la centrale
- "steps": numero di intervalli con durata uguale al campionamento del simulatore di traffico in cui l'OBU è presente nella simulazione
- "data len": totale di bytes inviati dall'OBU alla centrale (V2I) nell'arco della sua vita
- "tot dist": distanza totale calcolata percorsa dall'OBU
- "avg v frv": velocità media calcolata senza interpolazione dei dati
- "avg v int": velocità media calcolata con interpolazione dei dati
- "c dist": distanza effettiva percorsa (dati del simulatore di traffico Vissim)
- "dist diff": differenza in valore assoluto tra la distanza effettiva e la distanza calcolata in modo interpolato
- "life": tempo di vita dell'OBU in secondi (tempo totale in cui l'OBU è presente nella simulazione)

- "c avg v": velocità media effettiva (calcolata con dati del simulatore di traffico Vissim)
- "v diff" [1]: differenza in valore assoluto tra la distanza effettiva e la distanza calcolata in modo interpolato

Per quanto riguarda le statistiche generali V2I ulteriori parametri possono essere:

- "campion": parametro usato (o campionamento dei dati effettuato) dalla tecnica in questione
- "Dist err% ok": errore percentuale tra la distanza effettiva e quella calcolata (100*(dist diff)/(c dist))
- "V err% ok": errore percentuale tra la velocità effettiva e quella calcolata (100*(v diff)/(c avg v))
- "obu num": numero totale di OBU che compaiono nella simulazione
- "data x obu": numero di bytes medio inviato da ogni OBU alla centrale (V2I)

Le statistiche dettagliate appena descritte sono state in primo luogo ottenute tramite fogli di calcolo, poi fatte calcolare dal programma stesso, in modo da ottenere una piena validazione dei dati.

Le statistiche più generali, che verranno d'ora in poi presentate in quanto leggibili e riassuntive, sono la media delle statistiche dettagliate sopra citate.
I grafici che saranno mostrati in analisi V2I (sezione 5.3) saranno di due tipi: i primi mostreranno le percentuali di errore della distanza e della velocità, mentre i secondi mostreranno la quantità di dati (in byte) inviati dalle OBU alla centrale.

Entrambi i grafici faranno riferimento allo stesso scenario ben definito e le varie colonne degli istogrammi rappresenteranno una determinata tecnica con determinati parametri.

I parametri scelti per la simulazione sono frutto dei test effettuati per validare il simulatore e sono di seguito riportati.

Considerando che il simulatore di traffico Vissim [40] consente un campionamento dei dati del veicolo fino ad un massimo di 10 per secondo, ovvero ogni *step*

[1] Equivalente a "v diff (int)"

di esecuzione dura 0,1 secondi, questo sarà comunque un limite inferiore sotto il quale non si potrà scendere.

Considerata l'alta mole di dati prodotta, ad esempio 1000 OBU in una simulazione da 3600 secondi con 75 bytes di dati per OBU per step producono circa 257 mega bytes di dati, il campionamento, in alcuni casi, non potrà essere al massimo della finezza, in quanto 257 mega bytes / 0,1 risulta circa 2,51 giga bytes, invece campionando, ad esempio, ogni 0,5 secondi risulterebbe 514 mega bytes, esattamente 5 volte in meno.

Il tempo di campionamento a 0,5 secondi risulta ottimale anche per la definizione della linea guida, infatti, una comunicazione GPRS ha tempi di latenza che si aggirano intorno agli 0,6 0,7 secondi [14] e un invio al di sotto di tale soglia non sarebbe fattibile se non bufferizzato [10].

Per quanto riguarda le diverse tecniche, si utilizzeranno parametri che forniscono risultati fino al negativo, ovvero che aumentano troppo l'errore, in modo da non tralasciare importanti particolari. In dettaglio:

- per la tecnica *Time sampling* saranno utilizzati campionamenti a 0,1 0,5 1 2 4 e 8 secondi
- per la tecnica *Space sampling* saranno utilizzati campionamenti a 1 10 20 40 80 e 100 metri
- per la tecnica *Deterministic information-need (versione sperimentale)* saranno utilizzati campionamenti a 0.01 0.05 0.1 1 e 10 Km/h con *history* di dimensione variabile tra 1 3 5 9 e 15
- per la tecnica *Deterministic information-need (versione offline)* saranno utilizzati campionamenti a 0.025 0.05 1 2 4 8 e 10 Km/h
- per la tecnica *Map-based (versione offline)* l'unico parametro variabile è l'utilizzo del cambio di segmento di strada (T) o meno (F)
- per la tecnica *Simple regression* saranno utilizzati campionamenti a 0,5 1 5 7 10 20 e 40 metri con *history* di dimensione variabile tra 3 5 7 10 20 e 40
- per la tecnica *Regression* saranno utilizzati campionamenti con *history* di dimensione variabile tra 3 4 5 6 7 8 10 12 14 15 20 e 40.

5.2.2 Parametri per l'analisi V2V

A differenza dell'analisi V2I (vedi sezione 5.3), dove le tecniche *communication-saving* presentavano numerose variabili e parametri impostabili (vedi sezione 5.2.1), per quanto concerne l'analisi V2V, gli unici parametri variabili riguardano lo scenario (vedi sezione 5.1), come nel caso V2I, il campionamento della simulazione di traffico, a 0,1 e 0,5 secondi e infine la portata dell'antenna WiFi.

A differenza della figura 5.17, dove il raggio di azione dell'antenna WiFi dell'OBU è influenzata dagli oggetti circostanti e di conseguenza assume diverse forme, in questa simulazione si considera il raggio di azione perfettamente circolare.

Come specificato dallo standard IEEE_802.11 [43] e riportato in [44], il para-

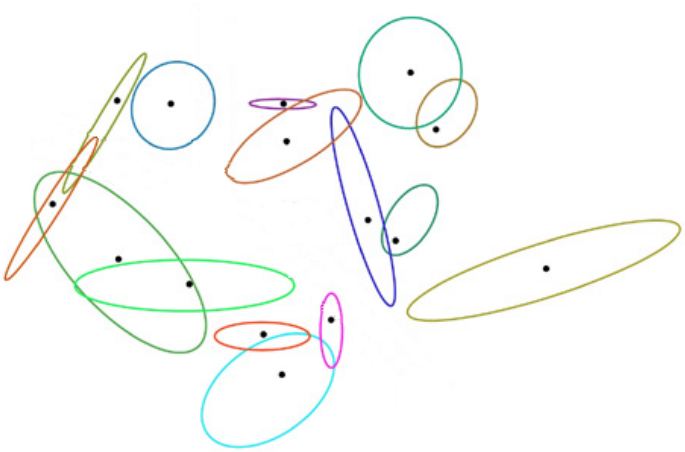

Figura 5.17: Esempio copertura antenna WiFi

metro che determina la portata dell'antenna WiFi sarà variato tra 100 120 140 e 250 metri, mentre i tempi di latenza, tra 1 e 3 millisecondi, non rappresentano un ostacolo e sono quindi ignorati.

Come per l'analisi V2I, per una maggiore comprensione, si ritiene opportuno descrivere sigle utilizzate per i nomi dei campi delle tabelle in ambito V2V:

- "scenario": nome dello scenario in cui si è svolta la simulazione (con riferimento alla sezione 5.1)
- "sim time (s)": tempo totale di simulazione (in secondi)
- "camp (s)": campionamento base (in secondi)
- "obu tot": numero totale di OBU coinvolte nella simulazione
- "grp len": lunghezza del gruppo, misurata in numero di OBU
- "grp changeXs": numero di cambiamenti nel gruppo per secondo, ovvero numero di veicoli che entrano od escono dal gruppo per ogni secondo
- "v in grp (s)": tempo medio di permanenza (in secondi) di un OBU all'interno di un gruppo (dato derivato da $grp\,len/grp\,changeXs$)
- "step life": tempo di vita totale delle OBU misurato in numero di step (dato derivato da $tot\,life\,(ms)/1000 * camp\,(s)$)
- "tot life (ms)": tempo di vita totale delle OBU (in millisecondi)
- "obu num": numero di OBU coinvolte nella simulazione

5.3 Analisi V2I

In questa sezione saranno riportate tutte le informazioni significative, sia sottoforma di grafico che di tabella, al fine di valutare le tecniche *communication-saving* sviluppate.

In particolare, nella sezione 5.3.1 sono riportate le stime preliminari effettuate e la dimensione teorica del flusso di dati che le OBU invieranno alla centrale; nella sezione 5.3.2 si analizza la tecnica *Simple time sampling*, mentre nella sezione 5.3.3 si analizza la tecnica *Simple space sampling*; nella sezione 5.3.4 si riportano le analisi sperimentali relative alle tecniche *Deterministic information-need*; nella sezione 5.3.5 si presentano le analisi sulla tecnica *Map-based sampling*, mentre nella sezione 5.3.6 si analizzano i dati ottenuti per la tecnica *Simple regression*; nella sezione 5.3.7 si analizzano i risultati ottenuti con la tecnica *Linear regression*, infine, nella sezione 5.3.8 si presentano le note e le deduzioni ottenute da queste analisi.

Si precisa inoltre che le prove sperimentali sono state eseguite sui diversi scenari della sezione 5.1 con durate differenti di simulazione, da un minuto a quattro ore, con i diversi parametri di cui alla sezione 5.2.1. Le tabelle riportate, per problemi di grafica, sono suddivise in parti (Es. parte 1 di 4, ... , parte 4 di 4), ma sono da intendersi come su una sola riga. Sempre per maggiore chiarezza, per campionamento base s'intende la distanza degli intervalli temporali durante la simulazione di traffico, mentre per campionamento si può intendere la scelta dei dati dell'OBU da inviare alla centrale fatta da una specifica tecnica.

5.3.1 Stime preliminari e dimensione flusso dati

Le tabelle presentate di seguito (5.2, 5.3, 5.4, 5.5, 5.6, 5.7) sono i risultati generali ottenuti dalla simulazione di traffico sullo scenario Roma (vedi sezione 5.1.2) e sono da leggersi, come già indicato, su una stessa riga.

I tempi di simulazione e il campionamento sono da leggersi nella prima colonna della prima tabella (5.2).

Tabella 5.2: Stime preliminari - parte 1 di 6

ROME	**hits**	**steps**	**data len**	**obu num**	**data x obu**	**tot dist**	**avg v frv**	**avg v int**
60 0,5	10057	10057	174043	174	1000,25	78550	9994	9996
60 0,1	44671	44671	792465	174	4554,4	66885	9435	9430
600 0,5	263095	263095	4689167	1739	2696,47	1867889	94208	94190
600 0,1	1310478	1310482	23356853	1739	13431,2	1868535	93911	93911
3600 0,5	1769579	1769579	32374325	10666	3035,28	12416358	571173	571164

Tabella 5.3: Stime preliminari - parte 2 di 6

c dist	**dist diff**	**c avg v**	**v diff**	**Dist err% ok**	Dist err% clc	Dist err% oktot	Dist err% clctot
80035	1485	10006	39	**1,8554%**	1,8905%	1,8554%	1,8905%
67177	299	9434	28	**0,4451%**	0,4470%	0,4347%	0,4366%
1881976	14133	94245	180	**0,7510%**	0,7566%	0,7485%	0,7542%
1870976	3013	93915	117	**0,1610%**	0,1612%	0,1305%	0,1306%
12501852	85785	571420	918	**0,6862%**	0,6909%	0,6839%	0,6886%

Tabella 5.4: Stime preliminari - parte 3 di 6

V err % ok	V err% clc	V err% oktot	V err% clctot	Dist err max	Dist rel max	Dist err% max
0,3898%	0,3902%	0,0999%	0,1000%	13,86	664	2,0873%
0,2968%	0,2969%	0,0424%	0,0424%	7,01	446	1,5717%
0,1910%	0,1911%	0,0584%	0,0584%	14,09	692	2,0361%
0,1246%	0,1246%	0,0043%	0,0043%	14,72	668	2,2036%
0,1607%	0,1607%	0,0448%	0,0448%	15,41	2338	0,6591%

Tabella 5.5: Stime preliminari - parte 4 di 6

Dist err med	Dist rel med	Dist err% med	Dist err min	Dist rel min	Dist err% min
9,42	461	2,0434%	0,33	27	1,2222%
1,81	463	0,3909%	0,01	258	0,0039%
8,98	803	1,1183%	0,13	35	0,3714%
1,47	793	0,1854%	0	2329	0,0000%
8,9	805	1,1056%	0,01	2320	0,0004%

Tabella 5.6: Stime preliminari - parte 5 di 6

V err max	V rel max	V err% max	V err med	V rel med	V err% med
1,48	70,34	2,1041%	0,11	66,74	0,1648%
0,58	31,99	1,8131%	0,19	33,46	0,5678%
2,15	69,6	3,0891%	0,06	60,1	0,0998%
1,41	68,73	2,0515%	0,05	59,55	0,0840%
1,59	73,2	2,1721%	0,06	58,86	0,1019%

Tabella 5.7: Stime preliminari - parte 6 di 6

V err min	V rel min	V err% min
0	79,65	0,0000%
0	41,78	0,0000%
0	77,35	0,0000%
0	73,48	0,0000%
0	76,8	0,0000%

Da questi dati è stato possibile ricavare i grafici seguenti:

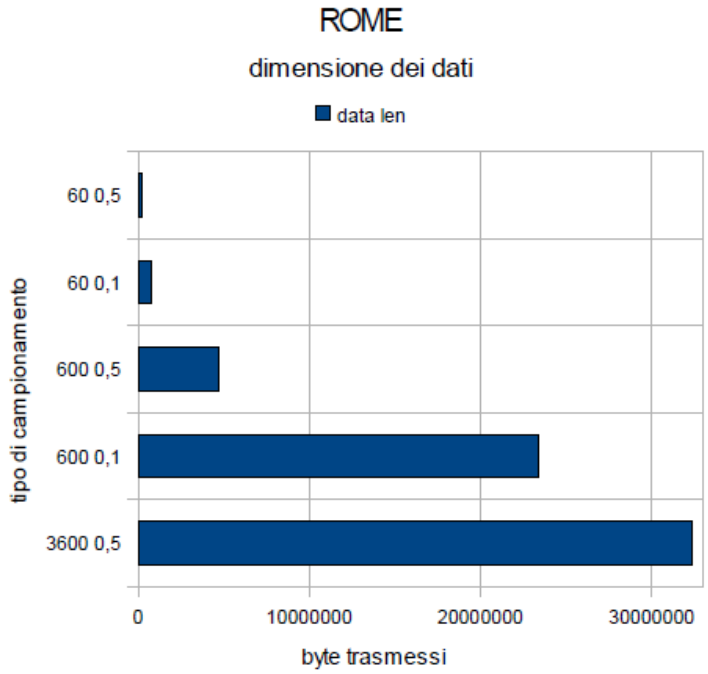

Figura 5.18: Grafico - Roma, dimensione dei dati

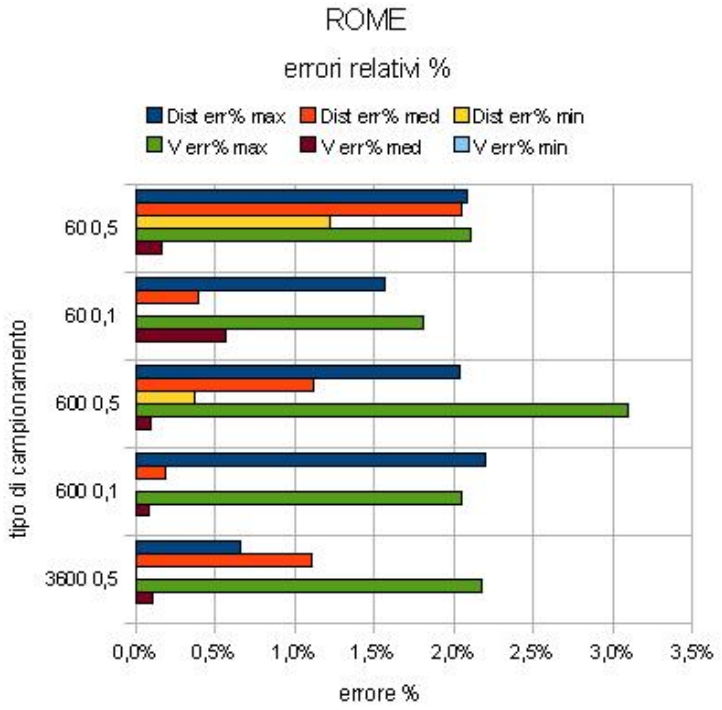

Figura 5.19: Grafico - Roma, errori relativi percentuali

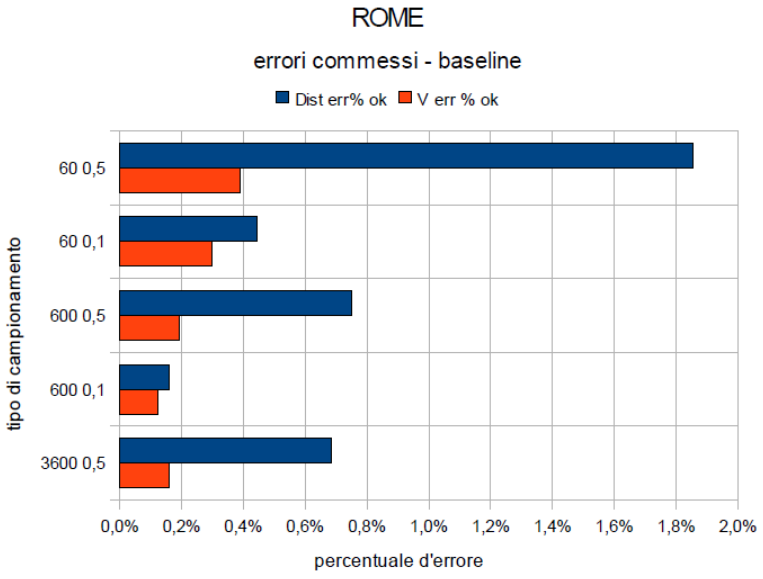

Figura 5.20: Grafico - Roma, errori commessi - baseline

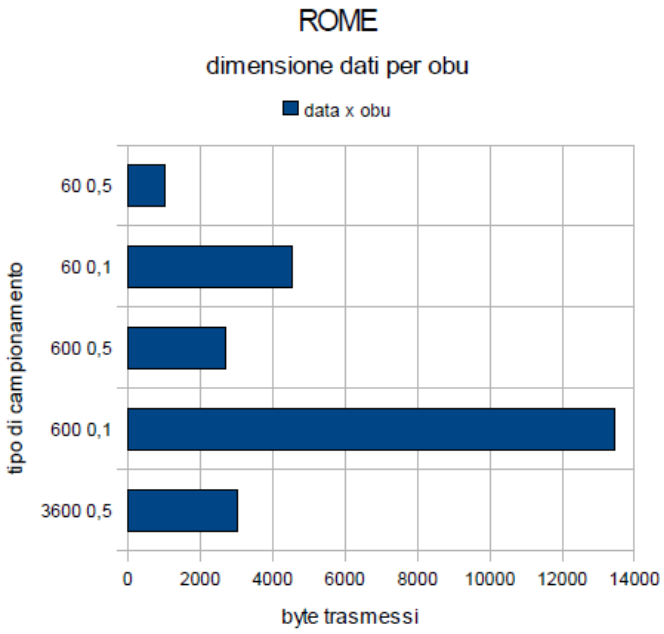

Figura 5.21: Grafico - Roma, dimensione dati per OBU

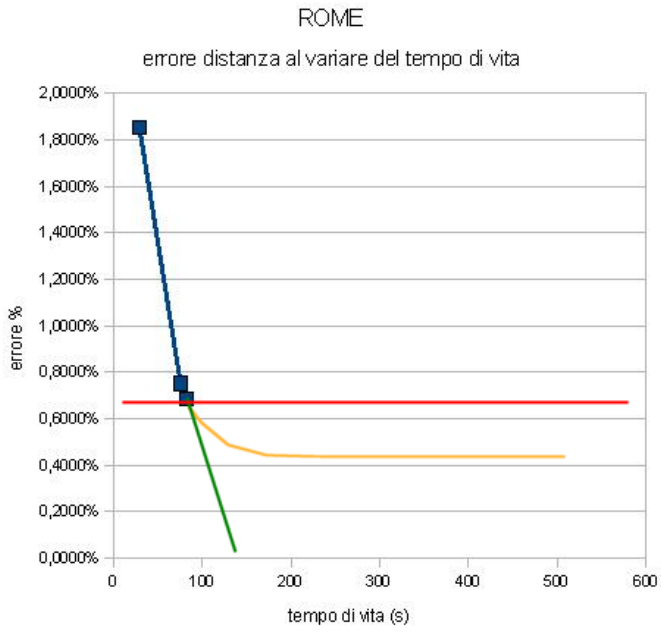

Figura 5.22: Grafico - Roma, errore distanza al variare del tempo di vita

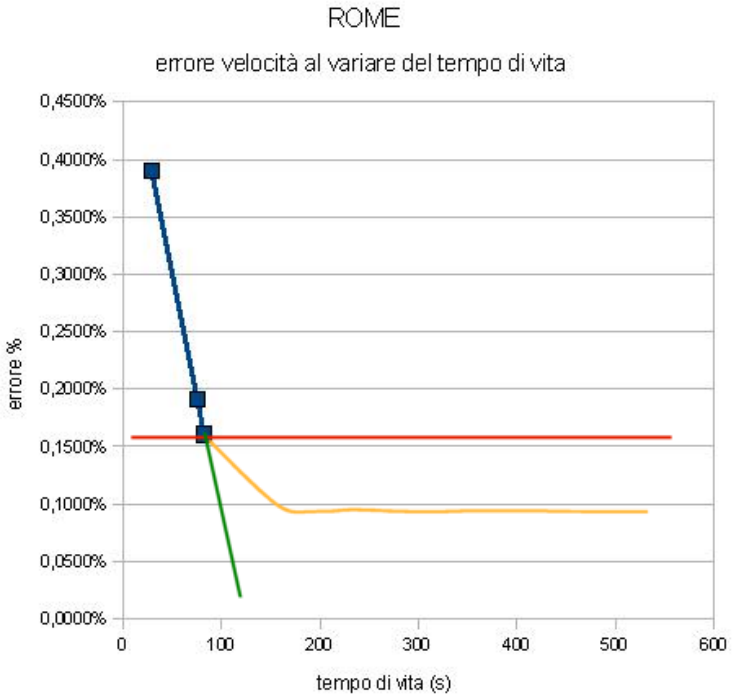

Figura 5.23: Grafico - Roma, errore velocità al variare del tempo di vita

Se i grafici 5.18 e 5.19 non evidenziano quasi nulla, a parte il totale dei dati scambiati nel primo caso, e il fatto che il valore mediano dell'errore % è molto più vicino al minimo che al massimo nel secondo caso, gli altri grafici fanno emergere utili, e spesso ovvie ma non provate, considerazioni.

In particolare, dal grafico 5.20 si deduce che, sia l'errore sulla distanza che sulla velocità:

1. diminuisce al diminuire del tempo di campionamento
2. diminuisce all'aumentare del tempo di simulazione
3. è inferiore all'1% quando il veicolo circola per più di circa 5 minuti.

Dal grafico 5.21 si deduce che la quantità di dati inviati dall'OBU alla centrale:

1. aumenta (linearmente) al diminuire del tempo di campionamento
2. aumenta all'aumentare del tempo di simulazione

3. è direttamente proporzionale all'errore commesso sia su distanza che su velocità.

Considerando il tempo di vita medio delle OBU, rispettivamente, in ordine di simulazione come in tabella, di 28,9 47,6 75,6 75,4 e 83,0 secondi, scartando i dati relativi al campionamento a 0,1 secondi, dato che mancano i dati sulla simulazione da 3600 secondi, si ottengono i grafici 5.22 e 5.23, rispettivamente indicanti l'errore di distanza e velocità al variare del tempo di vita dell'OBU (28,9 75,6 e 83,0 secondi).

Da questi due grafici quello che emerge è che all'aumentare del tempo di vita dell'OBU diminuisce l'errore in entrambi i casi.

La valutazione, puramente indicativa, fatta immettendo tre linee nel grafico, potrebbe evidenziare il comportamento dell'errore all'aumentare del tempo di vita dell'OBU, in caso ottimistico (massima pendenza), probabilistico (curva intermedia) e pessimistico (pendenza nulla).

Alcuni dati (vedi tabella 5.8) sullo scenario di Bologna (sezione 5.1.1), confermano le stesse tendenze, anche se gli errori sono leggermente superiori, forse dipendentemente dalla conformazione dello scenario, infatti, Roma (sezione 5.1.2) ha strade più dritte del centro di Bologna, come emerge dal confronto della figura 5.2 con la figura 5.9.

Tabella 5.8: Dati preliminari su Bologna

BOLOGNA	**avg life (s)**	**dist err%**	**v err%**
60 0,5	28	1,604%	0,560%
60 0,1	28	0,491%	1,688%
600 0,5	345,6	0,411%	0,461%
600 0,1	346,1	0,641%	1,486%

Note e considerazioni generali dall'analisi effettuata

Un campionamento ad intervalli più brevi possibili fornisce risultati migliori, infatti, l'errore su distanza e velocità, in dieci minuti di simulazione campionati a 0,1 secondi, è minore dello 0,2%, ma la quantità di dati trasmessa è circa cinque volte tanto quella del campionamento a 0,5 secondi, che fornisce comunque ottimi risultati, ovvero un errore inferiore allo 0,8%.

Dato che un campionamento a 0,1 secondi e il relativo invio di dati risulta difficile, sia per motivi legati all'architettura reale dell'OBU che non ha le performance di un PC moderno, e sia per motivi legati ai tempi di latenza del GPRS, la cui curva rispecchia una gaussiana centrata sui 500 millisecondi [14]; dato che l'errore diminuisce all'aumentare del tempo di vita dell'OBU ed è comunque inferiore allo 0,7% con un *life time* medio di 83 secondi, si conferma quanto indicato anche in 5.2.1, ovvero il tempo minimo tra una comunicazione e l'altra non può essere inferiore agli 0,5 secondi e si fissa questa soglia anche come parametro di confronto per le successive analisi V2I.

Una motivazione della deduzione riguardante il grafico 5.21, ovvero che la quantità di dati inviati dall'OBU alla centrale aumenta all'aumentare del tempo di simulazione, si può giustificare considerando che all'aumentare del tempo di simulazione aumenta anche il tempo di vita di alcune OBU e il relativo tragitto, oltre all'aumentare dello stesso tempo. Questo determina un maggior quantitativo di cifre per rappresentare questi dati, che a loro volta determinano una maggiore lunghezza dei messaggi. A conferma di tale supposizione, oltre ai file di *log* che asseriscono ciò, c'è l'esponenziale e non lineare diminuzione di questo fenomeno all'aumentare della durata della simulazione, visibile nei campionamenti a 0,5 secondi.

Volendo ottenere stime riguardanti il traffico di dati totale delle OBU verso la centrale, considerandone circa un milione [9], che inviano dati campionati a 0,5 secondi ogni 0,5 secondi, come nello scenario Roma di durata 3600 secondi, ovvero ogni OBU invia 3035,28 byte all'ora, si ottiene un flusso di dati di circa 0,8 MB/s pari a circa 2,83 GB all'ora e 68 GB al giorno. Per concludere, considerando queste stime molto realistiche, considerando che non solo questo tipo di informazioni devono essere inviate dall'OBU e considerando il numero crescente di quest'ultime, si può affermare con certezza che le tecniche *communication-saving* sono o diverranno presto una necessità.

5.3.2 Time sampling

Nell'analisi di questa tecnica si sono utilizzati diversi scenari con diversi tempi di simulazione, in particolare Roma con durate di 600 e 3600 secondi, Beijing con durata di 1200 secondi, Toll Plaza con durata di 3600 secondi e infine Bologna in tre modalità: normale da 600 e 3600 secondi, RID ovvero con numero di veicoli RIDotto e percorsi circolari, della durata di 7200 secondi e ZTL, ovvero con numero di veicoli ulteriormente ridotto rispetto RID ed eliminazione delle Zone a Traffico Limitato, della durata di 14400 secondi.

Roma, simulazione di 600 secondi

Tabella 5.9: Time sampling Roma 600 - Parte 1 di 2

campion	hits	steps	data len	tot dist	avg v frv	avg v int	c dist
0,5	263095	263095	4689167	1867889	94208	94190	1881976
1	131994	263095	2352420	1860407	94186	94078	1881976
2	66423	263095	1183638	1844501	94149	93837	1881976
4	33639	263095	599499	1811104	94045	93245	1881976
8	17258	263095	307606	1744755	93808	91624	1881976

Tabella 5.10: Time sampling Roma 600 - Parte 2 di 2

dist diff	life	c avg v	v diff (int)	**Dist err% ok**	**V err % ok**	**obu num**	**data x obu**
14133	131548	94245	180	**0,7510%**	**0,1910%**	1739	2696,47
21586	131548	94245	265	**1,1470%**	**0,2812%**	1739	1352,74
37475	131548	94245	509	**1,9913%**	**0,5401%**	1739	680,64
70872	131548	94245	1209	**3,7658%**	**1,2828%**	1739	344,74
137221	131548	94245	3205	**7,2913%**	**3,4007%**	1739	176,89

Come intuibile, anche dai grafici 5.24 e 5.25, relativi alle tabelle 5.9 e 5.10, si può vedere come, all'aumentare del tempo intercorso tra un invio e un altro, aumenti anche l'errore associato sia per la distanza che per la velocità. Parallelamente si può anche notare il *trade-off* dati inviati - errore commesso, infatti, se da una parte si ha il peggioramento delle stime dall'altro si ha un risparmio di comunicazione.

Da notare che l'errore commesso sulla distanza è inferiore allo 0,76% e quello sulla velocità è allo 0,2% con un *time sampling* di 0,5 secondi (base), mentre con un *time sampling* di 2 secondi (quattro volte tanto) si hanno errori rispettivamente inferiori al 2% e allo 0,55%.

La crescita dell'errore non è lineare, infatti al raddoppiare del tempo di campionamento, rispettivamente, l'errore segue un andamento con fattore moltipli-

cativo di 1,53 1,74 1,89 e 1,94, cioè segue un andamento iperbolico, come visibile dal grafico 5.24.

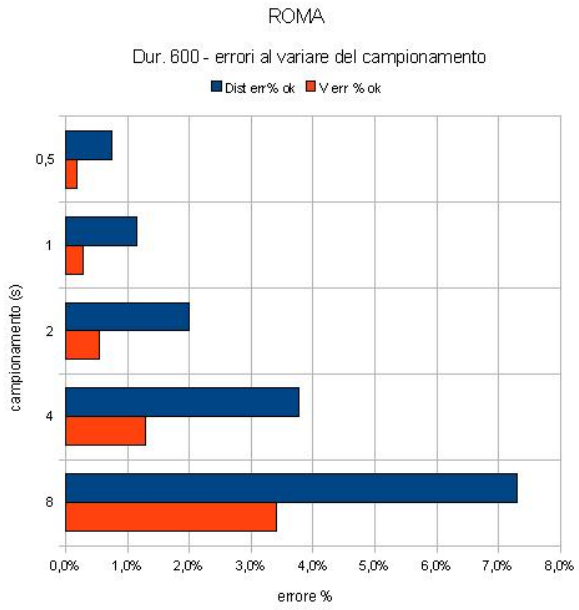

Figura 5.24: Grafico - time sampling Roma 600, errori commessi

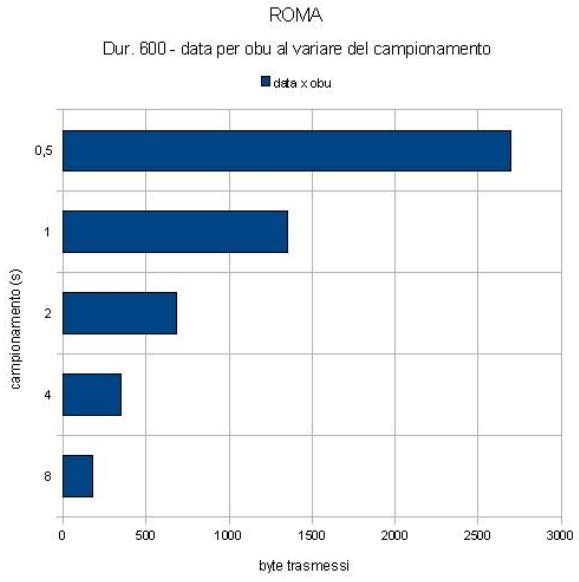

Figura 5.25: Grafico - time sampling Roma 600, dimensione dati per OBU

Bologna, simulazione di 600 secondi

Questa simulazione, i cui dati sono esigui, è da intendersi più come test che come analisi della tecnica *time sampling*, infatti, si ci aspettava da entrambe le prove un risultato pressoché identico.

Analizzando i grafici 5.26 e 5.27, relativi alle tabelle 5.11 e 5.12, si deduce che il test è andato a buon fine, infatti, le differenze sugli errori sono inferiori allo 0,05%, così come quelle sulla dimensione dei dati, ovvero praticamente nulle.

Si precisa che il risultato non poteva essere identico con errori dello 0% in quanto il diverso campionamento base (da 0,1 e 0,5 secondi) influenza anche il tempo di vita delle OBU, come visibile dalla tabella 5.12 colonna "life".
Si riporta un esempio per maggiore chiarezza:
Un OBU, nel caso di campionamento base a 0,1 secondi potrebbe comparire nella simulazione al tempo 10.3 e scomparire al tempo 20.1, ovvero avere un tempo di vita di $(20.1 - 10.3 =)$ *9.8 secondi.*
Nel caso di campionamento base a 0,5 secondi, la stessa OBU comparirebbe, per arrotondamento, a 10.5 secondi e scomparirebbe a 20.0 secondi, ovvero avrebbe un tempo di vita di $(20.0 - 10.5 =)$ *9.5 secondi, cioè 0.3 secondi in meno.*
Questo influenza anche la distanza percorsa, infatti, l'OBU con durata di vita maggiore percorre più strada dell'altra OBU.

Altra considerazione riguarda il numero di OBU che, dipendentemente dalle fasi finali nella quale un OBU può avere un tempo di vita di pochi decimi di secondo, varia tra i due tipi di campionamento. In questo caso è stato addirittura inserito un filtro di 10 secondi sulla durata di vita, ma questo non è servito ad appianare questa differenza, bensì a scartare risultati insignificanti, infatti, un OBU con durata di 9.8 secondi verrebbe scartata dal filtro, mentre la stessa OBU, campionata a 0,5 secondi, che avrebbe durata di 10.0 secondi, non lo sarebbe.

Tabella 5.11: Time sampling Bologna 600 - Parte 1 di 2

campion	hits	steps	data len	tot dist	avg v frv	avg v int	c dist
0,5s on 0.5	538494	2691161	10933227	1550371,05	19121,01	19202,27	1544805
0,5s on 0,1	536373	536373	10902803	1548424,99	18961,55	19051,86	1543971

Tabella 5.12: Time sampling Bologna 600 - Parte 2 di 2

dist diff	life	c avg v	v diff (int)	**Dist err% ok**	**V err % ok**	obu num	**data x obu**
6842,4	269116	18964,7	73,82	**0,4429%**	**0,3893%**	780	**14016**
6327,05	268186,5	18868,68	84,44	**0,4098%**	**0,4475%**	777	**14031**

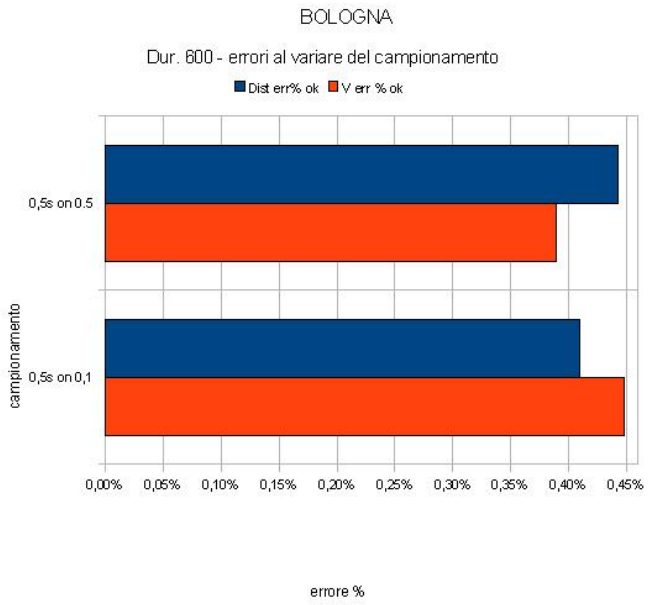

Figura 5.26: Grafico - time sampling Bologna 600, errori commessi

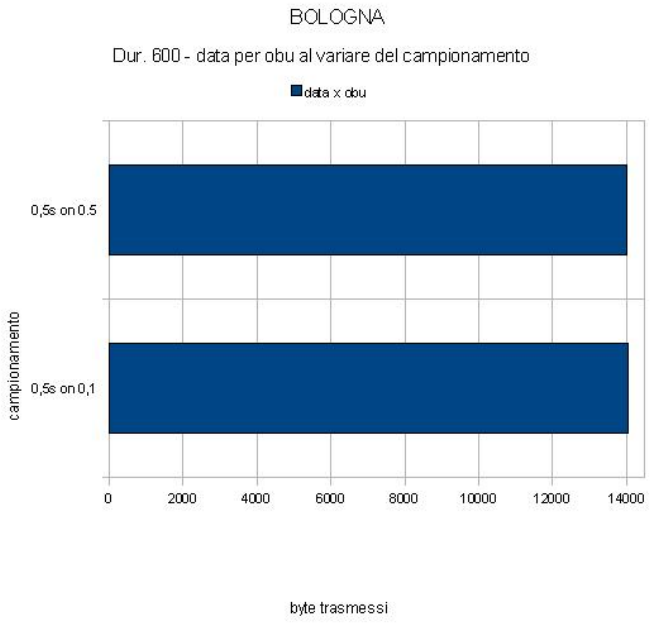

Figura 5.27: Grafico - time sampling Bologna 600, dimensione dati per OBU

Roma, simulazione di 3600 secondi

Lo scenario in questione è Roma (vedi sezione 5.1.2), con durata della simulazione di traffico di 2 ore ed intervallo di campionamento base a 0,5 secondi.

Dalle tabelle 5.13 e 5.14 e dai grafici 5.28 e 5.29 ad esse relativi, si può dedurre che, come nella sezione 5.3.2, all'aumentare del tempo intercorso tra un invio e un altro, aumenti anche l'errore associato, sia per la distanza che per la velocità. Parallelamente si può anche notare il *trade-off* dati inviati - errore commesso, infatti, se da una parte si ha il peggioramento delle stime dall'altro si ha un risparmio di comunicazione.

Tabella 5.13: Time sampling Roma 3600 - Parte 1 di 2

campion	hits	steps	data len	tot dist	avg v frv	avg v int	c dist
0,5s	1769579	1769579	32374325	12416358	571173	571164	12501852
1s	887466	1769578	16235670	12368911,33	571018,05	570692,64	12501841
2s	446363	1769578	8165176	12270306,55	570698,49	569730,39	12501841
4s	225813	1769578	4131011	12064434,57	569906,65	567965,38	12501841
8s	115726	1769578	2116490	11666764,74	568567,16	563485,08	12501841

Tabella 5.14: Time sampling Roma 3600 - Parte 2 di 2

dist diff	life	c avg v	v diff (int)	**Dist err% ok**	**V err % ok**	obu num	**data x obu**
85785	884790	571459	878	**0,6862%**	**0,1536%**	10666	**3035,28**
133021,14	884789	571458,84	1197,5	**1,0640%**	**0,2096%**	10666	**1522,19**
231534,91	884789	571458,84	2179,11	**1,8520%**	**0,3813%**	10666	**765,53**
437406,43	884789	571458,84	4541,31	**3,4987%**	**0,7947%**	10666	**387,31**
835076,26	884789	571458,84	10881,09	**6,6796%**	**1,9041%**	10666	**198,43**

Da notare che l'errore commesso è diminuito in tutti i casi rispetto all'analisi della sezione 5.3.2, come visibile in tabella comparativa 5.15.

Tabella 5.15: Tabella comparativa, errori Roma 600 e 3600

3600	**600**			**3600**	**600**	
Dist err% ok	**Dist err% ok**	**Diff**		**V err % ok**	**V err % ok**	**Diff**
0,6862%	0,7510%	0,0648%		0,1536%	0,1910%	0,0374%
1,0640%	1,1470%	0,0830%		0,2096%	0,2812%	0,0716%
1,8520%	1,9913%	0,1393%		0,3813%	0,5401%	0,1588%
3,4987%	3,7658%	0,2671%		0,7947%	1,2828%	0,4881%
6,6796%	7,2913%	0,6117%		1,9041%	3,4007%	1,4966%

Ciò conferma ulteriormente l'analisi preliminare della sezione 5.3.1 nella quale si asserisce che all'aumentare della vita dell'OBU (in questo caso 75,6 secondi per la simulazione da 600 e 83 secondi per quella da 3600) diminuisce, o al peggio rimane invariato, l'errore commesso, sia sulla velocità che sulla distanza.

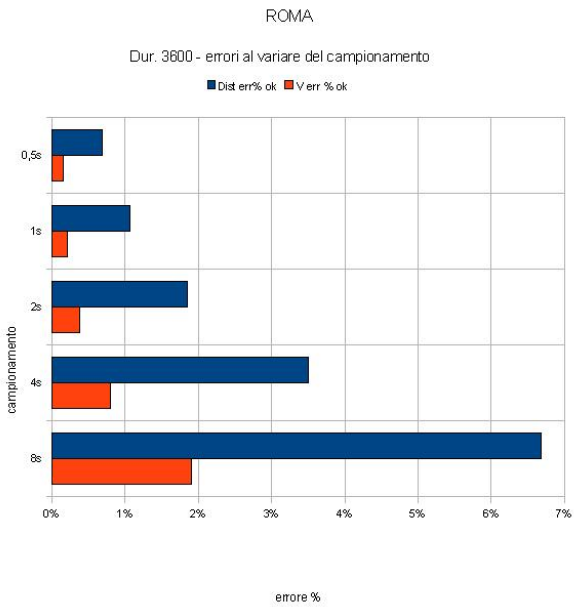

Figura 5.28: Grafico - time sampling Roma 3600, errori commessi

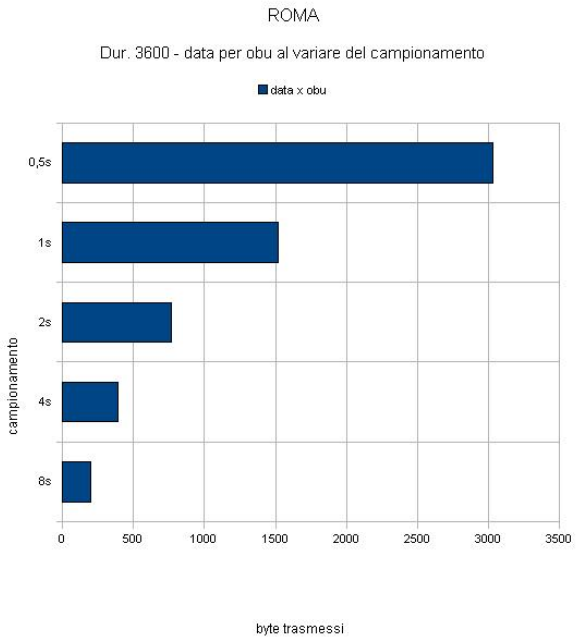

Figura 5.29: Grafico - time sampling Roma 3600, dimensione dati per OBU

Beijing, simulazione di 1200 secondi

Come nel caso precedente (vedi sezione 5.3.2), data la notevole quantità di veicoli, il campionamento base è di 0,5 secondi.

Dalle tabelle 5.16 e 5.17 e dai grafici 5.30 e 5.31 ad esse relativi, si può dedurre che, come nelle sezioni 5.3.2 e 5.3.2, all'aumentare del tempo intercorso tra un invio e un altro, aumenti anche l'errore associato, sia per la distanza che per la velocità. Parallelamente si può anche notare il *trade-off* dati inviati - errore commesso, infatti, se da una parte si ha il peggioramento delle stime dall'altro si ha un risparmio di comunicazione.

Un altro dato da notare in questa particolare simulazione è l'aumento del tasso di errore rispetto ai due scenari precedenti, che passa da percentuali inferiori allo 0,76% per la distanza e 0,2% per la velocità a percentuali dell'1,05% e dello 0,53%. Da notare però che, anche se il tempo medio di vita aumenta (circa 102 secondi), la distanza media percorsa (339,4 metri) e la velocità diminuiscono (15,94 Km/h) rispetto le simulazioni precedenti (Es. Roma 3600, tempo di vita medio 83 secondi, distanza media percorsa 1172,12 metri, velocità media 53.58 Km/h).

Considerando i valori percentuali degli errori applicati ai dati reali, pur avendo tassi di errore leggermente più alti delle simulazioni precedenti, questo significa in pratica sbagliare di 3,5 metri su 339,4 e 0,06 Km/h su 15,94. Dato che difficilmente un veicolo percorrerà così poca strada e comunque l'errore è irrisorio, si afferma che anche in questo caso i risultati sono molto soddisfacenti.

Tabella 5.16: Time sampling Beijing 1200 - Parte 1 di 2

campion	hits	steps	data len	tot dist	avg v frv	avg v int	c dist
0,5s	1168863	1168863	19972551	1923792,75	90938,01	90968,48	1943945
1s	585842	1168863	10011438	1911508,45	90753,73	90667,11	1943945
2s	294341	1168863	5030962	1887947,49	90365,93	90084,59	1943945
4s	148604	1168863	2541191	1842811,48	89548,44	89070,1	1943945
8s	75681	1168863	1295157	1753252,94	87771,46	86905,33	1943945

Tabella 5.17: Time sampling Beijing 1200 - Parte 1 di 2

dist diff	life	c avg v	v diff (int)	**Dist err% ok**	**V err % ok**	obu num	**data x obu**
20341,32	584431,5	91299,76	479,48	**1,0464%**	**0,5252%**	5728	**3486**
32491,78	584431,5	91299,76	753	**1,6714%**	**0,8248%**	5728	**1747**
56004,37	584431,5	91299,76	1345,44	**2,8810%**	**1,4736%**	5728	**878**
101135,92	584431,5	91299,76	2472,2	**5,2026%**	**2,7078%**	5728	**443**
190692,06	584431,5	91299,76	4936,18	**9,8095%**	**5,4066%**	5728	**226**

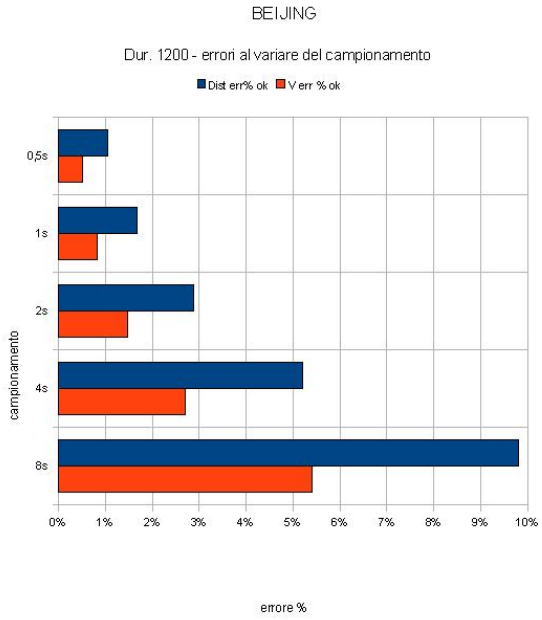

Figura 5.30: Grafico - time sampling Beijing 1200, errori commessi

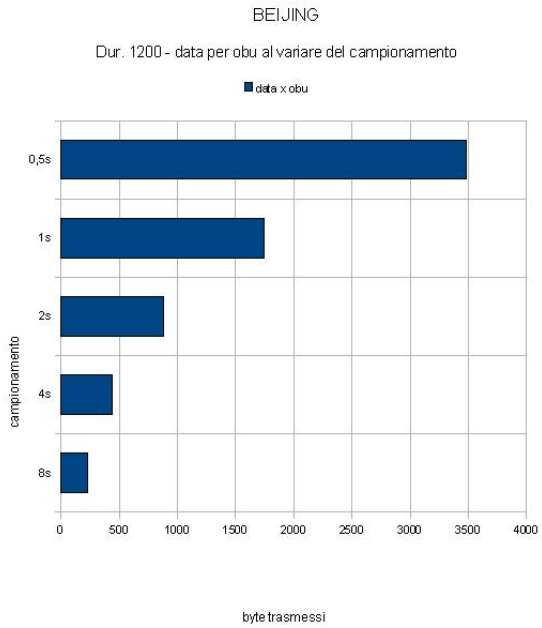

Figura 5.31: Grafico - time sampling Beijing 1200, dimensione dati per OBU

Toll Plaza, simulazione di 3600 secondi

In questo scenario, che ha un campionamento base a 0,5 secondi, non sono stati voluti analizzare ulteriori parametri al di fuori di 0,5 (base) e 1 secondi, dato che questi bastavano al confronto diretto con altre tecniche e fornivano già sufficienti prove per le affermazioni seguenti.
Confermando quanto riportato in sezione 5.3.2, in questo scenario migliorano sia gli errori relativi alla velocità che alla distanza, dati i tempi più lunghi sia di vita che di simulazione, oltre alla maggiore distanza coperta dalle OBU.

In particolare, come visibile e/o deducibile dalle tabelle 5.18 e 5.19, i cui grafici sono riportati nella sezione 5.3.7:

- l'errore su campionamento base della distanza è dello 0,44% mentre quello sulla velocità è dello 0,27%
- la vita media dell'OBU è 153,7 secondi
- la distanza effettiva media percorsa è di 1822.4 metri, mentre la distanza media percorsa, interpolata dai dati inviati, è di 1814.5 metri
- la velocità effettiva media è di 43.39 Km/h, mentre la velocità media, interpolata dai dati inviati, è di 43.34 Km/h

Le differenze tra dati reali e quelli calcolati via interpolazione dai dati inviati risultano mediamente di 7,9 metri e 0,05 Km/h.
Tassi di errore così bassi si prestano a tutti i tipi di servizi richiesti nei requisiti, soprattuto se si considera che il tempo di vita reale di un veicolo sarà con molta probabilità superiore ai 153,7 secondi (circa 2 minuti) ottenuti da questa simulazione.

Tabella 5.18: Time sampling Toll Plaza 3600 - Parte 1 di 2

campion	hits	steps	data len	tot dist	avg v frv	avg v int	c dist
0,5s	2428861	2428861	42726752	14338142,43	342447,65	342441,77	14400721
1s	1216379	2428861	21398435	14299353,86	342204,98	341910,71	14400721

Tabella 5.19: Time sampling Toll Plaza 3600 - Parte 2 di 2

dist diff	life	c avg v	v diff (int)	**Dist err% ok**	**V err % ok**	obu num	**data x obu**
63257,01	1214430,5	342906,13	909,13	**0,4393%**	**0,2651%**	7902	**5407**
101509,99	1214430,5	342906,13	1250,05	**0,7049%**	**0,3645%**	7902	**2707**

Bologna (RID) simulazione di 7200 secondi

In questo scenario, preso da Bologna (vedi sezione 5.1.1) e ridotto (RID) soprattuto in termini di numero di veicoli, che ha un campionamento base a 0,5 secondi, sono stati analizzati i parametri per la tecnica *simple time sampling* come nella sezione 5.3.2, 5.3.2 e 5.3.2. Confermando quanto riportato nelle sezioni appena citate, in questo scenario si attesta che gli errori relativi alla velocità e alla distanza sono abbastanza contenuti e crescono al crescere del campionamento, sempre con il relativo *trade-off* con la quantità di dati scambiati. In questo caso però, i tempi più lunghi sia di vita che di simulazione, oltre alla maggiore distanza coperta dalle OBU, non migliorano le stime sull'errore rispetto allo scenario simile, ma non eguale, della sezione 5.3.2.

In particolare, come visibile e/o deducibile dalle tabelle 5.20 e 5.21, i cui grafici relativi sono in figura 5.32 e 5.33, l'errore su campionamento base della distanza è dello 0,54% mentre quello sulla velocità è dello 0,53%[2]; la vita media dell'OBU è 3433,8 secondi; la distanza effettiva media percorsa è di 28205.3 metri, mentre la distanza media percorsa, interpolata dai dati inviati, è di 28357.3 metri; infine, la velocità effettiva media è di 29.6 Km/h (o 29.2 Km/h), mentre la velocità media, interpolata dai dati inviati, è di 29.625 Km/h.

Le differenze tra dati reali e quelli calcolati via interpolazione dai dati inviati risultano mediamente di 152 metri e 0,025 Km/h. Tassi di errore così bassi si prestano a tutti i tipi di servizi richiesti nei requisiti.

Tabella 5.20: Time sampling Bologna (RID) 7200 - Parte 1 di 2

campion	hits	steps	data len	tot dist	avg v frv	avg v int	c dist
0,5s	940851	940851	19694170	3884955,21	4035,66	4058,64	3864127
1s	470458	940851	9847666	3864645,36	4034,28	4037,19	3864127
2s	235261	940851	4924450	3812962,16	4033,19	3984,36	3864127
4s	117666	940851	2463030	3702906,37	4031,41	3872,95	3864127
8s	58868	940851	1232220	3482312,83	4041,42	3640,68	3864127

Tabella 5.21: Time sampling Bologna (RID) 7200 - Parte 2 di 2

dist diff	life	c avg v	v diff (int)	**Dist err% ok**	**V err % ok**	obu num	**data x obu**
20868,91	470425,5	3998,91	21,15	**0,5401%**	**0,5289%**	137	**143753**
36013,88	470425,5	4038,11	37,08	**0,9320%**	**0,9183%**	137	**71880**
51164,84	470425,5	4038,11	53,75	**1,3241%**	**1,3311%**	137	**35944**
161220,63	470425,5	4038,11	165,16	**4,1722%**	**4,0900%**	137	**17978**
381814,17	470425,5	4038,11	397,42	**9,8810%**	**9,8417%**	137	**8994**

[2]Un leggera discrepanza tra dati nella colonna "c avg v" in tabella 5.21, dovuta a cause non note, porterebbe, se cambiata, ad un errore inferiore pari allo 0,5084%

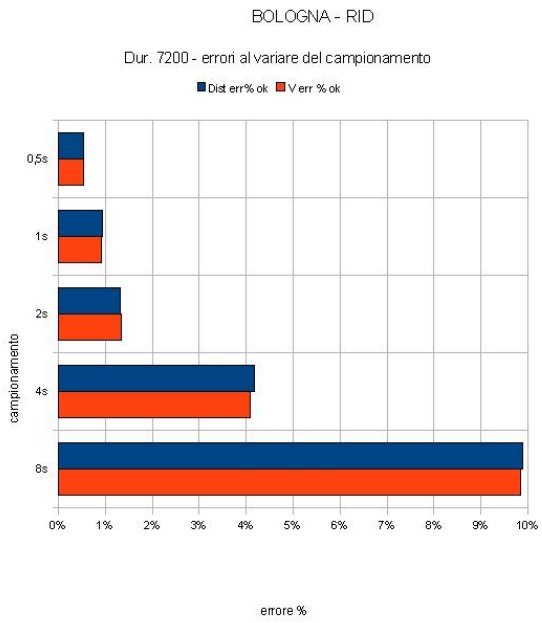

Figura 5.32: Grafico - time sampling Bologna (RID) 7200, errori commessi

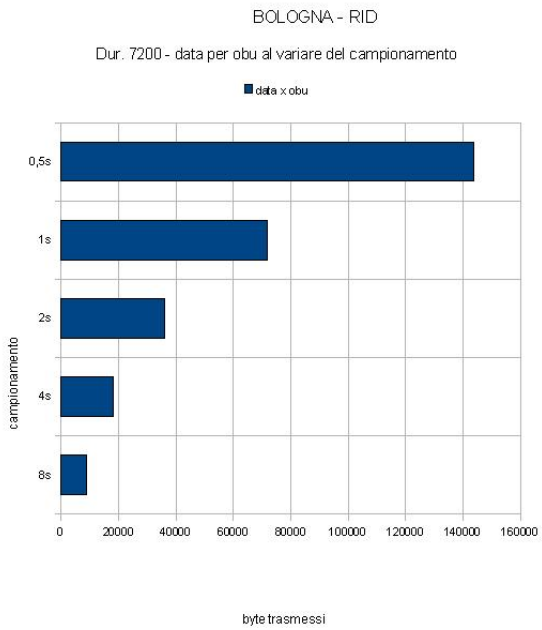

Figura 5.33: Grafico - time sampling Bologna (RID) 7200, dim. dati per OBU

Bologna (ZTL) simulazione di 14400 secondi

In questo scenario, preso da Bologna (vedi sezione 5.1.1) e ridotto sia escludendo le zone a traffico limitato dalla mappa (ZTL) sia riducendo il numero di veicoli, sono stati analizzati alcuni parametri su campionamenti base a 0,1 e 0,5 secondi per la tecnica *simple time sampling* in stile sezioni 5.3.2, 5.3.2, 5.3.2 e 5.3.2. Confermando quanto riportato nelle sezioni appena citate, in questo scenario si attesta che gli errori relativi alla velocità e alla distanza sono abbastanza contenuti e crescono al crescere del campionamento, sempre con il relativo *trade-off* con la quantità di dati scambiati.

Anche in questo caso, i tempi più lunghi sia di vita che di simulazione, oltre alla maggiore distanza coperta dalle OBU, migliorano notevolmente le stime sull'errore rispetto agli scenari simili, ma non eguali, delle sezioni 5.3.2 e 5.3.2, come sostenuto in sezione 5.3.1. La differenza principale riguarda il tempo di campionamento che ha ottenuto i risultati migliori, ovvero quello a 1 secondo sia per campionamento base a 0,5 che a 0,1 secondi invece del campionamento minimo a 0,5 secondi, per cause non note.

In particolare, come visibile e/o deducibile dalle tabelle 5.22, 5.23, 5.24 e 5.25 i cui grafici relativi sono in figura 5.34, 5.35, 5.36 e 5.37:

- l'errore minimo commesso (campionamento 1s on 0,1) sulla distanza è dello 0,073% mentre quello sulla velocità è dello 0,074%
- la vita media dell'OBU, in entrambi i campionamenti base, è 2 ore 35 minuti e 53 secondi (discrepanza di 0,8 secondi tra i due)
- la distanza effettiva media percorsa è di 79916 metri (on 0,1) e 79390 metri (on 0,5), mentre la distanza media percorsa, interpolata dai dati inviati, è rispettivamente di 79961 e 79483 metri
- la velocità effettiva media è di 30.96 Km/h (on 0,1) e 30.754 Km/h (on 0,5), mentre la velocità media, interpolata dai dati inviati, è rispettivamente di 30.98 Km/h e 30.79 Km/h.

Le differenze tra dati reali e quelli calcolati via interpolazione dai dati inviati risultano mediamente di 69 metri e 0,028 Km/h. Tassi di errore così bassi si prestano a tutti i tipi di servizi richiesti nei requisiti, considerando soprattutto che il tragitto percorso è stato di circa 80 Km.

Campionamento base a 0,5 secondi:

Tabella 5.22: Time sampling Bologna (ZTL) 14400 (0,5)- Parte 1 di 2

campion	hits	steps	data len	tot dist	avg v frv	avg v int	c dist
0,5s	947904	947904	20065820	4067417,62	1566,47	1573,7	4048885
1s	473966	947904	10033241	4053636,6	1566,45	1568,41	4048885
1,5s	315986	947904	6688989	4036150,07	1566,48	1561,7	4048885
2s	236995	947904	5016756	4016597,63	1566,2	1554,18	4048885

Tabella 5.23: Time sampling Bologna (ZTL) 14400 (0,5)- Parte 2 di 2

dist diff	life	c avg v	v diff (int)	**Dist err% ok**	**V err % ok**	obu num	**data x obu**
18532,62	473952	1566,47	7,23	**0,4577%**	**0,4613%**	51	**393447**
4807,26	473952	1566,47	1,96	**0,1187%**	**0,1248%**	51	**196730**
13717,36	473952	1566,47	5,18	**0,3388%**	**0,3307%**	51	**131156**
32287,37	473952	1566,47	12,29	**0,7974%**	**0,7848%**	51	**98367**

Campionamento base a 0,1 secondi:

Tabella 5.24: Time sampling Bologna (ZTL) 14400 (0,1)- Parte 1 di 2

campion	hits	steps	data len	tot dist	avg v frv	avg v int	c dist
0,5s on 0,1	947881	4739322	20069503	4092149,43	1577,67	1584,06	4075707
1s on 0,1	473954	4739322	10035024	4078011,71	1577,67	1578,65	4075707
2s on 0,1	236990	4739322	5017823	4040149,6	1577,74	1564,1	4075707
4s on 0,1	118510	4739322	2508958	3957679,97	1577,22	1532,25	4075707

Tabella 5.25: Time sampling Bologna (ZTL) 14400 (0,1)- Parte 2 di 2

dist diff	life	c avg v	v diff (int)	**Dist err% ok**	**V err % ok**	obu num	**data x obu**
16442,43	473932,1	1577,67	6,39	**0,4034%**	**0,4052%**	51	**393519**
2971,96	473932,1	1577,67	1,17	**0,0729%**	**0,0739%**	51	**196765**
35557,4	473932,1	1577,67	13,57	**0,8724%**	**0,8600%**	51	**98388**
118027,03	473932,1	1577,67	45,42	**2,8959%**	**2,8788%**	51	**49195**

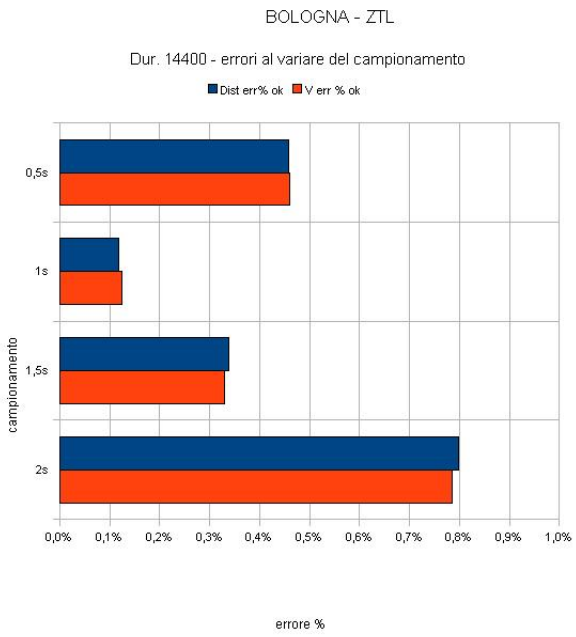

Figura 5.34: Grafico - time sampling Bologna 14400 (ztl 0,5), errori commessi

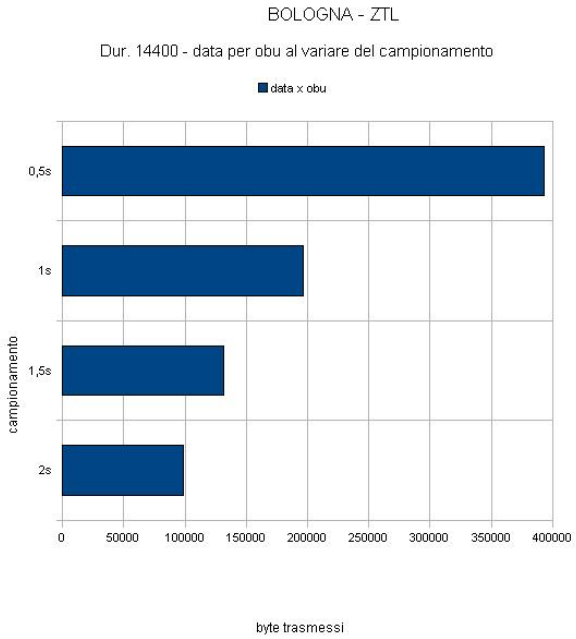

Figura 5.35: Grafico - time sampl. Bologna 14400 (ztl 0,5), dim. dati per OBU

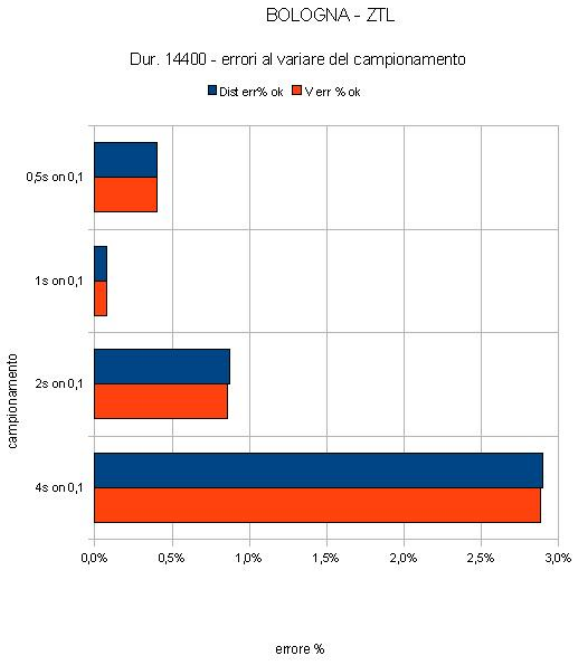

Figura 5.36: Grafico - time sampling Bologna 14400 (ztl 0,1), errori commessi

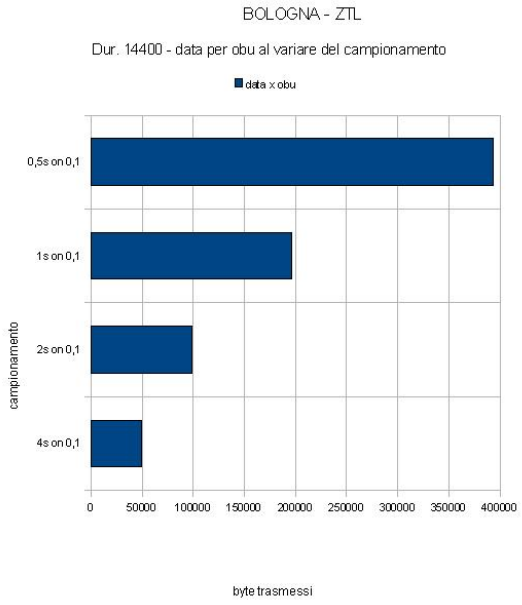

Figura 5.37: Grafico - time sampl. Bologna 14400 (ztl 0,1), dim. dati per OBU

Note e considerazioni generali dall'analisi effettuata

Da tutte le analisi effettuate sulla tecnica *simple time sampling*, sia su scenari con campionamento base a 0,1 secondi che con campionamento base a 0,5 secondi, nella quasi totalità dei casi, all'aumentare del tempo intercorso tra un invio e un altro, aumenta anche l'errore associato sia per la distanza che per la velocità. Parallelamente si può anche notare il *trade-off* dati inviati - errore commesso, infatti, se da una parte si ha il peggioramento delle stime dall'altro si ha un risparmio di comunicazione.

La crescita dell'errore, relativo sia a distanza che velocità, non segue un andamento lineare, infatti al raddoppiare del tempo di campionamento, rispettivamente, l'errore segue un andamento iperbolico, particolarmente visibile nel grafico 5.24 e in misura minore in tutti gli altri grafici dello stesso tipo riportati in questa sezione.

I tassi di errore, risultati praticamente sempre inferiori all'1% nel campionamento a 0,5 secondi, e mediamente intorno allo 0,5%, quindi particolarmente bassi, si prestano a tutti i tipi di servizi richiesti nei requisiti.

Da considerare che non sempre il risultato migliore, ottenuto in termini di errore, è pervenuto da campionamento minimo a 0,5 secondi, infatti, nella sezione 5.3.7 il campionamento che ha dato risultati migliori è stato quello da 1 secondo.

In linea con gli obiettivi proposti, spesso non sarà possibile inviare dati con campionamento a 0,5 secondi, sia per questioni di latenza che di banda. Di conseguenza o si adotteranno strategie di *buffering*, impattando solo sulla banda, oppure si dovrà spesso scegliere un campionamento a frequenza inferiore, come ad esempio 1, 2 o 4 secondi, aumentando il tasso di errore, ma riducendo sia la banda sia la frequenza trasmissiva.

5.3.3 Space sampling

In questa sezione si riportano le prove sperimentali fatte sullo scenario di Roma (vedi sezione 5.1.2) con tempo di simulazione di 600 secondi relative alla tecnica *simple space sampling* la cui implementazione è visibile alla sezione 4.2.2.

Per definire gli aspetti di questo tipo di tecnica non è stato necessario riportare ulteriori prove su altri scenari, in quanto gli ulteriori dati ottenuti sono mediamente conformi a questi.

Come visibile e/o deducibile dalle tabelle 5.26 e 5.27, i cui grafici relativi sono in figura 5.38 e 5.39, la prima differenza con le tecniche viste nelle sezioni precedenti, con particolare riferimento alla sezione 5.3.2 tecnica *simple time sampling*, riguarda gli alti errori sulla velocità media interpolata dai dati inviati.

Tabella 5.26: Space sampling Roma 600- Parte 1 di 2

campion	hits	steps	data len	tot dist	avg v frv	avg v int	c dist
0,5s	263095	263095	4689167	1867889	94208	94190	1881976
1m	245287	263095	4398183	1867849	96325	96291	1881976
10m	132108	263095	2365454	1860954	99529	99248	1881976
20m	77029	263095	1379157	1849356	99880	99231	1881976
40m	42688	263095	764114	1822082	100208	98910	1881976
80m	22719	263095	406630	1765521	100439	97713	1881976
100m	18315	263095	327961	1727116	100376	96839	1881976

Tabella 5.27: Space sampling Roma 600- Parte 2 di 2

dist diff	life	c avg v	v diff (int)	**Dist err% ok**	**V err % ok**	**obu num**	**data x obu**
14133	131548	94245	180	**0,7510%**	**0,1910%**	1739	2696,47
14173	131548	94245	2223	**0,7531%**	**2,3587%**	1739	2529,14
21031	131548	94245	5216	**1,1175%**	**5,5345%**	1739	1360,24
32620	131548	94245	5534	**1,7333%**	**5,8719%**	1739	793,07
59894	131548	94245	5661	**3,1825%**	**6,0067%**	1739	439,4
116455	131548	94245	6296	**6,1879%**	**6,6805%**	1739	233,83
154860	131548	94245	6939	**8,2286%**	**7,3627%**	1739	188,59

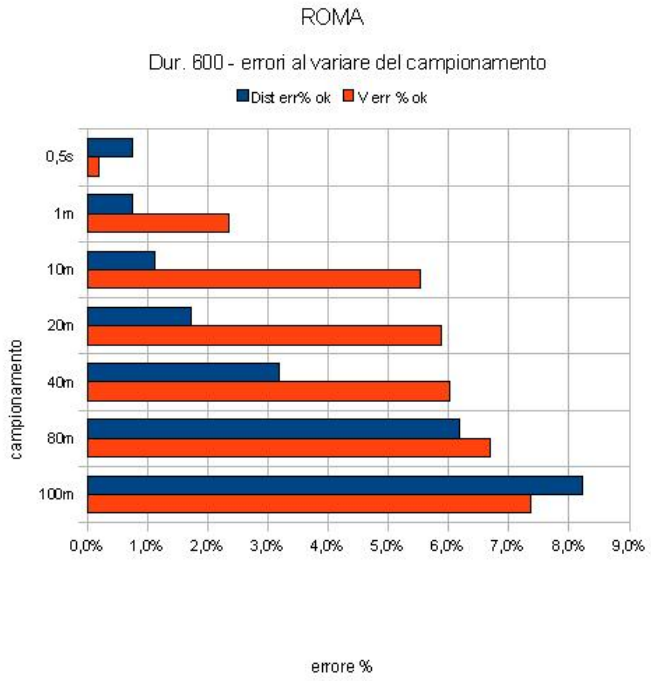

Figura 5.38: Grafico - space sampling Roma 600, errori commessi

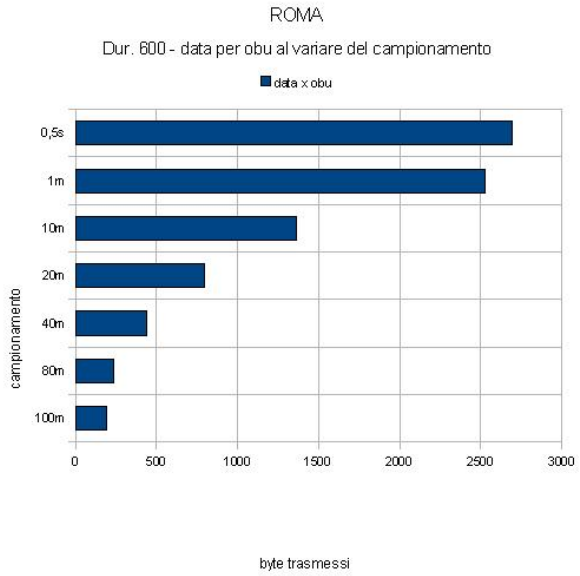

Figura 5.39: Grafico - space sampling Roma 600, dimensione dati per OBU

Note e considerazioni generali dall'analisi effettuata

Dai risultati ottenuti con l'analisi di questa tecnica, emerge una differenza accentuata con la tecnica *simple time sampling* della sezione 5.3.2, infatti, se in quest'ultima gli errori relativi a distanza e velocità erano sempre simili, per quanto riguarda il campionamento effettuato con *simple space sampling* questo non risulta vero. Dai grafici riportati si può vedere chiaramente che nei campionamenti più fini l'errore sulla velocità è notevolmente superiore a quello sulla distanza, mentre man mano si cresce nell'aumentare la distanza di campionamento, anche l'errore sulla velocità si conforma a quello sulla distanza.

Le ragioni di questo fenomeno vanno ricercate principalmente nella dipendenza tra velocità e frequenza di trasmissione. A differenza della tecnica *simple time sampling*, dove le informazioni sono inviate ad intervalli regolari di tempo, in questo caso le informazioni sono inviate ad intervalli non regolari, infatti, al crescere della velocità cresce la frequenza trasmissiva dato che aumenta lo spazio percorso nello stesso tempo, come descritto dalla formula $\delta V = \delta s/\delta t$. Considerando poi che l'intervallo minimo di campionamento del tempo Δt è pari all'intervallo di campionamento base, in questo caso 0,5 secondi, si deduce anche il perché, pur il veicolo viaggiando ad una velocità media di circa 14 m/s cioè 7 metri ogni 0,5 secondi, la frequenza di trasmissione non sia anch'essa 7 volte superiore al campionamento temporale di 0,5 secondi.

Un considerazione puramente arbitraria in merito all'alto tasso di errore sulla velocità, potrebbe riguardare la bassa frequenza trasmissiva impostata dalla tecnica nei punti critici a bassa velocità, quali ad esempio le curve, che richiedono maggiore precisione a livello interpolativo.

In definitiva, da quanto dedotto, la tecnica *simple space sampling* dovrebbe essere utilizzata per ridurre i continui campionamenti della tecnica *simple time sampling* quando il veicolo viaggia a velocità particolarmente ridotte, come ad esempio nelle code del traffico cittadino, facendo risparmiare sia sulla banda che sul numero di comunicazioni.

5.3.4 Deterministic information-need

Come riportato nella sezione 4.2.2, di questa tecnica ne sono state implementate tre versioni e potute analizzare due: la prima ("versione sperimentale con history") verifica se la media delle n velocità trascorse si discosta più di una certa tolleranza dalla velocità attuale, mentre la seconda ("versione off-line con e senza filtro") richiede al simulatore la velocità teorica nel tratto in percorrenza; Tutte e tre le versioni si basano su differenze di velocità in quanto esse determinano un cambiamento nello stato del veicolo, ad esempio code, semafori, curve, etcetera.

Versione sperimentale con history

Il concetto chiave di questa tecnica è la variazione, ovvero i cambiamenti di velocità nel veicolo, la cui misura è decisa dai due parametri considerati, la tolleranza sulla velocità e la media delle h velocità precedenti, ovvero la dimensione della *history*.

Le tabelle 5.28 e 5.29 ed i grafici 5.40 e 5.41 relativi, mostrano il comportamento dell'errore e della dimensione dei dati al variare dei parametri.

Tabella 5.28: Det. information-need (sperimentale) Roma 600 - Parte 1 di 2

campion	hits	steps	data len	tot dist	avg v frv	avg v int	c dist
0,5s	263095	263095	4689167	1867889	94208	94190	1881976
0.01km/h-h15	241914	263095	4336183	1866732	95869	95818	1881976
0.05km/h-h15	228013	263095	4086937	1866286	95750	95619	1881976
0.1km/h-h1	211004	263095	3782383	1861716	95536	95172	1881976
0.1km/h-h3	216104	263095	3873641	1865570	95664	95524	1881976
0.1km/h-h9	216927	263095	3888397	1865747	95683	95545	1881976
0.1km/h-h15	216935	263095	3888543	1865754	95683	95545	1881976
1km/h-h9	69882	263095	1255063	1783188	91362	88989	1881976
10km/h-h9	6018	263095	108102	712348	86997	39310	1881976

Tabella 5.29: Det. information-need (sperimentale) Roma 600 - Parte 2 di 2

dist diff	life	c avg v	v diff (int)	**Dist err% ok**	**V err % ok**	**obu num**	**data x obu**
14133	131548	94245	180	**0,7510%**	**0,1910%**	1739	2696,47
15289	114007	99244	3496	**0,8124%**	**3,5226%**	1739	2493,49
15734	107318	107318	11741	**0,8360%**	**10,9404%**	1739	2350,17
20304	105502	126921	31784	**1,0789%**	**25,0423%**	1739	2175,03
16449	108052	117103	21614	**0,8740%**	**18,4573%**	1739	2227,51
16273	108464	116157	20647	**0,8647%**	**17,7751%**	1739	2236
16266	108468	116146	20636	**0,8643%**	**17,7673%**	1739	2236,08
98788	34941	635482	546493	**5,2492%**	**85,9966%**	1739	721,72
1169628	3009	7427828	7388517	**62,1489%**	**99,4708%**	1739	62,16

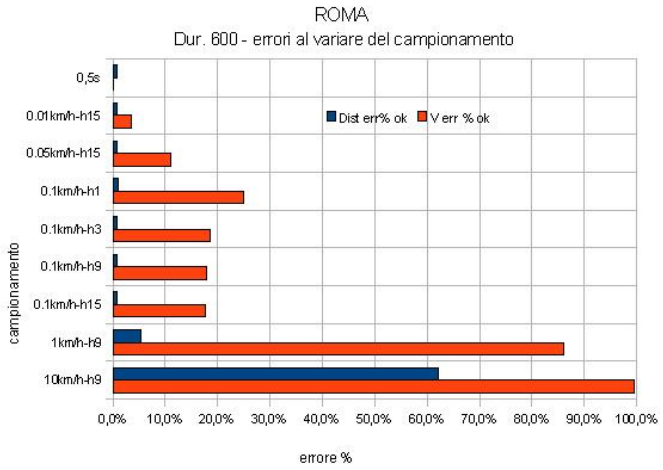

Figura 5.40: Grafico - det. info-need (exp) Roma 600, errori commessi

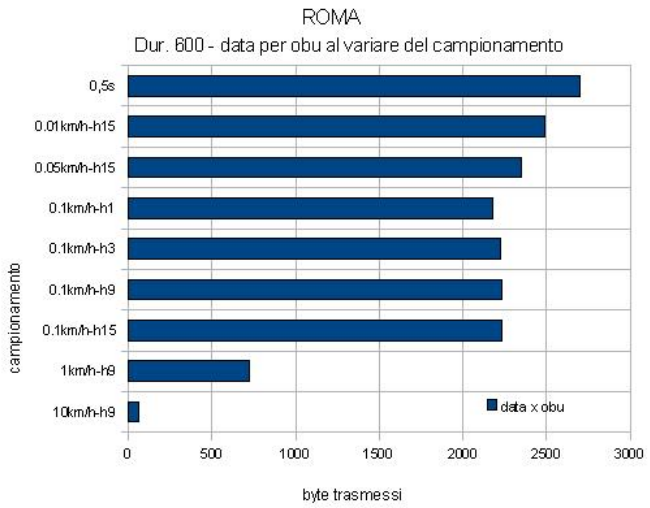

Figura 5.41: Grafico - det. info-need (exp) Roma 600, dim. dati per OBU

Versione off-line con e senza filtro

A differenza della prima implementazione analizzata, il cui concetto chiave di variazione rimane immutato, il campionamento è deciso da un solo parametro, ovvero la tolleranza sulla velocità, cioè quanto si discosta al massimo la velocità effettiva del veicolo da quella segnalata sulla mappa. Da qui, emerge la stretta dipendenza con le mappe fornite sull'architettura reale dell'OBU, le quali potrebbero presentare notevoli differenze con quelle analizzate.

Un ulteriore confronto fatto ha riguardato l'utilizzo di un filtro sui 60 secondi (f60) nel secondo caso, che ha permesso di scartare tutte le OBU non significative, cioè che non inviavano dati conseguentemente al loro basso tempo di vita.

Le tabelle 5.30 e 5.31 ed i grafici 5.42 e 5.43 relativi, mostrano il comportamento dell'errore e della dimensione dei dati al variare dei parametri in modalità senza filtro.

Le tabelle 5.32 e 5.33 ed i grafici 5.44 e 5.45 relativi, mostrano il comportamento dell'errore e della dimensione dei dati al variare dei parametri in modalità con filtro.

Tabella 5.30: Det. info-need (off-line senza filtro) Roma 600 - Parte 1 di 2

campion	hits	steps	data len	tot dist	avg v frv	avg v int	c dist
0,5s	263095	263095	4689167	1867889	94208	94190	1881976
0.025Km/h	258718	263095	4611654	1866331	94102	94083	1881976
0.05Km/h	228373	263095	4068949	1850731	93263	93242	1881976
1Km/h	199567	263095	3553908	1823460	92231	92208	1881976
2Km/h	143737	263095	2555187	1645591	87016	86599	1881976
4Km/h	97759	263095	1732080	833665	64124	63176	1881976
8Km/h	65434	263095	1158898	585004	50427	48755	1881976
10Km/h	54774	263095	966882	478114	43939	42317	1881976

Tabella 5.31: Det. info-need (off-line senza filtro) Roma 600 - Parte 2 di 2

dist diff	life	c avg v	v diff (int)	**Dist err% ok**	**V err % ok**	**obu num**	**data x obu**
14133	131548	94245	180	**0,7510%**	**0,1910%**	1739	2696,47
15690	131548	94245	309	**0,8337%**	**0,3279%**	1739	2651,9
31248	131548	94245	1185	**1,6604%**	**1,2574%**	1739	2339,82
58516	131548	94245	2257	**3,1093%**	**2,3948%**	1739	2043,65
236385	131548	94245	8036	**12,5605%**	**8,5267%**	1739	1469,34
1048311	131548	94245	31409	**55,7027%**	**33,3270%**	1739	996,02
1296972	131548	94245	45814	**68,9154%**	**48,6116%**	1739	666,42
1403862	131548	94245	52232	**74,5951%**	**55,4215%**	1739	556

Tabella 5.32: Det. info-need (off-line con filtro) Roma 600 - Parte 1 di 2

campion	hits	steps	data len	tot dist	avg v frv	avg v int	c dist
0,5s	263095	263095	4689167	1867889	94208	94190	1881976
0.025Km/h	258718	263095	4611654	1866331	94102	94083	1881976
0.05Km/h	228373	263095	4068949	1850731	93263	93242	1881976
1Km/h	199567	263095	3553908	1823460	92231	92208	1881976
2Km/h	143737	263095	2555187	1645591	87016	86599	1881976
4Km/h	97759	263095	1732080	833665	64124	63176	1881976
8Km/h	65434	263095	1158898	585004	50427	48755	1881976
10Km/h	54774	263095	966882	478114	43939	42317	1881976

Tabella 5.33: Det. info-need (off-line con filtro) Roma 600 - Parte 2 di 2

dist diff	life	c avg v	v diff (int)	**Dist err% ok**	**V err % ok**	**obu num**	**data x obu**
14133	131548	94245	180	**0,7510%**	**0,1910%**	1739	2696,47
15642	131548	94245	291	**0,8311%**	**0,3088%**	1739	2651,9
31051	131548	94245	1084	**1,6499%**	**1,1502%**	1739	2339,82
58250	131548	94245	2086	**3,0952%**	**2,2134%**	1739	2043,65
208623	131548	94245	5704	**11,0853%**	**6,0523%**	1739	1469,34
721823	131548	94245	10814	**38,3545%**	**11,4743%**	1739	996,02
789917	131548	94245	13717	**41,9727%**	**14,5546%**	1739	666,42
797250	131548	94245	14803	**42,3624%**	**15,7069%**	1739	556

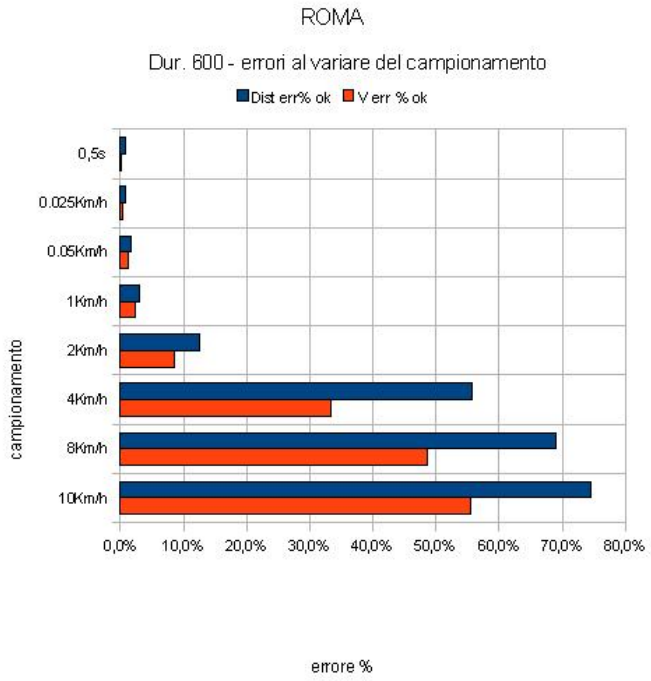

Figura 5.42: Grafico - det. info-need (off-line) Roma 600, errori commessi sf

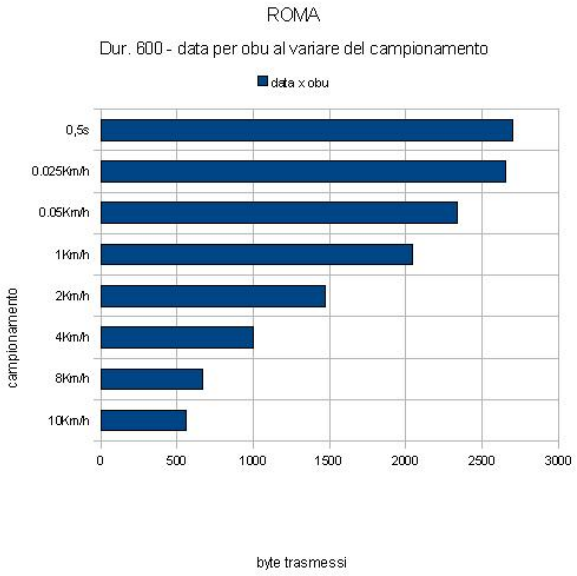

Figura 5.43: Grafico - det. info-need (off-line) Roma 600, dim. dati per OBU sf

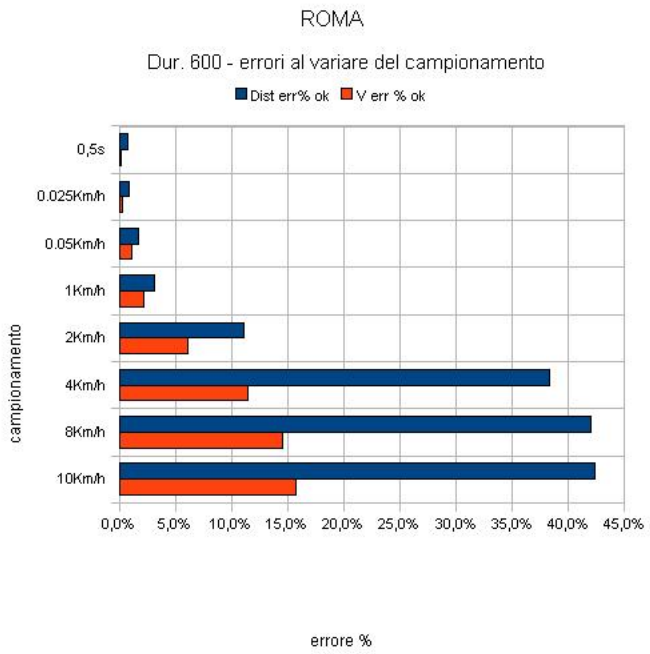

Figura 5.44: Grafico - det. info-need (off-line) Roma 600, errori commessi cf

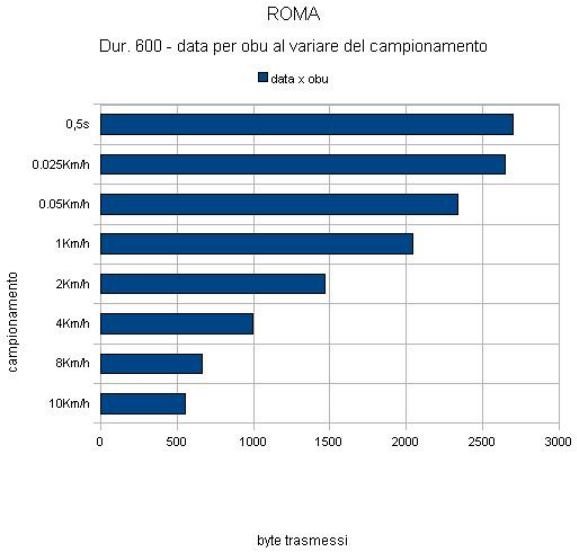

Figura 5.45: Grafico - det. info-need (off-line) Roma 600, dim. dati per OBU cf

Note e considerazioni generali dall'analisi effettuata

Per quanto riguarda la "Versione sperimentale con history", dai grafici e dalle tabelle relative si può dedurre che la dimensione della *history* non influenza particolarmente la tecnica, mentre la tolleranza sulla velocità si.

Gli errori sulla distanza risultano in linea con le tecniche precedenti analizzate, mentre gli errori sulla velocità risultano molto più marcati.

L'utilizzo di questa tecnica è da scartare per i servizi che richiedono precisione, ma potrebbe risultare utile per individuare i punti di rallentamento del veicolo (stop, code a tratti, etc.), anche se a tal scopo la versione *off-line* risulta migliore sotto gli aspetti di precisione e carico computazionale.

Per quanto riguarda le versioni *off-line* la prima nota è relativa alla modalità senza filtro, che riporta valori condizionati dai veicoli che compaiono verso la fine della simulazione e hanno tempo di vita troppo basso per una valida analisi. Per questo le considerazioni seguenti sono fatte sulla base della tecnica *off-line* con filtro sui 60 secondi.

Dai grafici 5.44 e 5.45 e dalle relative tabelle 5.32 e 5.33, si può dedurre che la tecnica in questione segue l'andamento di massima della tecnica *simple time sampling*, presentando però una percentuale di errore più alta in considerazione alla quantità di dati inviati.

L'utilizzo di questa tecnica è da ritenersi fattibile per individuare i punti di rallentamento del veicolo (stop, code a tratti, etc.), oppure da affiancare via *and* e *or* ad altre tecniche *communication-saving*.

5.3.5 Map-based sampling

In questa sezione si riportano le prove sperimentali fatte sullo scenario di Roma (vedi sezione 5.1.2) con tempo di simulazione di 600 secondi relative alla tecnica *Map-based sampling* la cui implementazione è visibile alla sezione 4.2.2.

Per definire gli aspetti di questo tipo di tecnica non è stato necessario riportare ulteriori prove su altri scenari, in quanto gli ulteriori dati ottenuti sono mediamente conformi a questi.

Come visibile e/o deducibile dalle tabelle 5.34 e 5.35, i cui grafici relativi sono in figura 5.46 e 5.47, la prima differenza con le tecniche viste nelle sezioni precedenti, riguarda la ridotta scelta dei parametri di campionamento, ovvero due.

Tabella 5.34: Map-based (off-line) Roma 600 - Parte 1 di 2

campion	hits	steps	data len	tot dist	avg v frv	avg v int	c dist
0,5s	263095	263095	4689167	1867889	94208	94190	1881976
map F	3135	263095	56143	1317562	98820	75231	1881976
map T	21327	263095	380149	1697080	95849	95012	1881976

Tabella 5.35: Map-based (off-line) Roma 600 - Parte 2 di 2

dist diff	life	c avg v	v diff (int)	**Dist err% ok**	**V err % ok**	**obu num**	**data x obu**
14133	131548	94245	180	**0,7510%**	**0,1910%**	1739	2696,47
564414	131548	94245	21424	**29,9905%**	**22,7322%**	1739	32,28
184896	131548	94245	5968	**9,8246%**	**6,3324%**	1739	218,6

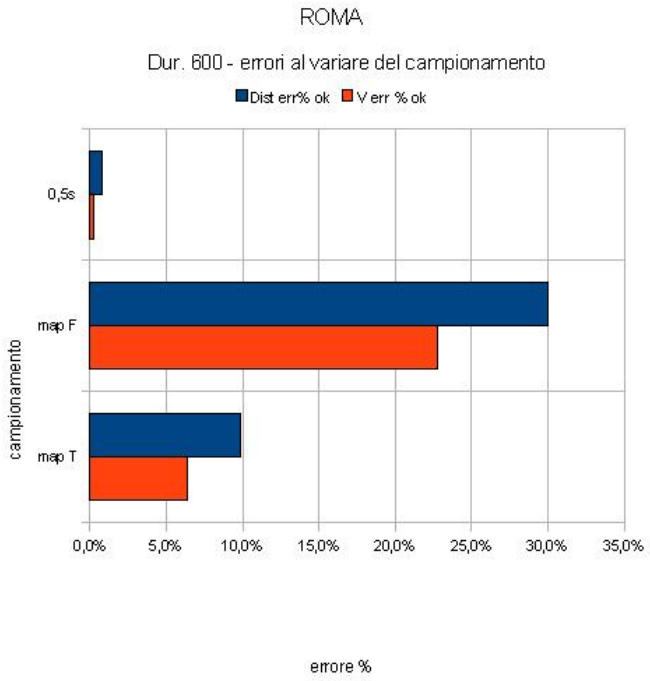

Figura 5.46: Grafico - Map-based (off-line) Roma 600, errori commessi

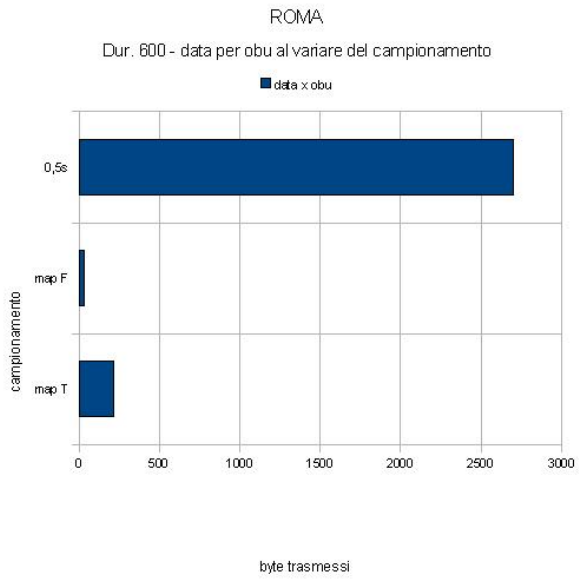

Figura 5.47: Grafico - Map-based (off-line) Roma 600, dim. dati per OBU

Note e considerazioni generali dall'analisi effettuata

Confrontando i risultati ottenuti con quelli delle tecniche precedenti, si nota che l'errore commesso, in entrambi i casi, è decisamente superiore.

Prendendo come riferimento i risultati ottenuti dalla tecnica *simple time sampling* con tempo di campionamento a 8 secondi e confrontandoli con quelli ottenuti dalla tecnica corrente, con variazione dei segmenti attiva (map T), emerge che gli errori sono decisamente minori a parità di dati inviati.

L'utilizzo migliore che si potrebbe fare di questa tecnica riguarderebbe la ricostruzione dell'itinerario percorso, infatti, se la centrale possedesse la stessa mappa presente sull'OBU, quest'ultima potrebbe ricostruire il percorso esatto effettuato con una quantità di dati irrisoria ed un errore sulla distanza minimo.

5.3.6 Simple Regression

Nell'analisi di questa tecnica si sono utilizzati due diversi scenari con diversi tempi di simulazione, ovvero Roma 5.1.2 con durate di 600 e 3600 secondi.

Nella simulazione da 600 secondi sono riportate due analisi: la prima che prende in considerazione tutti i dati e la seconda che deriva della prima, ma prende in considerazione solo i dati delle OBU che hanno durata di vita tra 160 e 240 secondi, dove 240 secondi è una delle massime durate di vita nella simulazione.

Roma, simulazione di 600 secondi, con e senza filtro

Le tabelle 5.36 e 5.37 con i relativi grafici 5.48 e 5.49 mostrano i risultati senza applicazione di filtri.

Tabella 5.36: Simple regression (senza filtro) Roma 600 - Parte 1 di 2

campion	hits	steps	data len	tot dist	avg v frv	avg v int	c dist
0,5s	263095	263095	4689167	1867889	94208	94190	1881976
0,5m_h3	61164	263095	1818944	1791845	93365	93322	1881016
1m_h5	40454	263095	1203648	1766169	92726	92679	1881016
1m_h10	20227	263095	600924	1707144	92135	92038	1879970
5m_h10	22357	263095	664948	1761735	91775	91767	1880481
5m_h20	10961	263095	325895	1655575	90164	90144	1877914
10m_h5	42836	263095	1272568	1828590	93250	93250	1881016
10m_h10	22802	263095	677795	1781291	91831	91842	1880481
10m_h20	11343	263095	337495	1689080	89809	89831	1878155
10m_h40	4971	263095	147767	1447509	84411	84583	1852263
20m_h20	11516	263095	342662	1698568	89398	89447	1878155
20m_h40	5336	263095	159064	1531172	84777	84945	1858073
40m_h5	42842	263095	1272752	1828625	93248	93248	1881016

Tabella 5.37: Simple regression (senza filtro) Roma 600 - Parte 2 di 2

dist diff	life	c avg v	v diff (int)	**Dist err% ok**	**V err % ok**	**obu num**	**data x obu**
14133	131548	94245	180	**0,7510%**	**0,1910%**	1739	2696,47
89630	131548	93351	1249	**4,7650%**	**1,3380%**	1739	1045,97
114847	131548	93351	1214	**6,1056%**	**1,3005%**	1739	692,15
172826	131548	92758	2265	**9,1930%**	**2,4418%**	1739	345,56
118746	131548	92988	1825	**6,3147%**	**1,9626%**	1739	382,37
222339	131548	91943	3576	**11,8397%**	**3,8894%**	1739	187,4
53468	131548	93351	1509	**2,8425%**	**1,6165%**	1739	731,78
99400	131548	92988	1794	**5,2859%**	**1,9293%**	1739	389,76
189075	131548	92010	3309	**10,0671%**	**3,5963%**	1739	194,07
404754	131548	87994	5539	**21,8519%**	**6,2947%**	1739	84,97
179587	131548	92010	3481	**9,5619%**	**3,7833%**	1739	197,05
326901	131548	88607	5604	**17,5935%**	**6,3246%**	1739	91,47
53432	131548	93351	1511	**2,8406%**	**1,6186%**	1739	731,89

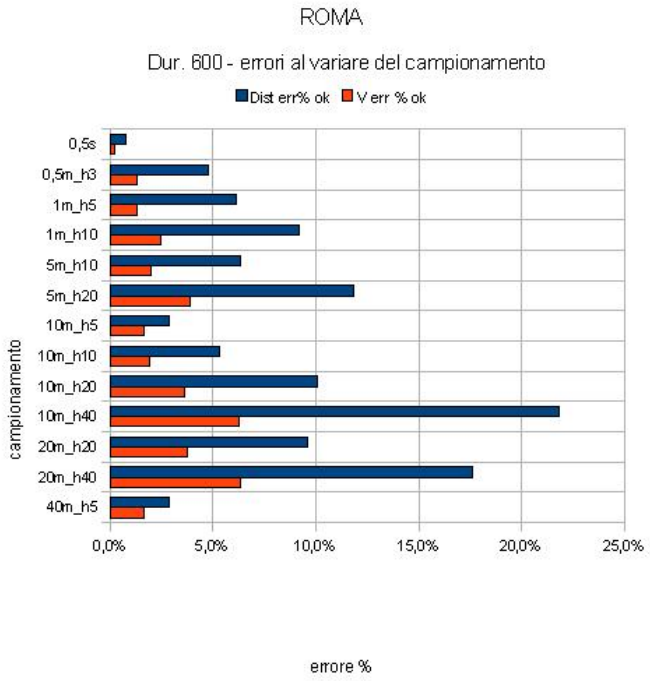

Figura 5.48: Grafico - Simple regression Roma 600, errori commessi

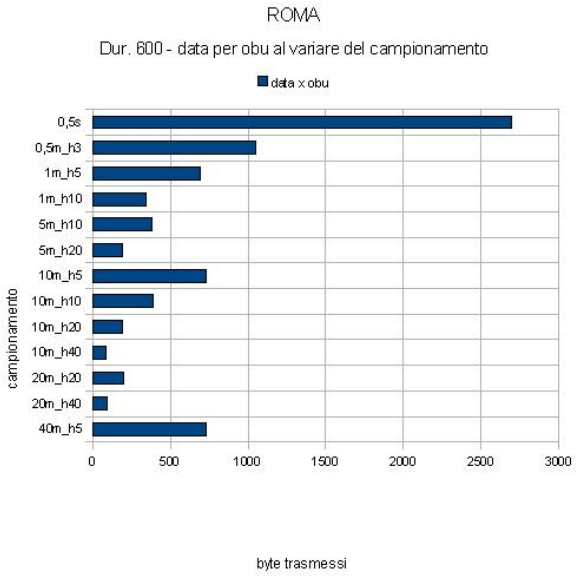

Figura 5.49: Grafico - Simple regression Roma 600, dimensione dati per OBU

Le tabelle 5.38 e 5.39 con i relativi grafici 5.50 e 5.51 mostrano i risultati ottenuti dalla prova precedente con l'applicazione di un filtro di selezione, fatto in modo manuale, tra 160 e 240 secondi, dove 240 è una delle durate massime di vita di un OBU all'interno di questa simulazione.

Come si può subito intuire dai dati, il *trade-off* dati inviati - errore commesso (se da una parte si ha il peggioramento delle stime dall'altro si ha un risparmio di comunicazione) privilegia la tecnica in questione rispetto al *simple time sampling* normale.

Tabella 5.38: Simple regression (ris. filtrati) Roma 600 - Parte 1 di 2

campion	hits	steps	data len	tot dist	avg v frv	avg v int	c dist
0,5s	263095	263095	4689167	1867889	94208	94190	1881976
10m_h5	79	480	2289	1365,66	20,6	20,6	1376
10m_h5	79	480	2315	1180,94	17,75	17,75	1190
10m_h5	76	458	2265	1399,04	22,12	22,12	1410
10m_h5	69	419	2026	1723,14	29,78	29,78	1737
10m_h5	67	406	2009	1408,72	25,12	25,12	1421
10m_h5	67	406	2007	1387,28	17,81	17,81	1400
10m_h5	63	382	1914	1741,19	33	33	1757
10m_h5	62	373	1852	1492,67	29	29	1508
10m_h5	60	361	1790	1767,13	35,45	35,45	1785
10m_h5	56	340	1718	1583,98	33,71	33,71	1602
10m_h5	52	320	1596	2361,58	53,39	53,39	2387
AVG	730	4425	21781	17411,33	317,73	317,73	17573

Tabella 5.39: Simple regression (ris. filtrati) Roma 600 - Parte 2 di 2

dist diff	life	c avg v	v diff (int)	**Dist err% ok**	**V err % ok**	**obu num**	**data x obu**
14133	131548	94245	180	**0,7510%**	**0,1910%**	1739	2696,47
10,34	240	20,64	0,04	**0,7515%**	**0,1938%**	1	2289
9,06	240	17,79	0,04	**0,7613%**	**0,2248%**	1	2315
10,96	229	22,17	0,05	**0,7773%**	**0,2255%**	1	2265
13,86	209,5	29,85	0,07	**0,7979%**	**0,2345%**	1	2026
12,28	203	25,2	0,08	**0,8642%**	**0,3175%**	1	2009
12,72	202,5	17,88	0,07	**0,9086%**	**0,3915%**	1	2007
15,81	191	33,12	0,12	**0,8998%**	**0,3623%**	1	1914
15,33	186,5	29,11	0,11	**1,0166%**	**0,3779%**	1	1852
17,87	180,5	35,6	0,15	**1,0011%**	**0,4213%**	1	1790
18,02	170	33,92	0,21	**1,1248%**	**0,6191%**	1	1718
25,42	160	53,71	0,32	**1,0649%**	**0,5958%**	1	1596
161,67	2212	318,99	1,26	**0,9200%**	**0,3950%**	11	1980,09

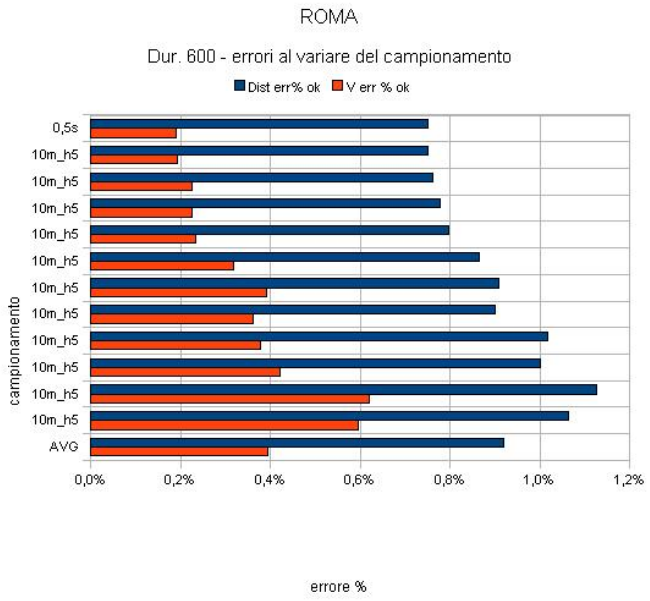

Figura 5.50: Grafico - Simple regression Roma 600 (filtrato), errori commessi

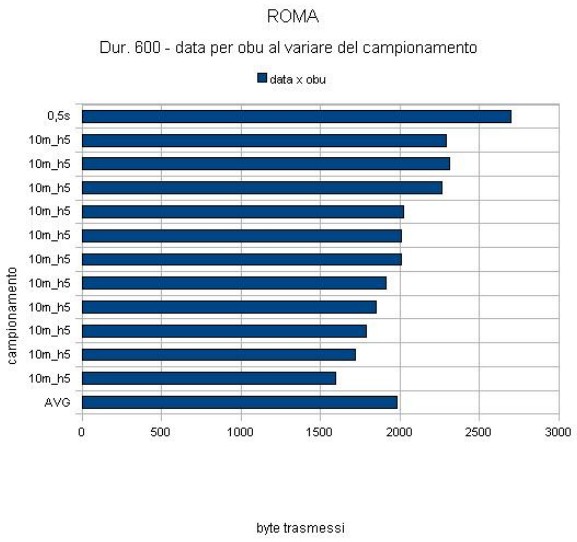

Figura 5.51: Grafico - Simple regression Roma 600 (filtr.), dim. dati per OBU

Roma, simulazione di 3600 secondi

Le tabelle 5.40 e 5.41 con i relativi grafici 5.52 e 5.53 mostrano i risultati ottenuti dallo scenario Roma 5.1.2 con durata di simulazione di 3600 secondi.

Date le poche considerazioni in più derivate da questa simulazione rispetto a quella di durata 600 secondi sullo stesso scenario, si preferisce riportarle come conclusione di questo paragrafo, in modo da poterle confrontare con altre più significative.

Tabella 5.40: Simple regression Roma 3600 - Parte 1 di 2

campion	hits	steps	data len	tot dist	avg v frv	avg v int	c dist
0,5s	1769579	1769579	32374325	12416358	571173	571164	12501852
7m_h3	435694	1769579	13309545	12270559	568329	568330	12501654
7m_h5	288534	1769579	8814548	12173409	566689	566710	12501534
7m_h7	214371	1769579	6550312	12063067	565369	565410	12501503
7m_h10	152529	1769579	4663755	11818552	562576	562624	12500584
10m_h3	435701	1769579	13309757	12270414	568186	568187	12501534
10m_h5	288704	1769579	8819840	12171680	566606	566613	12501534
10m_h7	214966	1769579	6567782	12067705	565053	565095	12501503
10m_h10	153910	1769579	4704530	11877396	562571	562673	12500839

Tabella 5.41: Simple regression Roma 3600 - Parte 1 di 2

dist diff	life	c avg v	v diff (int)	**Dist err% ok**	**V err % ok**	**obu num**	**data x obu**
85785	884790	571459	878	**0,6862%**	**0,1536%**	1739	18616,63
231096	884790	571459	3712	**1,8485%**	**0,6496%**	1739	7653,56
328125	884790	571459	5520	**2,6247%**	**0,9659%**	1739	5068,75
438436	884790	571459	7320	**3,5071%**	**1,2809%**	1739	3766,71
682032	884790	571459	10613	**5,4560%**	**1,8572%**	1739	2681,86
231120	884790	571459	3709	**1,8487%**	**0,6490%**	1739	7653,68
329854	884790	571459	5587	**2,6385%**	**0,9777%**	1739	5071,79
433798	884790	571459	7488	**3,4700%**	**1,3103%**	1739	3776,76
623443	884790	571459	10412	**4,9872%**	**1,8220%**	1739	2705,31

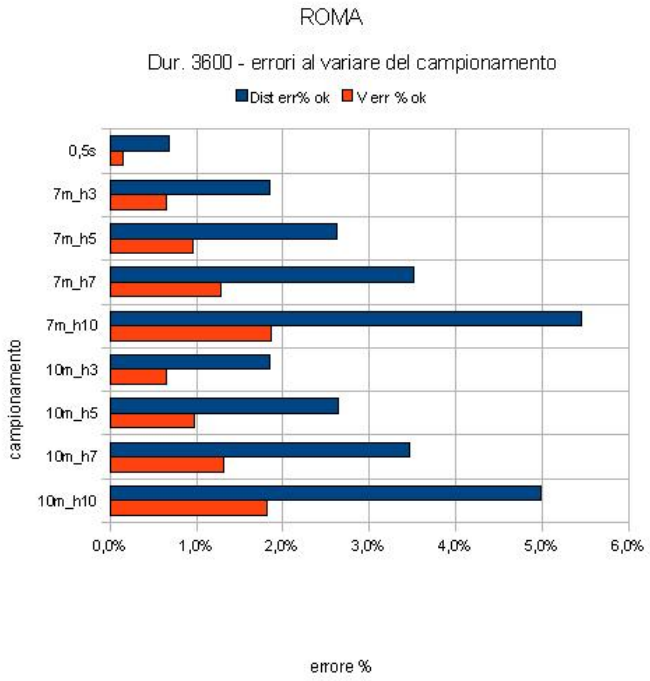

Figura 5.52: Grafico - Simple regression Roma 3600, errori commessi

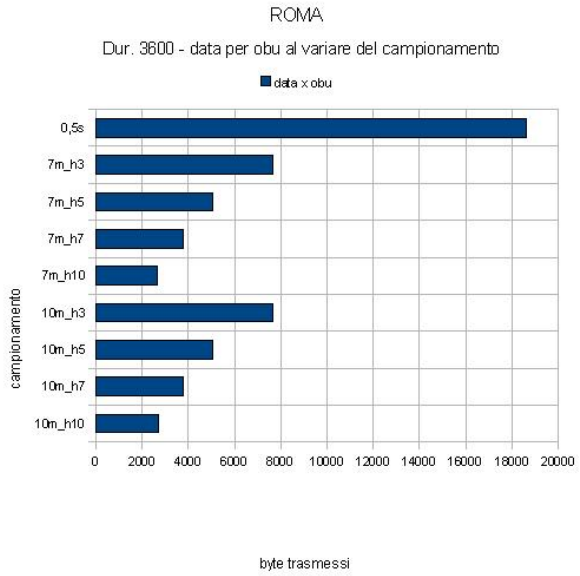

Figura 5.53: Grafico - Simple regression Roma 3600, dimensione dati per OBU

Note e considerazioni generali dall'analisi effettuata

Dall'analisi dei risultati relativi a questa tecnica (*Simple regression*), si può dedurre che nel totale della simulazione, sia di durata 600 secondi che 3600 secondi, considerando anche il *trade-off* dati inviati - errore commesso (se da una parte si ha il peggioramento delle stime dall'altro si ha un risparmio di comunicazione), vi è stato un lieve peggioramento rispetto alla tecnica *simple time sampling*.

Considerando, nella simulazione da 600 secondi, le OBU con tempo di vita massimo, ovvero da 160 a 240 secondi circa, si sono ottenute delle stime che attestano la validità di questa tecnica, infatti, dal grafico 5.50 si può vedere come l'errore rimanga pressoché invariato dal campionamento base a 0,5 secondi per le OBU con tempo di vita intorno ai 240 secondi, mentre l'invio di dati subisce una notevole variazione in positivo, ovvero se ne inviano circa il 13% in meno. Purtroppo dalla simulazione di durata 3600, anche se non considerati filtri, dove il tempo medio di vita di un OBU supera spesso i 240 secondi, i risultati non sono stati eccellenti come questi, ma hanno confermato la tendenza generale espressa all'inizio di questo paragrafo.

Un campionamento extra, effettuato a posteriori, ha dimostrato che, avendo come campionamento base 0,1 secondi e selezionando un una finestra di 10 dati passati (*history*) ed una tolleranza di 10 metri, si ottengono risultati in linea, dal punto di vista sia degli errori che del numero di dati inviati, con la tecnica *simple time sampling* (errore distanza 0,96%, velocità 0,42%, dati per OBU 2018).

I buoni risultati ottenuti con questa tecnica hanno infine indirizzato allo sviluppo della vera e propria tecnica di regressione la cui analisi è presentata alla sezione 5.3.7.

5.3.7 Linear regression

Come nell'analisi per la tecnica *simple time sampling* della sezione 5.3.2, per questa tecnica si sono utilizzati diversi scenari con diversi tempi di simulazione, in particolare Roma con durate di 600 e 3600 secondi, Beijing con durata di 1200 secondi, Toll Plaza con durata di 3600 secondi e infine Bologna in tre modalità: normale da 600 e 3600 secondi, RID ovvero con numero di veicoli RIDotto e percorsi circolari, della durata di 7200 secondi e ZTL, ovvero con numero di veicoli ulteriormente ridotto rispetto RID ed eliminazione delle Zone a Traffico Limitato, della durata di 14400 secondi.

Il campionamento base negli scenari di questa sezione è stato svolto a 0,5 secondi e 0,1 ed è identificato, rispettivamente, dalle sigle "on 0,5s" e "on 0,1s", mentre l'eventuale ausilio di un filtro è identificato dalla sigla "fx", dove x può essere un qualsiasi numero in virgola mobile ed è il valore che assume il filtro stesso nello specifico caso. In tutti i casi non specificati, si assume un campionamento base a 0,5 secondi.

Roma, simulazione di 600 secondi

Dall'analisi dei dati contenuti nelle tabelle 5.42 e 5.43 e dai relativi grafici 5.54 e 5.55 si deduce che la tecnica di regressione produce errori più elevati del campionamento con la tecnica *simple time sampling*, questo almeno con campionamento base a 0,5 secondi.

In particolare i campionamenti con *history* di lunghezza 3 e 5, sempre su campionamento base a 0,5 secondi, non hanno dimostrato un buon *trade-off* dati inviati - errore commesso, infatti, se da una parte si ha un risparmio di comunicazione di circa 2,5 e 3,7 volte rispetto al caso base, dall'altro si ha un peggioramento troppo accentuato delle stime.

Utilizzando invece la tecnica di regressione con campionamento base a 0,1 secondi, si ci accorge che il campionamento con *history* di lunghezza 5 equivale (ovviamente) al campionamento base per quanto riguarda gli errori, mentre la dimensione dei dati inviati cresce di circa 1,38 volte rispetto il caso base, questo proprio per la struttura dei dati inviati da questa tecnica, differente e più complessa delle altre.

Un buon compromesso, ovvero un buon *trade-off* dati inviati - errore commesso, si ottiene, sempre con campionamento base a 0,1 secondi, con *history* di lunghezza 8 10 e 15, che fornisce una via intermedia di campionamento tra i parametri 0,5 e 1 secondi della tecnica *simple time sampling*; il *trade-off* non è però migliorato.

Tabella 5.42: Regression Roma 600 - Parte 1 di 2

campion	hits	steps	data len	tot dist	avg v frv	avg v int	c dist
0,5s on 0.5s	263095	263095	4689167	1867889	94208	94190	1881976
regression h3 on 0.5s	64687	263094	1921967	1818292,4	62189,02	93632,6	1881967
regression h5 on 0.5s	42842	263094	1272848	1805808,99	73625,29	93322,53	1881967
regression h5 on 0.1s	217399	1310483	6455941	1855961,69	74900,22	93826,53	1870976
regression h8 on 0.1s	144622	1310483	4295162	1851876,41	81705,26	93754,71	1870976
regression h10 on 0.1s	118176	1310483	3509884	1849313,67	83950,64	93740,44	1870976
regression h15 on 0.1s	80965	1310483	2404852	1843245,26	86702,3	93587,18	1870976

Tabella 5.43: Regression Roma 600 - Parte 2 di 2

dist diff	life	c avg v	v diff (int)	**Dist err% ok**	**V err % ok**	obu num	**data x obu**
14133	131548	94245	180	**0,7510%**	**0,1910%**	1739	**2696,47**
62578,42	131547	92351,2	560,12	**3,3252%**	**0,6065%**	1739	**1105,21**
74699,04	131547	92351,2	779,68	**3,9692%**	**0,8443%**	1739	**731,94**
14721,87	131048,3	91815,69	185,54	**0,7869%**	**0,2021%**	1739	**3712,44**
18708,76	131048,3	91815,69	245,1	**0,9999%**	**0,2669%**	1739	**2469,9**
21185,52	131048,3	91815,69	284,56	**1,1323%**	**0,3099%**	1739	**2018,33**
27197,57	131048,3	91815,69	389,36	**1,4537%**	**0,4241%**	1739	**1382,89**

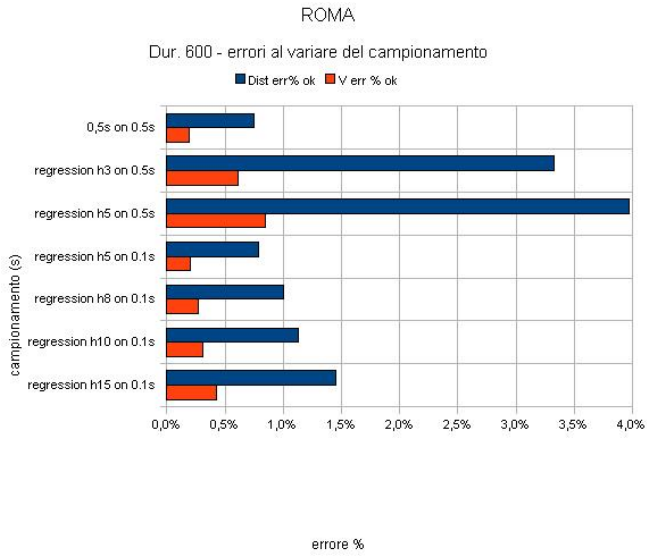

Figura 5.54: Grafico - Regression, Roma 600, errori commessi

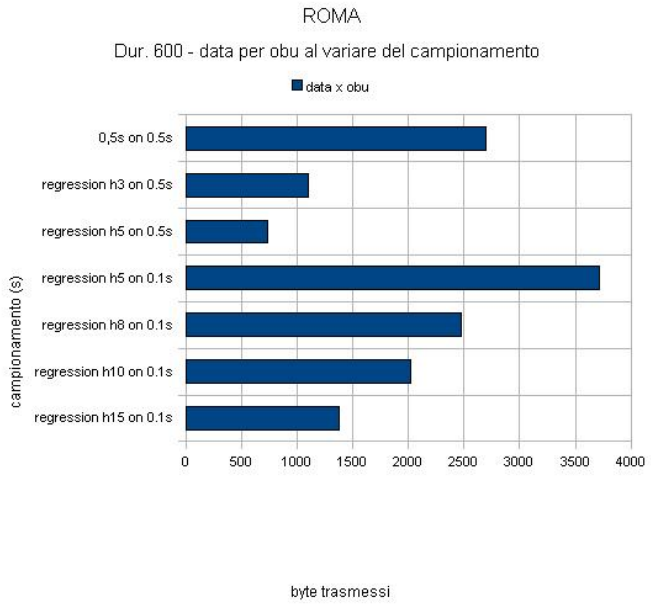

Figura 5.55: Grafico - Regression, Roma 600, dimensione dati per OBU

Bologna, simulazione di 600 secondi, con e senza filtro

Dall'analisi dei dati contenuti nelle tabelle 5.44 e 5.45 e dai relativi grafici 5.56 e 5.57 si sono raffinati i parametri con cui si sono eseguite le analisi di seguito descritte i cui dati figurano nelle tabelle 5.46 e 5.47 e nei grafici associati 5.58 e 5.59.

La prima valutazione di questi dati ha confermato le buone possibilità fornite dalla tecnica *regression*, che permette di campionare i dati ad intervalli molto distanti da un punto di vista temporale minimizzando gli errori.

Tabella 5.44: Regression Bologna 600 - Parte 1 di 2

campion	hits	steps	data len	tot dist	avg v frv	avg v int	c dist
0,5s on 0.5	538494	2691161	10933227	1550371,05	19121,01	19202,27	1544805
0,5s on 0,1	536373	536373	10902803	1548424,99	18961,55	19051,86	1543971
regression h3 on 0,5	133615	536373	4641419	1530501,68	12336,38	18745,21	1543971
regression h3 on 0,1	537886	2691161	18676864	1551149,77	12757,26	19202,07	1544805
regression h5 on 0,5	88956	536373	3090019	1524969,99	14370,82	18639	1543971
regression h5 on 0,1	448088	2691161	15558723	1551536,03	15220,78	19192,43	1544805
regression h7 on 0,5	66621	536373	2314314	1518528,48	14899,44	18473,06	1543971
regression h7 on 0,1	335965	2691161	11665725	1551555,39	16218,23	19177	1544805
regression h10 on 0,5	48338	536373	1679248	1507682	14998,56	18233,25	1543971
regression h10 on 0,1	244229	2691161	8480400	1551893,32	16875,89	19158,8	1544805
regression h15 on 0,5	33122	536373	1150662	1488806,38	14528,48	17799,84	1543971
regression h15 on 0,1	167780	2691161	5825927	1552829,86	17238,65	19105,34	1544805
regression h20 on 0,5	25134	536373	873248	1460469,51	13607,08	17232,34	1543971
regression h20 on 0,1	127755	2691161	4436273	1554125,38	17311,48	19088,75	1544805
regression h40 on 0,1	65239	2691161	2265572	1552903,26	16830,59	19001,35	1544805

Tabella 5.45: Regression Bologna 600 - Parte 2 di 2

dist diff	life	c avg v	v diff (int)	**Dist err% ok**	**V err % ok**	obu num	**data x obu**
6842,4	269116	18964,7	73,82	**0,4429%**	**0,3893%**	780	**14016**
6327,05	268186,5	18868,68	84,44	**0,4098%**	**0,4475%**	777	**14031**
14112,41	268186,5	18992,17	290,54	**0,9140%**	**1,5298%**	777	**5973**
8249,73	269116	19146,11	97,5	**0,5340%**	**0,5092%**	780	**23944**
19718,75	268186,5	18992,17	437,32	**1,2771%**	**2,3026%**	777	**3976**
8849,36	269116	19146,11	113,59	**0,5728%**	**0,5933%**	780	**19947**
26483,56	268186,5	18992,17	625,39	**1,7153%**	**3,2929%**	777	**2978**
9547,2	269116	19146,11	130,06	**0,6180%**	**0,6793%**	780	**14956**
38122,65	268186,5	18992,17	895,88	**2,4691%**	**4,7171%**	777	**2161**
10938,85	269116	19146,11	159,6	**0,7081%**	**0,8336%**	780	**10872**
58753,66	268186,5	18992,17	1428,54	**3,8054%**	**7,5217%**	777	**1480**
13862,49	269116	19146,11	238,5	**0,8974%**	**1,2457%**	780	**7469**
87502,86	268186,5	18992,17	2014,75	**5,6674%**	**10,6083%**	777	**1123**
16727,56	269116	19146,11	304,09	**1,0828%**	**1,5883%**	780	**5687**
27715,87	269116	19146,11	607,22	**1,7941%**	**3,1715%**	780	**2904**

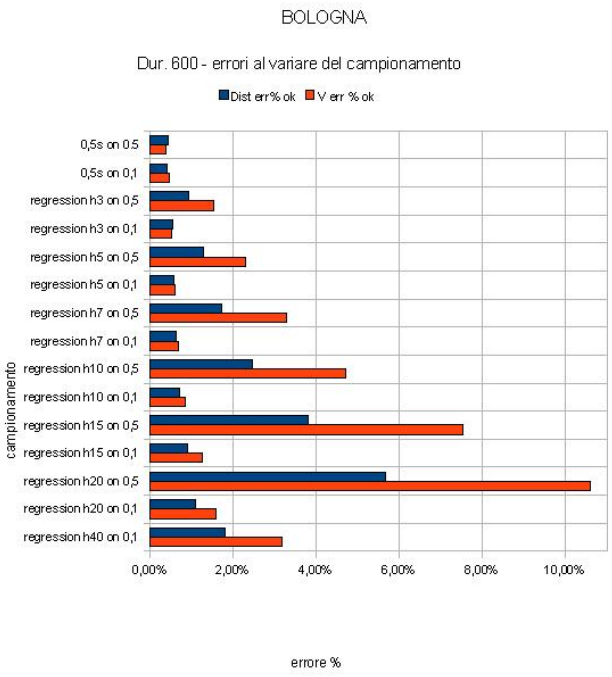

Figura 5.56: Grafico - Regression, Bologna 600, errori commessi sf

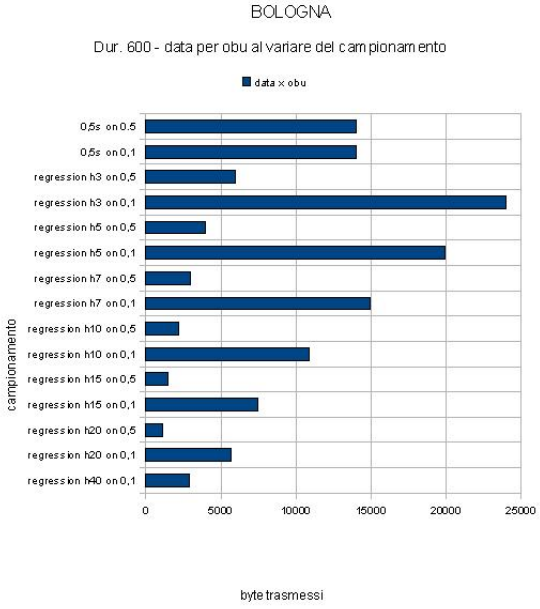

Figura 5.57: Grafico - Regression, Bologna 600, dimensione dati per OBU sf

Dall'analisi dei dati contenuti nelle tabelle 5.46 e 5.47 e dai relativi grafici 5.58 e 5.59 si deduce che l'immissione di un filtro per scartare le OBU con tempo di vita inferiore a 60 secondi ha determinato un notevole miglioramento delle stime sull'errore commesso, sia in termini di distanza che di velocità.

In particolare, considerando il campionamento con *history* di dimensione 3, e confrontando le stime ottenute con filtro a 0,5 e 60 secondi, si ci accorge che la riduzione dell'errore è di circa 2,6 volte per la velocità e 1,5 volte per la distanza.

Valutando anche il comportamento reale dei veicoli, che difficilmente avranno tempo di vita di solo 60 secondi, si può affermare che, scegliendo gli opportuni parametri, la tecnica di regressione ha un ottimo *trade-off* dati inviati - errore commesso, infatti, se da una parte si ha il peggioramento delle stime dell'errore di solo 1.4 volte per la distanza e 1.5 volte per la velocità, dall'altro si ha un risparmio di circa 2.5 volte sulla comunicazione.

Tabella 5.46: Regression Bologna 600 (con filtro) - Parte 1 di 2

campion	hits	steps	data len	tot dist	avg v frv	avg v int	c dist
0,5s on 0.5	538494	2691161	10933227	1550371,05	19121,01	19202,27	1544805
0,5s on 0,1	536373	536373	10902803	1548424,99	18961,55	19051,86	1543971
regression h3 on 0,5 f0.5	133615	536373	4641419	1530501,68	12336,38	18745,21	1543971
regression h7 on 0,1 f0.5	335965	2691161	11665725	1551555,39	16218,23	19177	1544805
regression h10 on 0,1 f0.5	244229	2691161	8480400	1551893,32	16875,89	19158,8	1544805
regression h8 on 0,1 f60	298593	2691161	10368240	1551703,96	16511,63	19175,98	1544805

Tabella 5.47: Regression Bologna 600 (con filtro) - Parte 2 di 2

dist diff	life	c avg v	v diff (int)	**Dist err% ok**	**V err % ok**	obu num	**data x obu**
6842,4	269116	18964,7	73,82	**0,4429%**	**0,3893%**	780	**14016**
6327,05	268186,5	18868,68	84,44	**0,4098%**	**0,4475%**	777	**14031**
14112,41	268186,5	18992,17	290,54	**0,9140%**	**1,5298%**	777	**5973**
9547,2	269116	19146,11	130,06	**0,6180%**	**0,6793%**	780	**14956**
10938,85	269116	19146,11	159,6	**0,7081%**	**0,8336%**	780	**10872**
8823,65	269116	13594,35	75,93	**0,5712%**	**0,5585%**	780	**13292**

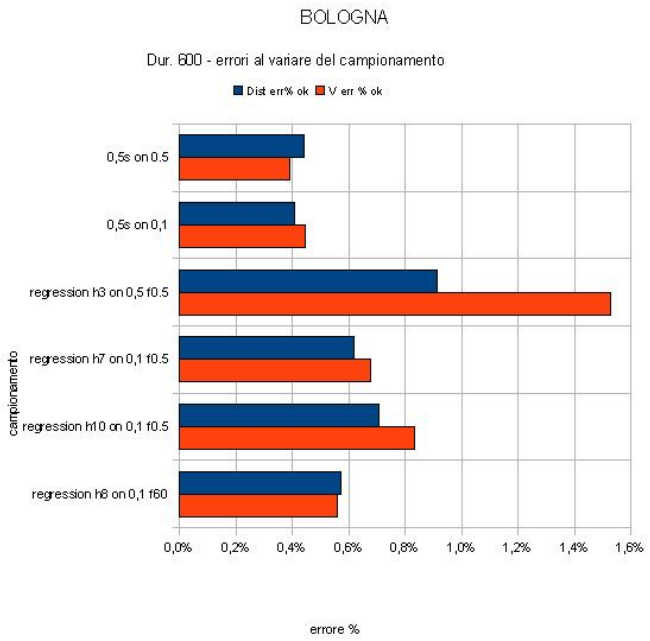

Figura 5.58: Grafico - Regression, Bologna 600, errori commessi cf

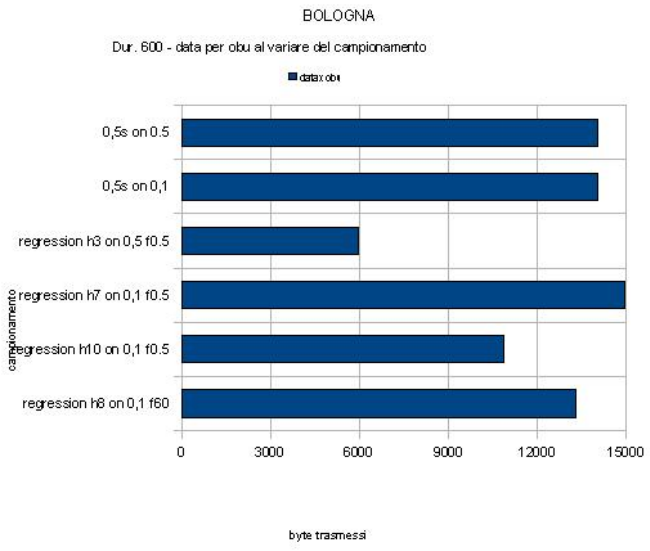

Figura 5.59: Grafico - Regression, Bologna 600, dimensione dati per OBU cf

Roma, simulazione di 3600 secondi

Dall'analisi dei dati contenuti nelle tabelle 5.48 e 5.49 e dai relativi grafici 5.60 e 5.61 si deduce che in questo scenario, a differenza dell'analisi precedente (sezione 5.3.7), il tempo medio di vita dei veicoli è di soli 83 secondi contro i 345 precedenti; da qui, con tutta probabilità, il peggioramento delle stime sull'errore.

L'applicazione di un filtro da 60 secondi, confrontato con lo stesso campionamento, ma con filtro da 0,5 secondi, ha dimostrato, ancora una volta, che all'aumentare del tempo di vita dell'OBU diminuiscono gli errori commessi sia sulla distanza che sulla velocità.

Tabella 5.48: Regression Roma 3600 - Parte 1 di 2

campion	hits	steps	data len	tot dist	avg v frv	avg v int	c dist
0,5s	1769579	1769579	32374325	12416358	571173	571164	12501852
regression h3 f0.5	435700	1769578	13310370	12111183,71	377826,66	568922,19	12501841
regression h3 f60	435700	1769578	13310370	12111183,71	377826,66	568922,19	12501841
regression h5 f0.5	288736	1769578	8821457	12035105,59	448856,28	568230	12501841
regression h5 f60	288736	1769578	8821457	12035105,59	448856,28	568230	12501841

Tabella 5.49: Regression Roma 3600 - Parte 2 di 2

dist diff	life	c avg v	v diff (int)	**Dist err% ok**	**V err % ok**	obu num	**data x obu**
85785	884790	571459	878	**0,6862%**	**0,1536%**	10666	**3035,28**
390660,78	884789	571458,84	3849	**3,1248%**	**0,6735%**	10666	**1247**
196564,14	884789	269115,68	1836,04	**1,5723%**	**0,6822%**	10666	**1247**
466736,28	884789	571458,84	5331,19	**3,7333%**	**0,9329%**	10666	**827**
235210,83	884789	269115,68	2596,01	**1,8814%**	**0,9646%**	10666	**827**

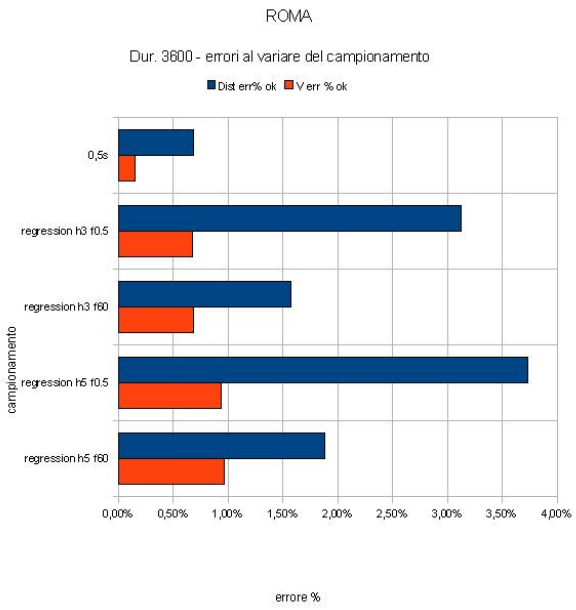

Figura 5.60: Grafico - Regression, Roma 3600, errori commessi

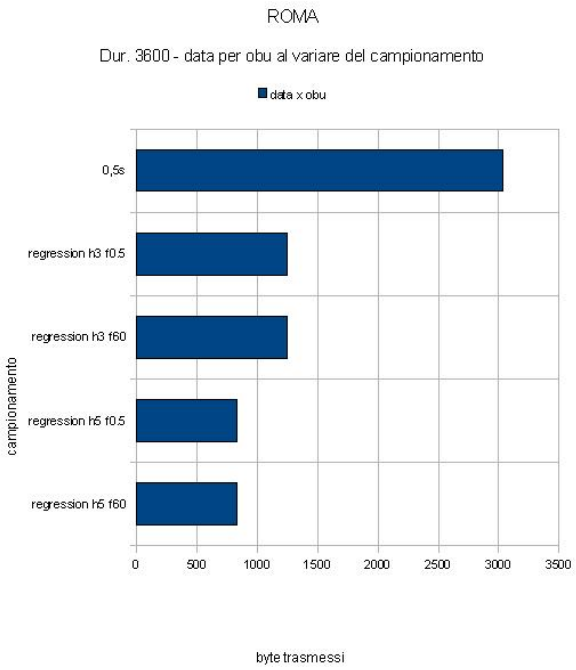

Figura 5.61: Grafico - Regression, Roma 3600, dimensione dati per OBU

Beijing, simulazione di 1200 secondi

Dall'analisi dei dati contenuti nelle tabelle 5.50 e 5.51, e dai relativi grafici 5.62 e 5.63, di seguito presentati, si deduce che i risultati ottenuti non hanno soddisfatto le aspettative.

In particolare, le stime sull'errore sono risultate molto elevate e, confrontate con un campionamento a 4 secondi della tecnica *simple time sampling*, hanno evidenziato un aumento dell'errore sia sulla distanza che sulla velocità, oltre ad un invio di dati in media 2,8 volte superiore.

Tali risultati hanno portato ad ulteriori analisi per verificare eventuali problemi della tecnica di regressione, come presentato di seguito.

Nelle tabelle 5.52 e 5.53, e dai relativi grafici 5.64 e 5.65, di seguito presentati, si deduce che i risultati della tecnica di regressione, effettuati su un campionamento base a 0,1 secondi, sono in linea con quelli della tecnica *simple time sampling*, anche se quest'ultima è da preferire per quanto riguarda il *trade-off* dati inviati - errore commesso, infatti, a parità di errore si ha un guadagno medio di circa 2 volte sulla comunicazione.

Probabilmente, come nei casi precedenti visibili in sezione 5.3.7 e 5.3.7, l'utilizzo di un filtro, ad esempio sui 60 secondi, avrebbe prodotto risultati nettamente migliori ed in linea con i precedenti, considerando soprattuto che il tempo di vita medio dell'OBU in questa simulazione è basso, ovvero di circa 100 secondi, e, dalle analisi precedenti, si è dimostrato che al crescere del tempo di vita si ottengono risultati migliori.

Tabella 5.50: Regression Beijing 1200 (on 0,5) - Parte 1 di 2

campion	hits	steps	data len	tot dist	avg v frv	avg v int	c dist
0,5s	1168863	1168863	19972551	1923792,75	90938,01	90968,48	1943945
regression h3 on 0,5s	288616	1168863	8656885	1844086,39	57440,53	88546,95	1943945
regression h4 on 0,5s	230335	1168863	6909037	1833567,48	63772,09	87960,64	1943945
regression h5 on 0,5s	191471	1168863	5743617	1820821,24	66388,19	87192,17	1943945

Tabella 5.51: Regression Beijing 1200 (on 0,5) - Parte 2 di 2

dist diff	life	c avg v	v diff (int)	**Dist err% ok**	**V err % ok**	obu num	**data x obu**
20341,32	584431,5	91299,76	479,48	**1,0464%**	**0,5252%**	5728	**3486**
99863,42	584431,5	91299,76	2882,67	**5,1372%**	**3,1574%**	5728	**1511**
110379,96	584431,5	91299,76	3491,79	**5,6781%**	**3,8245%**	5728	**1206**
123124,21	584431,5	91299,76	4284,55	**6,3337%**	**4,6928%**	5728	**1002**

Tabella 5.52: Regression Beijing 1200 (on 0,1) - Parte 1 di 2

campion	hits	steps	data len	tot dist	avg v frv	avg v int	c dist
0,5s on 0,5s	1168863	1168863	19972551	1923792,75	90938,01	90968,48	1943945
0,5s on 0,1s	1111986	5549046	19047843	1933772,2	92329,44	92333,79	1945487
regression h5 on 0,1s	921476	5549046	27618597	1923094,08	71633,06	91829,42	1945487
regression h7 on 0,1s	690456	5549046	20694668	1918688,37	75293,18	91578,55	1945487
regression h8 on 0,1s	613359	5549046	18384486	1916222,04	76614,12	91468,17	1945487
regression h10 on 0,1s	501305	5549046	15025930	1911748,63	78857,41	91266,99	1945487

Tabella 5.53: Regression Beijing 1200 (on 0,1) - Parte 2 di 2

dist diff	life	c avg v	v diff (int)	**Dist err% ok**	**V err % ok**	obu num	**data x obu**
20341,32	584431,5	91299,76	479,48	**1,0464%**	**0,5252%**	5728	**3486**
12315,49	554904,6	92466,65	293,3	**0,6330%**	**0,3172%**	5728	**3325**
22602,62	554904,6	92466,65	777,13	**1,1618%**	**0,8404%**	5728	**4821**
26951,09	554904,6	92466,65	1029,2	**1,3853%**	**1,1131%**	5728	**3612**
29418,17	554904,6	92466,65	1147,61	**1,5121%**	**1,2411%**	5728	**3209**
33864,6	554904,6	92466,65	1366,99	**1,7407%**	**1,4784%**	5728	**2623**

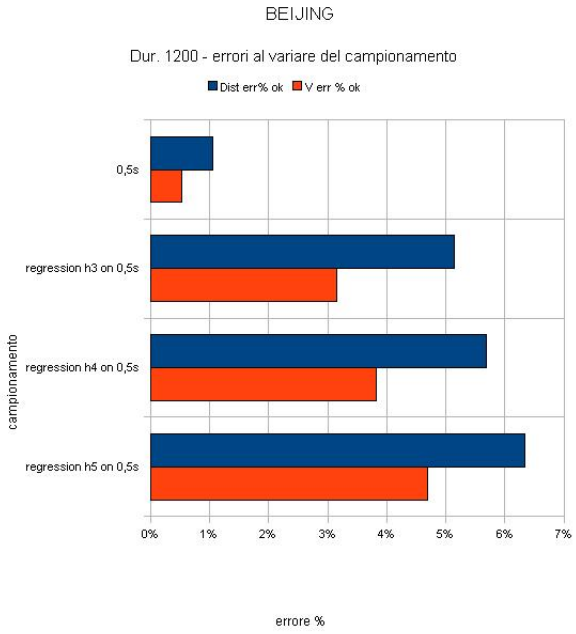

Figura 5.62: Grafico - Regression, Beijing 1200 (on 0,5), errori commessi

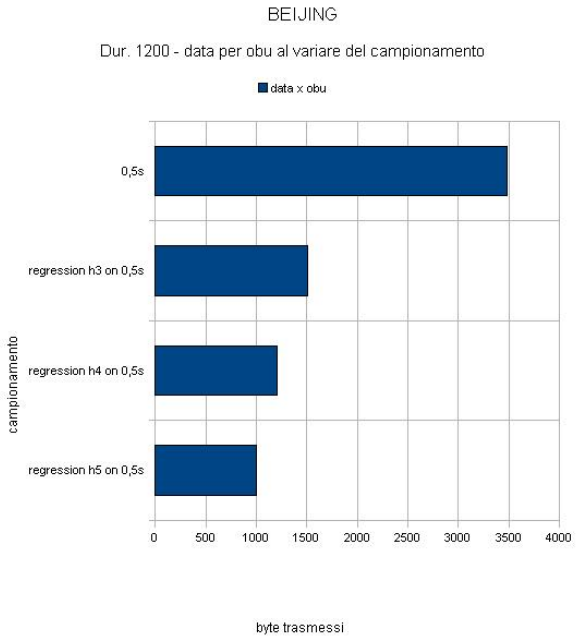

Figura 5.63: Grafico - Regression, Beijing 1200 (on 0,5), dim. dati per OBU

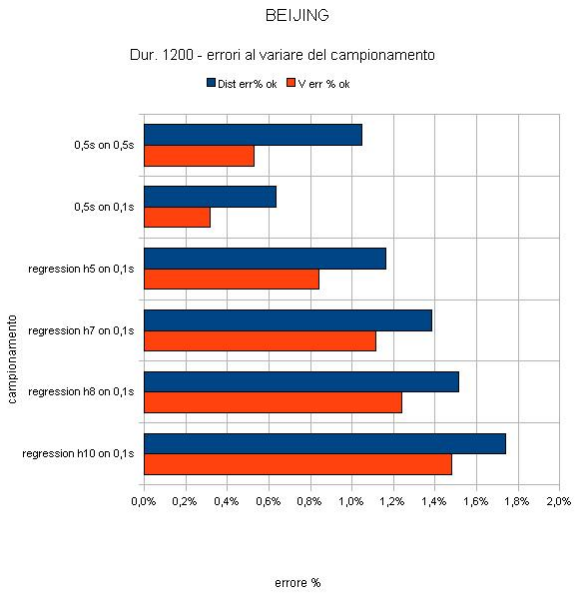

Figura 5.64: Grafico - Regression, Beijing 1200 (on 0,1), errori commessi

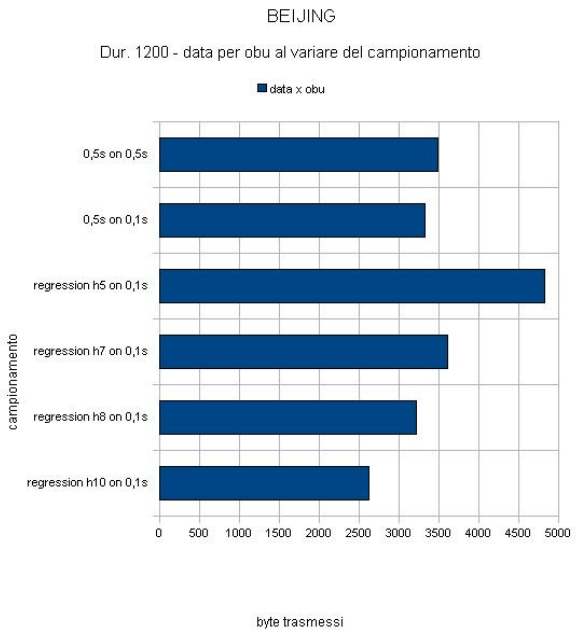

Figura 5.65: Grafico - Regression, Beijing 1200 (on 0,1), dim. dati per OBU

Toll Plaza, simulazione di 3600 secondi

Dall'analisi dei dati contenuti nelle tabelle 5.54 e 5.55 e dai relativi grafici 5.66 e 5.67 si deduce che, come nel caso precedente visibile in sezione 5.3.7, i risultati ottenuti non sono pienamente soddisfacenti.

Le cause di tali errori sono da attribuire soprattutto, sempre come nel caso precedente visibile in sezione 5.3.7, alla scarsa durata di vita dei veicoli, che mediamente si aggira sui 154 secondi, tempi molto diversi dalla realtà, nella quale difficilmente un conducente si mette alla guida per soli 2 minuti.

L'utilizzo di un filtro a 90 secondi ha confermato le deduzioni precedenti, secondo le quali all'aumentare del tempo di vita dell'OBU diminuiscono gli errori commessi sia sulla distanza che sulla velocità. In questo particolare caso la diminuzione dell'errore sulla distanza è stata dello 0,27% e quella sulla velocità dello 0,21%.

Tabella 5.54: Regression Toll Plaza 3600 - Parte 1 di 2

campionamento (0,5)	hits	steps	data len	tot dist	avg v frv	avg v int	c dist
0,5s	2428861	2428861	42726752	14338142,43	342447,65	342441,77	14400721
1s	1216379	2428861	21398435	14299353,86	342204,98	341910,71	14400721
regression h3 on 0.5	602287	2428861	17743140	14093465,02	225378,17	339228,87	14400721
regression h3 on 0.5 f90	602287	2428861	17743140	14093465,02	225378,17	339228,87	14400721
regression h5 on 0.5	400222	2428861	11794840	14025045	268266,4	337561,31	14400721

Tabella 5.55: Regression Toll Plaza 3600 - Parte 2 di 2

dist diff	life	c avg v	v diff (int)	**Dist err% ok**	**V err % ok**	obu num	**data x obu**
63257,01	1214430,5	342906,13	909,13	**0,4393%**	**0,2651%**	7902	**5407**
101509,99	1214430,5	342906,13	1250,05	**0,7049%**	**0,3645%**	7902	**2707**
307255,98	1214430,5	342906,13	3829,99	**2,1336%**	**1,1169%**	7902	**2245**
268475,9	1214430,5	290619,58	2629,93	**1,8643%**	**0,9049%**	7902	**2245**
327914,17	1214430,5	290619,58	3893,78	**2,2771%**	**1,3398%**	7902	**1492**

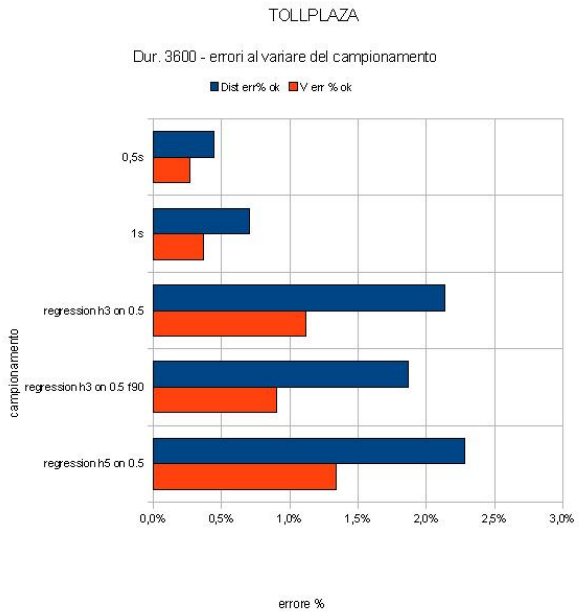

Figura 5.66: Grafico - Regression, Toll Plaza 3600, errori commessi

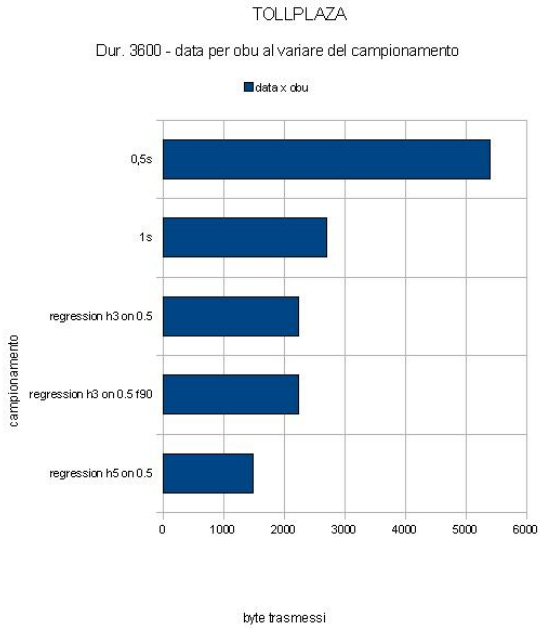

Figura 5.67: Grafico - Regression, Toll Plaza 3600, dimensione dati per OBU

Bologna (RID) simulazione di 7200 secondi

Dall'analisi dei dati contenuti nelle tabelle 5.56 e 5.57 e dai relativi grafici 5.68 e 5.69 si deduce che, in questo scenario appositamente creato per aumentare la durata di vita dei veicoli, ma non modificato da un punto di vista stradale, ovvero la gran parte delle strade, gli incroci, i semafori, etc., sono rimasti identici, mentre le vie in uscita rientrano nella mappa, i risultati ottenuti sono stati ottimi.

In particolare, tutte e tre le prove sperimentali con i differenti parametri, ovvero *history* di lunghezza 3, 4 e 5, hanno prodotto errori minori [3] del campionamento base a 0,5 secondi; in dettaglio:

- con il parametro *history* di lunghezza 3 l'errore sulla distanza si è ridotto di **5,6** volte, quello sulla velocità di **2** volte e l'invio di dati è diminuito di **2,3** volte
- con il parametro *history* di lunghezza 4 l'errore sulla distanza si è ridotto di **3,5** volte, quello sulla velocità di **1,9** volte e l'invio di dati è diminuito di **2,9** volte
- con il parametro *history* di lunghezza 5 l'errore sulla distanza si è ridotto di **1,8** volte, quello sulla velocità è aumentato di **1,1** volte e l'invio di dati è diminuito di **3,5** volte.

L'ottima performance della tecnica di regressione deriva anche, come già introdotto, dalla longevità dei veicoli, che si aggira sui 57 minuti e 14 secondi.

Tabella 5.56: Regression Bologna (RID) 7200 - Parte 1 di 2

campion	hits	steps	data len	tot dist	avg v frv	avg v int	c dist
0,5s	940851	940851	19694170	3884955,21	4035,66	4058,64	3864127
regression h3 on 0.5	235124	940851	8440011	3865316,13	2630,58	4034,44	3864127
regression h4 on 0.5	188087	940851	6751424	3859789,5	2910,26	4029,08	3864127
regression h5 on 0.5	156726	940851	5625646	3853361,49	3058,12	4016,31	3864127

Tabella 5.57: Regression Bologna (RID) 7200 - Parte 1 di 2

dist diff	life	c avg v	v diff (int)	**Dist err% ok**	**V err % ok**	obu num	**data x obu**
20868,91	470425,5	3998,91	21,15	**0,5401%**	**0,5289%**	137	**143753**
3722,86	470425,5	4038,11	10,71	**0,0963%**	**0,2652%**	137	**61605**
5917,64	470425,5	4038,11	11,21	**0,1531%**	**0,2776%**	137	**49280**
11889,08	470425,5	4038,11	23,11	**0,3077%**	**0,5722%**	137	**41063**

[3] A parte l'aumento quasi irrisorio di uno 0,04% dell'ultimo caso nell'errore sulla velocità (considerando anche l'invio dati che risulta 3.5 volte inferiore al campionamento base)

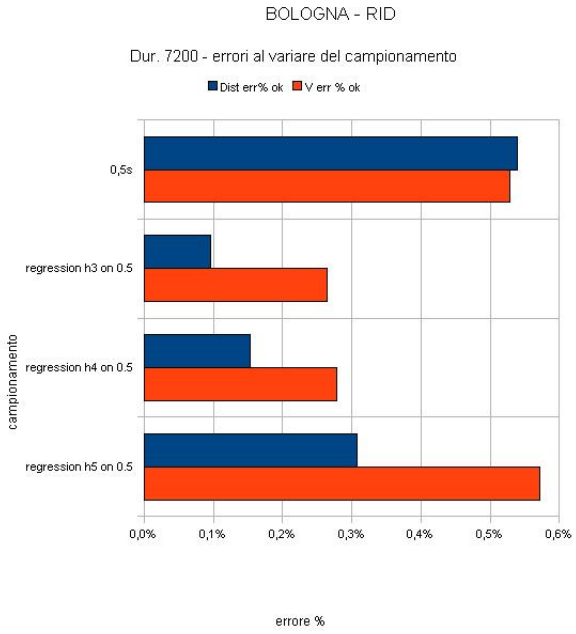

Figura 5.68: Grafico - Regression, Bologna (RID) 7200, errori commessi

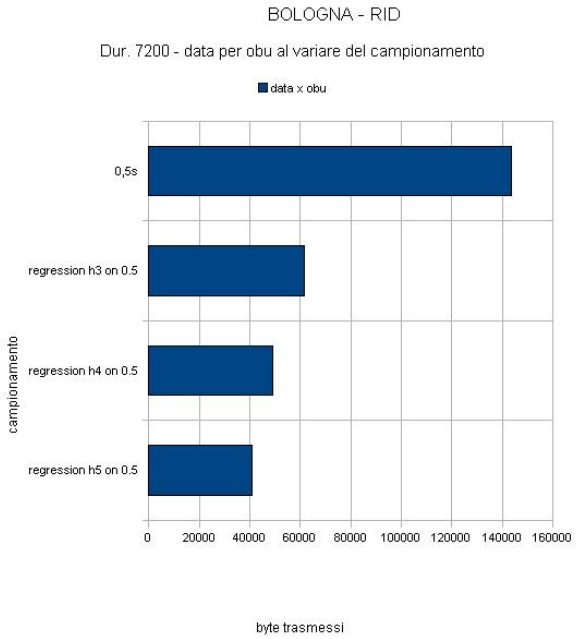

Figura 5.69: Grafico - Regression, Bologna (RID) 7200, dim. dati per OBU

Bologna (ZTL) simulazione di 14400 secondi

Dall'analisi dei dati contenuti nelle tabelle 5.59, 5.60, 5.61 e 5.62 e dai relativi grafici 5.70, 5.71, 5.72 e 5.73 di seguito presentati, si deduce che, a parte un fenomeno particolare dovuto al netto miglioramento delle stime sull'errore con un campionamento a 1 secondo piuttosto che con quello più fine a 0,5 secondi, i risultati riguardanti la tecnica *Linear Regression* sono molto soddisfacenti.

In particolare, con campionamento base a 0,5 secondi, il risultato migliore, in termini di errori, ottenuto con la tecnica *simple time sampling*, ovvero quello con campionamento a 1 secondo, confrontato con il risultato migliore ottenuto con la tecnica *Linear Regression*, ovvero quello con dimensione dell'*history* pari a 5, ha valorizzato la tecnica *Linear Regression*, infatti, l'errore sulla distanza è diminuito di **1,2** volte, quello sulla velocità di **1,5** volte e l'invio di dati è diminuito di **1,8** volte.

Con campionamento base a 0,1 secondi, invece, il risultato migliore, in termini di errori, ottenuto con la tecnica *simple time sampling*, ovvero quello con campionamento a 1 secondo, confrontato con il risultato migliore ottenuto sempre con la tecnica *simple time sampling* con campionamento a 0,5 secondi, ha valorizzato la tecnica *Linear Regression* per quanto riguarda il *trade-off* errori - dati inviati, infatti, se l'errore sulla distanza è diminuito di **1,6** volte e quello sulla velocità di **1,7** volte, l'invio di dati è rimasto invariato e superiore di 1,8 volte a quello della tecnica *Linear Regression*.

Il riassunto di quanto detto è visibile nella tabella di confronto 5.58 sottostante.

Tabella 5.58: Linear regression - Bologna ZTL 14400, confronto migliori risultati

campion	Dist err% ok	V err% ok	data x obu	Rapp. dist	Rapp. V	Rapp. data
1s on 0.5	0,1187%	0,1248%	196730	1	1	1
1s on 0.1	0,0729%	0,0739%	196765	1,6	1,7	1
regression h5 on 0.5	0,0989%	0,0824%	112654	1,2	1,5	1,8

Tabella 5.59: Regression Bologna 14400 (ZTL on 0,5) - Parte 1 di 2

campion	hits	steps	data len	tot dist	avg v frv	avg v int	c dist
0,5s on 0.5	947904	947904	20065820	4067417,62	1566,47	1573,7	4048885
1s on 0.5	473966	947904	10033241	4053636,6	1566,45	1568,41	4048885
2s on 0.5	236995	947904	5016756	4016597,63	1566,2	1554,18	4048885
regression h3 on 0.5	236944	947904	8618343	4055344,42	1026,74	1569,35	4048885
regression h4 on 0.5	189552	947904	6894632	4052680,59	1141,73	1568,36	4048885
regression h5 on 0.5	157953	947904	5745396	4047259,97	1206,49	1566,4	4048885
regression h6 on 0.5	135389	947904	4924697	4042628,02	1242,39	1564,53	4048885

Tabella 5.60: Regression Bologna 14400 (ZTL on 0,5) - Parte 2 di 2

dist diff	life	c avg v	v diff (int)	**Dist err% ok**	**V err % ok**	obu num	**data x obu**
18532,62	473952	1566,47	7,23	**0,4577%**	**0,4613%**	51	**393447**
4807,26	473952	1566,47	1,96	**0,1187%**	**0,1248%**	51	**196730**
32287,37	473952	1566,47	12,29	**0,7974%**	**0,7848%**	51	**98367**
6459,42	473952	1566,47	2,87	**0,1595%**	**0,1833%**	51	**168987**
4192,97	473952	1566,47	1,99	**0,1036%**	**0,1273%**	51	**135188**
4002,66	473952	1566,47	1,29	**0,0989%**	**0,0824%**	51	**112654**
7945,27	473952	1566,47	2,65	**0,1962%**	**0,1692%**	51	**96562**

Tabella 5.61: Regression Bologna 14400 (ZTL on 0,1) - Parte 1 di 2

campion	hits	steps	data len	tot dist	avg v frv	avg v int	c dist
0,5s on 0,1	947881	4739322	20069503	4092149,43	1577,67	1584,06	4075707
regression h5 on 0,1	789858	4739322	28728624	4099377,21	1256,56	1586,89	4075707
regression h7 on 0,1	592387	4739322	21546047	4100881,35	1339,61	1587,48	4075707
regression h8 on 0,1	526563	4739322	19152054	4101639,48	1364,53	1587,79	4075707
regression h10 on 0,1	430820	4739322	15670037	4103588,07	1397,52	1588,55	4075707
regression h12 on 0,1	364538	4739322	13259170	4106134,83	1417,16	1589,54	4075707
regression h14 on 0,1	315926	4739322	11491234	4109212,32	1429,69	1590,78	4075707

Tabella 5.62: Regression Bologna 14400 (ZTL on 0,1) - Parte 2 di 2

dist diff	life	c avg v	v diff (int)	**Dist err% ok**	**V err % ok**	obu num	**data x obu**
16442,43	473932,1	1577,67	6,39	**0,4034%**	**0,4052%**	51	**393519**
23670,21	473932,1	1577,67	9,21	**0,5808%**	**0,5840%**	51	**563306**
25174,35	473932,1	1577,67	9,81	**0,6177%**	**0,6218%**	51	**422471**
25932,48	473932,1	1577,67	10,12	**0,6363%**	**0,6412%**	51	**375530**
27881,07	473932,1	1577,67	10,88	**0,6841%**	**0,6898%**	51	**307255**
30427,83	473932,1	1577,67	11,87	**0,7466%**	**0,7524%**	51	**259983**
33505,32	473932,1	1577,67	13,11	**0,8221%**	**0,8309%**	51	**225318**

Ulteriori prove relative alla tecnica *Linear Regression* con campionamento base a 0,1 secondi e *history* di lunghezza massima 30, hanno confermato l'andamento dei dati riportati in tabella 5.61 e 5.62. Tali prove sono state fatte per verificare il registrarsi dell'eventuale minimo relativo degli errori al crescere dell'*history* per la tecnica *Linear Regression* e del tempo di campionamento per la tecnica *Simple time sampling*.

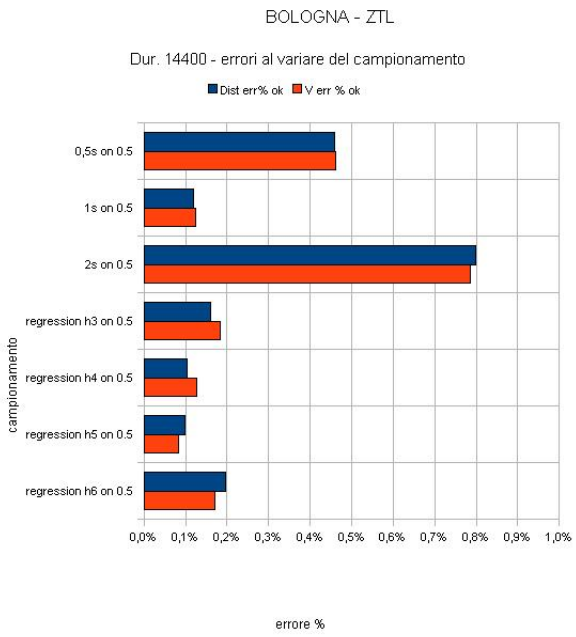

Figura 5.70: Grafico - Regression, Bologna 14400 (ztl on 0,5), errori commessi

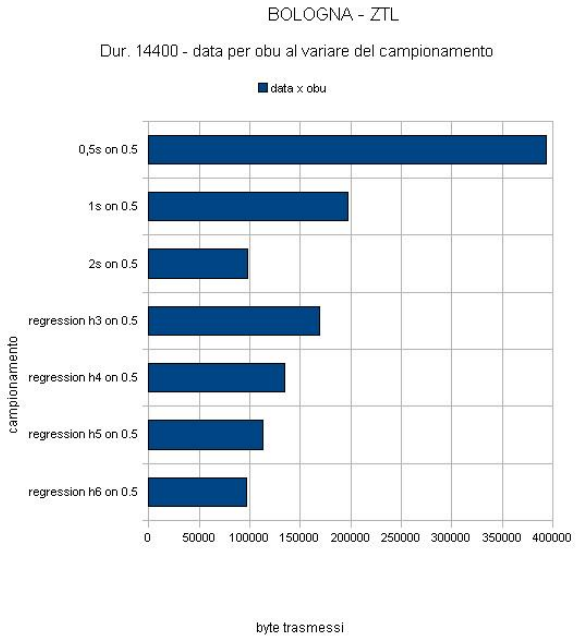

Figura 5.71: Grafico - Regress., Bologna 14400 (ztl on 0,5), dim. dati per OBU

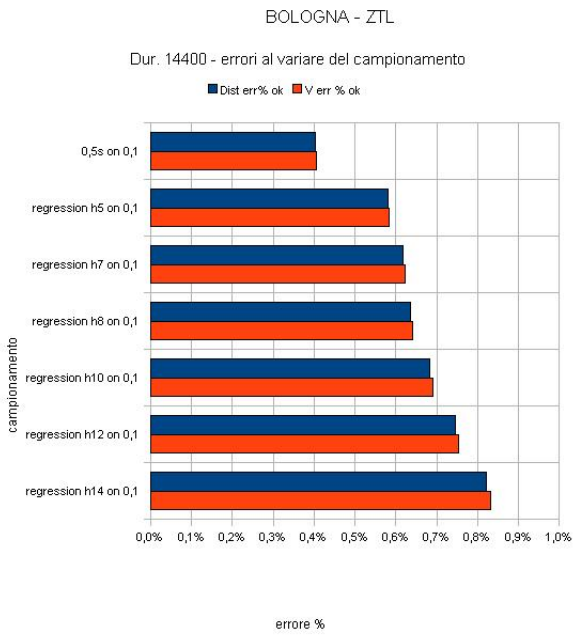

Figura 5.72: Grafico - Regression, Bologna 14400 (ztl on 0,1), errori commessi

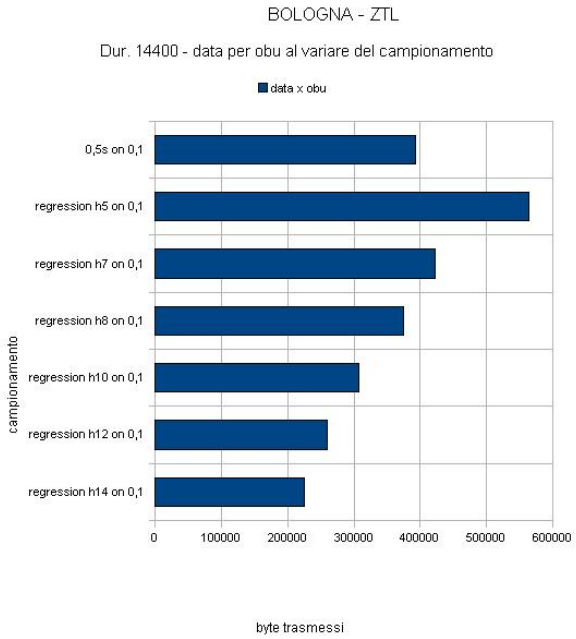

Figura 5.73: Grafico - Regress., Bologna 14400 (ztl on 0,1), dim. dati per OBU

Note e considerazioni generali dall'analisi effettuata

Da tutte le analisi effettuate sulla tecnica *Linear Regression*, sia su scenari con campionamento base a 0,1 secondi che con campionamento base a 0,5 secondi, nella quasi totalità dei casi, all'aumentare della dimensione della *history*, aumenta anche l'errore associato sia per la distanza che per la velocità. Parallelamente si può anche notare il *trade-off* dati inviati - errore commesso, infatti, se da una parte si ha il peggioramento delle stime dall'altro si ha un risparmio di comunicazione.

La crescita dell'errore, relativo sia a distanza che velocità, non segue un andamento lineare, infatti al raddoppiare del tempo di campionamento, rispettivamente, l'errore segue un andamento iperbolico, particolarmente visibile nel grafico 5.72 e in misura minore in tutti gli altri grafici dello stesso tipo riportati in questa sezione.

I tassi di errore, risultati praticamente sempre inferiori all'1% nei casi migliori, quindi particolarmente bassi, si prestano a tutti i tipi di servizi richiesti nei requisiti.

Da considerare che non sempre il risultato migliore, ottenuto in termini di errore, è pervenuto da lunghezza dell'*history* minima, infatti, come visibile nella tabella di confronto 5.58 il campionamento che ha dato risultati migliori è stato quello con lunghezza 5.

A differenza della tecnica *Simple time sampling*, dove spesso non sarà possibile inviare dati con campionamento a 0,5 secondi, sia per questioni di latenza che di banda, con la tecnica *Linear regression* il problema non si pone in quanto il *buffering* è automatico e dipendente dalla lunghezza stessa dell'*history*. Questo non ridurrà la banda aumentando l'errore come nel caso di *Simple time sampling*, ma ridurrà la banda mantenendo praticamente invariato il tasso di errore.

5.3.8 Note e deduzioni conclusive

Dalle varie analisi svolte, relative alle tecniche *communication-saving* implementate e collaudate, con particolare riferimento alle sezioni conclusive di ognuna, si riportano le seguenti note e conclusioni relative all'analisi V2I:

- Un campionamento ad intervalli più brevi possibili fornisce quasi sempre risultati migliori in termini di riduzione dell'errore sulla distanza e sulla velocità.

- Da stime riguardanti il traffico di dati totale delle OBU verso la centrale, con numero pari a circa un milione, che inviano dati campionati a 0,5 secondi ogni 0,5 secondi, si ottiene un flusso di dati di circa 0,8 MB/s pari a circa 2,83 GB all'ora e 68 GB al giorno. Considerando queste stime molto realistiche, considerando che non solo questo tipo di informazioni devono essere inviate dall'OBU e considerando il numero crescente di quest'ultime, si può affermare con certezza che le tecniche *communication-saving* sono o diverranno presto una necessità.

- Da tutte le analisi effettuate sulla tecnica *simple time sampling*, nella quasi totalità dei casi, all'aumentare del tempo intercorso tra un invio e un altro, aumenta anche l'errore associato sia per la distanza che per la velocità. Parallelamente si può anche notare il *trade-off* dati inviati - errore commesso, infatti, se da una parte si ha il peggioramento delle stime dall'altro si ha un risparmio di comunicazione. La crescita dell'errore, relativo sia a distanza che velocità, non segue un andamento lineare, infatti al raddoppiare del tempo di campionamento, rispettivamente, l'errore segue un andamento iperbolico. I tassi di errore ottenuti particolarmente bassi, si prestano a tutti i tipi di servizi richiesti nei requisiti. Da considerare che con questa tecnica non sempre il risultato migliore in termini di errore, è pervenuto da campionamento minimo a 0,5 secondi. Sia per questioni di latenza che di banda, con questa tecnica non sarà possibile inviare dati ogni 0,5 secondi, di conseguenza o si adotteranno strategie di *buffering*, impattando solo sulla banda, oppure si dovrà spesso scegliere un campionamento a frequenza inferiore, come ad esempio 1, 2 o 4 secondi, aumentando il tasso di errore, ma riducendo sia la banda sia la frequenza trasmissiva.

- Per quanto riguarda il campionamento effettuato con *simple space sampling*, nei campionamenti più fini l'errore sulla velocità è notevolmente superiore a quello sulla distanza, mentre man mano si cresce nell'aumentare la distanza di campionamento, anche l'errore sulla velocità si conforma a quello sulla distanza. Le ragioni di questo fenomeno vanno ricercate principalmente nella dipendenza tra velocità e frequenza di trasmissione ($\delta V = \delta s/\delta t$). In definitiva, da quanto dedotto, la tecnica *simple space sampling* dovrebbe essere utilizzata per ridurre i continui campionamenti della tecnica *simple time sampling* quando il veicolo viaggia a velocità particolarmente ridotte, come ad esempio nelle code del traffico cittadino, facendo risparmiare sia sulla banda che sul numero di comunicazioni.

- Valutando le varie tecniche *deterministic information-need*, per quanto riguarda la "Versione sperimentale con history", gli errori sulla distanza risultano in linea con le altre tecniche analizzate, mentre gli errori sulla velocità risultano molto più marcati; l'utilizzo di questa tecnica è da scartare per i servizi che richiedono precisione, ma potrebbe risultare utile per individuare i punti di rallentamento del veicolo (stop, code a tratti, etc.), anche se a tal scopo la versione *off-line* risulta migliore sotto gli aspetti di precisione e carico computazionale. L'utilizzo di quest'ultima tecnica è quindi da ritenersi fattibile sia per individuare i punti di rallentamento del veicolo sia da affiancare via *and* e *or* ad altre tecniche *communication-saving*.

- Per quanto riguarda la tecnica *Map-based sampling*, prendendo come riferimento i risultati ottenuti dalla tecnica *simple time sampling*, emerge che gli errori sono decisamente superiori a parità di dati inviati. L'utilizzo migliore che si potrebbe fare di questa tecnica riguarderebbe la ricostruzione dell'itinerario percorso, infatti, se la centrale possedesse la stessa mappa presente sull'OBU, quest'ultima potrebbe ricostruire il percorso esatto effettuato con una quantità di dati irrisoria ed un errore sulla distanza minimo.

- Dall'analisi dei risultati relativi a questa tecnica (*Simple regression*), si può dedurre che, considerando anche il *trade-off* dati inviati - errore commesso, vi è stato un lieve peggioramento rispetto alla tecnica *simple time sampling*, anche se, avendo un campionamento base inferiore e selezionando determinati parametri, si ottengono risultati in linea sia dal punto di

vista degli errori che del numero di dati inviati. I discreti risultati ottenuti hanno quindi indirizzato allo sviluppo della vera e propria tecnica di regressione *Linear Regression*.

- Da tutte le analisi effettuate sulla tecnica *Linear Regression* è emerso che, nella quasi totalità dei casi, all'aumentare della dimensione della *history*, aumenta anche l'errore associato sia per la distanza che per la velocità. Parallelamente si può anche notare il *trade-off* dati inviati - errore commesso. I tassi di errore, risultati praticamente sempre inferiori all'1% nei casi migliori, quindi particolarmente bassi, si prestano a tutti i tipi di servizi richiesti nei requisiti. Da considerare che non sempre il risultato migliore, ottenuto in termini di errore, è pervenuto da lunghezza dell'*history* minima. A differenza della tecnica *Simple time sampling*, dove spesso non sarà possibile inviare dati con campionamento a 0,5 secondi, sia per questioni di latenza che di banda, con la tecnica *Linear regression* il problema non si pone in quanto il *buffering* è automatico e dipendente dalla lunghezza stessa dell'*history*. Questo non ridurrà la banda aumentando l'errore come nel caso di *Simple time sampling*, ma ridurrà la banda mantenendo praticamente invariato il tasso di errore.

5.4 Analisi V2V

In questa sezione saranno riportate tutte le informazioni significative, sia sottoforma di grafico che di tabella, al fine di stimare la necessaria quantità percentuale di veicoli necessari per rendere possibile un efficiente utilizzo delle comunicazioni *Vehicle to Vehicle* (V2V) e per soddisfare, nel modo più completo possibile, gli ulteriori obiettivi proposti nella sezione 1.2.2.

In particolare, nella sezione 5.4.1 sono riportate le analisi effettuate variando il campionamento base; nella sezione 5.4.2 si studiano i dati ottenuti dalla simulazione al variare dello scenario, mentre nella sezione 5.4.3 si analizzano le variazioni registrate al crescere del numero di OBU in uno scenario fissato; nella sezione 5.4.4 si presentano le analisi ottenute variando la portata dell'antenna WiFi, infine, nella sezione 5.4.5 si presentano le note e le deduzioni ottenute da queste analisi.

Si precisa inoltre che le prove sperimentali sono state eseguite sui diversi scenari della sezione 5.1 con durate differenti di simulazione, da dieci minuti a quattro ore, con i diversi parametri di cui alla sezione 5.2.1. Le tabelle riportate, per eventuali problemi di grafica, potrebbero risultare suddivise in parti (Es. parte 1 di 4, ... , parte 4 di 4), ma sono da intendersi come su una sola riga.

5.4.1 Analisi al variare del campionamento base

Dall'analisi dei dati contenuti nelle tabelle 5.63 e 5.64, e dai relativi grafici 5.74, 5.75 e 5.76 di seguito presentati, si deduce che:

- salvo il caso dello scenario Beijing, la dimensione media del gruppo rimane pressoché invariata, con differenze nell'ordine della seconda cifra decimale
- la variazione del gruppo, ovvero il numero di veicoli che entrano ed escono dal gruppo per ogni secondo, sale con il campionamento base più fine
- come diretta conseguenza dei punti precedenti (il dato è derivato come visibile in sezione 5.2.2), la permanenza media dell'OBU all'interno del gruppo diminuisce con il campionamento base più fine.

Come da attese, il tipo di campionamento non influisce particolarmente sulla dimensione dei gruppi, dato che, considerando in questo caso una portata di 100 metri dell'antenna WiFi, la distanza percorsa in 0,1 o 0,5 secondi, pur risentendo di un'eventuale alta velocità, difficilmente impatterà su distanze simili. Un esempio di ciò potrebbe derivare da un veicolo che viaggia a 50 Km/h che campionato a 0,1 secondi avrebbe sensibilità massima sulla distanza di 1,4 metri per step, mentre campionato a 0,5 secondi questa distanza aumenterebbe di 5 volte, ovvero 6,9 metri per step. Considerando le difficoltà a procedere con campionamenti troppo fini e tenendo conto che le velocità medie registrate difficilmente superano i 50Km/h, si preferisce, nelle fasi successive, considerare campionamenti base a 0,5 secondi.

Per quanto riguarda la variazione del gruppo, si constata che vi sono molti più cambiamenti se l'OBU può entrare/uscire dal gruppo ogni 0,1 secondi piuttosto che ogni 0,5, come da esempio[4]:

istante	gruppo		istante	gruppo
0,5	0,1,2,3		0,5	0,1,2,3
			0,6	0,1,2
			0,7	0,1,3
			0,8	0,1,2
			0,9	0,1,3
1	0,1,3		1	0,1,3

[4]Nel caso a sinistra, con campionamento a 0,5 secondi, le variazioni sono 1, mentre nel caso a destra, con campionamento a 0,1 secondi, le variazioni sono 4; l'intervallo temporale è lo stesso.

Tabella 5.63: Dati V2V al variare del campionamento base - Parte 1 di 2

scenario	sim time (s)	camp (s)	obu tot	grp len
roma	600	0,5	1739	3,25
roma	600	0,1	1739	3,23
beijing	1200	0,5	5728	29,98
beijing	1200	0,1	5728	26,36
bologna	600	0,5	777	10,46
bologna	600	0,1	780	10,53
bologna (ztl)	14400	0,5	51	3,16
bologna (ztl)	14400	0,1	51	3,23

Tabella 5.64: Dati V2V al variare del campionamento base - Parte 2 di 2

grp changeXs	v in grp (s)	step life	tot life (ms)
0,21	18,83	150,29	75145,5
0,49	9,16	752,58	75258,5
0,16	176,69	203,06	101530,6
0,7	39,27	967,74	96775,8
0,81	18,85	689,33	344665,4
1,54	13	3449,22	344922,4
0,42	7,9	18585,35	9292676,47
0,58	5,94	92926,88	9292688,3

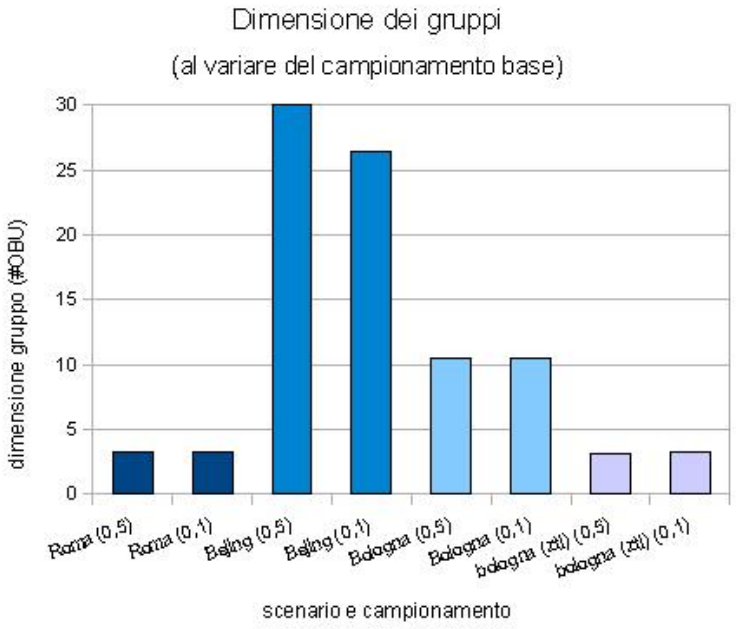

Figura 5.74: Grafico - Dimensione gruppo, var. campionamento base

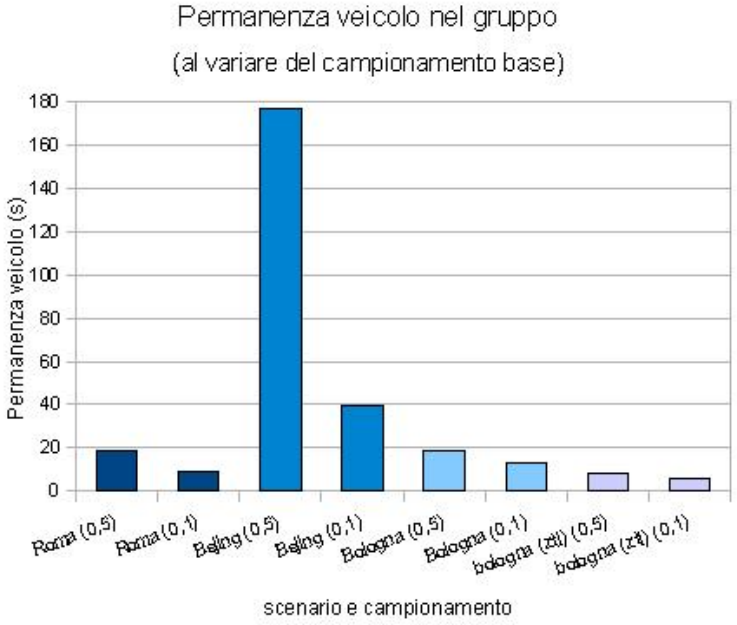

Figura 5.75: Grafico - Permanenza veicolo, var. campionamento base

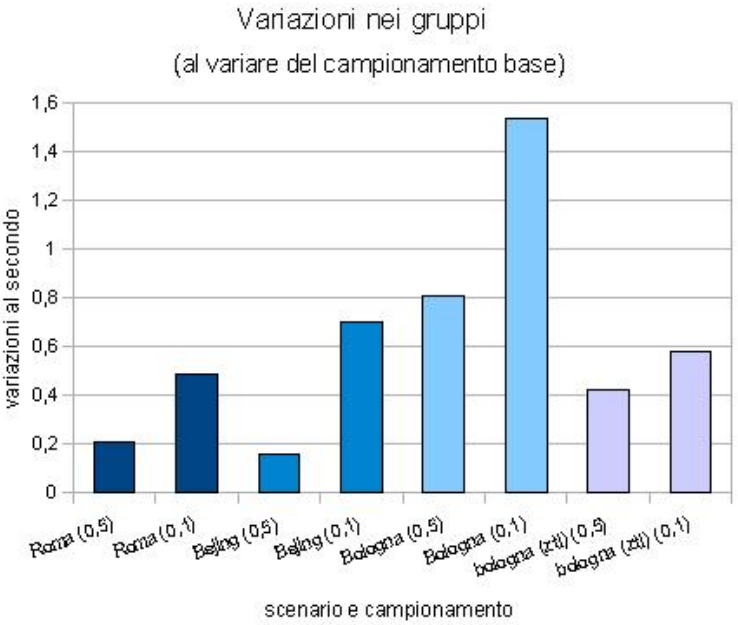

Figura 5.76: Grafico - Variazione gruppo, var. campionamento base

5.4.2 Analisi al variare dello scenario

Dall'analisi dei dati contenuti nelle tabelle 5.65 e 5.66, e dai relativi grafici 5.77, 5.78 e 5.79 di seguito presentati, si deduce che:

- la dimensione media del gruppo diminuisce all'aumentare del tempo di simulazione nello scenario di Roma
- gli incroci trafficati, le colonne e il basso numero di strade ortogonali percorribili, permettono l'aumento della dimensione dei gruppi
- gli scenari del centro, come Bologna (standard), e le relative basse velocità, permettono anch'essi l'aumento della dimensione dei gruppi.

Pur non avendo molti parametri a disposizione per una più accurata indagine, e non potendo simulare un'intera autostrada, una città di notevoli dimensioni o anche un centro storico con più di 20 semafori, causa le limitazioni del simulatore di traffico (vedi sezione 5.1), si presume comunque che le deduzioni fatte siano discretamente attendibili, soprattuto perché ricavate in modo empirico.

Tabella 5.65: Dati V2V al variare dello scenario - Parte 1 di 2

scenario	sim time (s)	camp (s)	obu tot	grp len
roma	600	0,5	1739	3,25
roma	3600	0,5	10666	1,43
beijing	1200	0,5	5728	29,98
toll plaza	3600	0,5	7902	3,2
bologna	600	0,5	777	10,46
bologna (RID)	7200	0,5	137	3,94
bologna (ZTL)	14400	0,5	51	3,16

Tabella 5.66: Dati V2V al variare dello scenario - Parte 2 di 2

grp changeXs	v in grp (s)	step life	tot life (ms)
0,21	18,83	150,29	75145,5
0,04	47,74	164,91	82454,2
0,16	176,69	203,06	101530,6
0,06	60,65	306,37	153186,5
0,81	18,85	689,33	344665,4
0,61	27,78	6866,53	3433262,8
0,42	7,9	18585,35	9292676,47

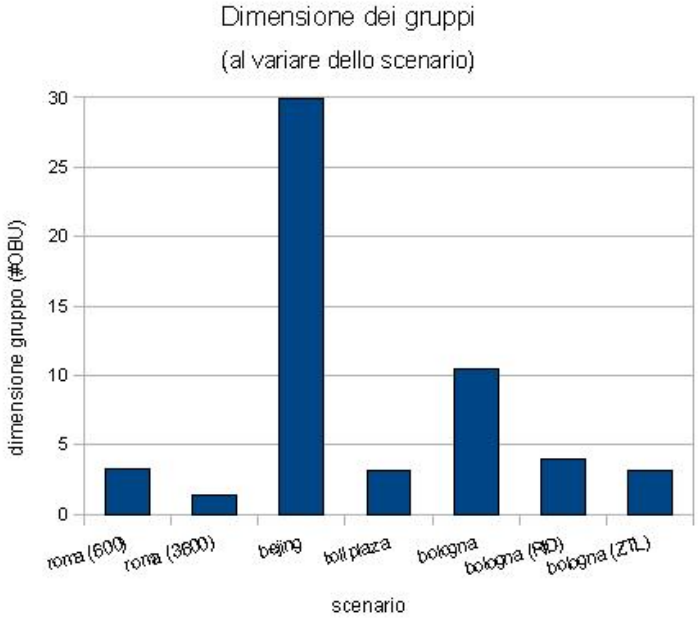

Figura 5.77: Grafico - Dimensione gruppo, var. scenario

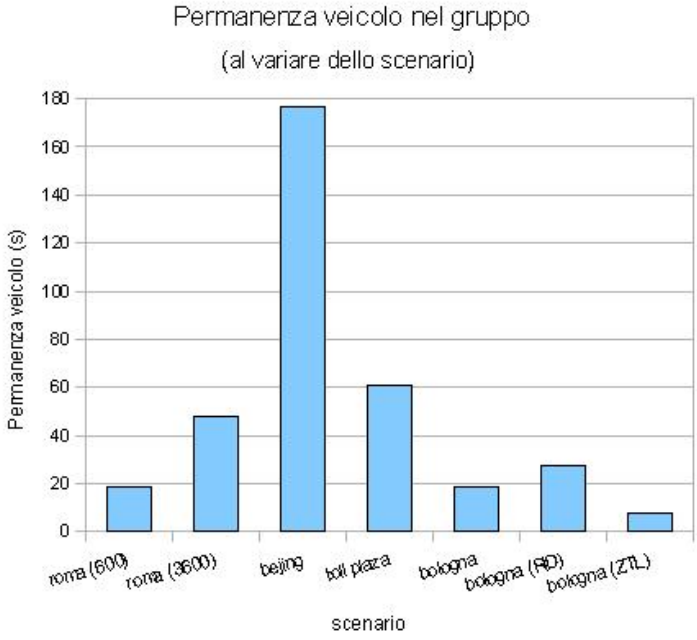

Figura 5.78: Grafico - Permanenza veicolo, var. scenario

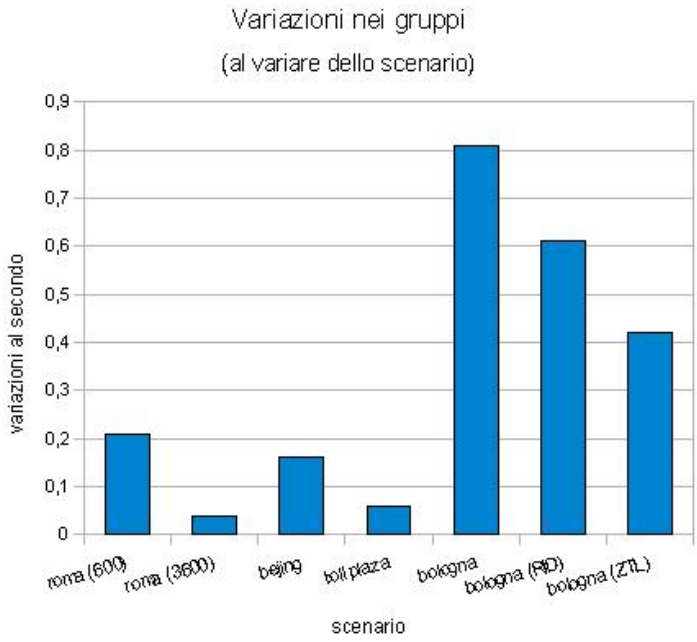

Figura 5.79: Grafico - Variazione gruppo, var. scenario

5.4.3 Analisi al variare del numero di OBU

Lo scenario di analisi per questa simulazione è stato Bologna in modalità ZTL con numero di OBU variabile tra 25, 51, 80 e 108. In pratica, partendo un numero minimo di OBU (25) è stata eseguita la simulazione sullo stesso scenario (Bologna ZTL) con egual tempo (14400 secondi) con numero di OBU aumentato di 2, 2,5 e 3 volte rispetto il valore iniziale in modo da ottenere i dati seguenti.
Dal grafico 5.80 di seguito presentato, dove sull'ordinata è riportata la dimensione del gruppo il cui valore è pari all'ampiezza dell'intervallo disegnato, mentre sull'ascissa è riportato l'identificativo del gruppo formato dall'OBU con identificativo eguale, si è potuto dedurre che l'andamento della dimensione dei gruppi segue un comportamento standard. Questo comportamento è caratterizzato da tassi di entrata/uscita, dipendenti dal numero di veicoli coinvolti nella simulazione, ma sempre collegati ad una curva ben definita. La dimensione del gruppo altresì aumenta con l'aumentare del numero di OBU coinvolte.
Sempre dalle tabelle di dati relative al grafico 5.80, non riportate in quanto occupanti numerose pagine (anche più di quindici con formati ridotti), si sono potuti estrarre i dati delle altre tabelle presenti in questa sezione, infatti, dall'analisi dei dati contenuti nella tabella 5.67, e dai relativi grafici 5.81, 5.82 e 5.83 di seguito presentati, si deduce in modo più chiaro che:

- la dimensione media dei gruppi aumenta in modo logaritmico[5] all'aumentare del numero di OBU coinvolte
- la variazione media dei gruppi aumenta in modo molto simile alla dimensione media dei gruppi
- il tempo medio di permanenza di un veicolo in un gruppo diminuisce, fino a stabilizzarsi, all'aumentare del numero di OBU coinvolte
- la vita media dei gruppi (equivalente alla vita media delle singole OBU) diminuisce all'aumentare del numero di OBU.

Dalle prime tre deduzioni fatte si può asserire che i dati di ogni scenario convergono ad un risultato decisamente stabile e misurabile, mentre l'ultima deduzione evidenzia solo il fatto che, in questa simulazione di traffico, le OBU che "compaiono" dopo "vivono" per meno tempo rispetto alle altre.

[5] in modo simile alla curva del logaritmo naturale per valori superiori a 1

Tabella 5.67: Dati V2V al variare del numero di OBU

obu num	grp len	grp changeXs	v in grp (s)	step life	life
0	0	0	0	0	0
25	2,69	0,28	11,78	26165,8	13082900
51	3,82	0,54	7,18	23692,45	11846225,49
81	4,63	0,67	6,99	20576,72	10288358,02
108	4,87	0,68	7,36	17943,67	8971833,33

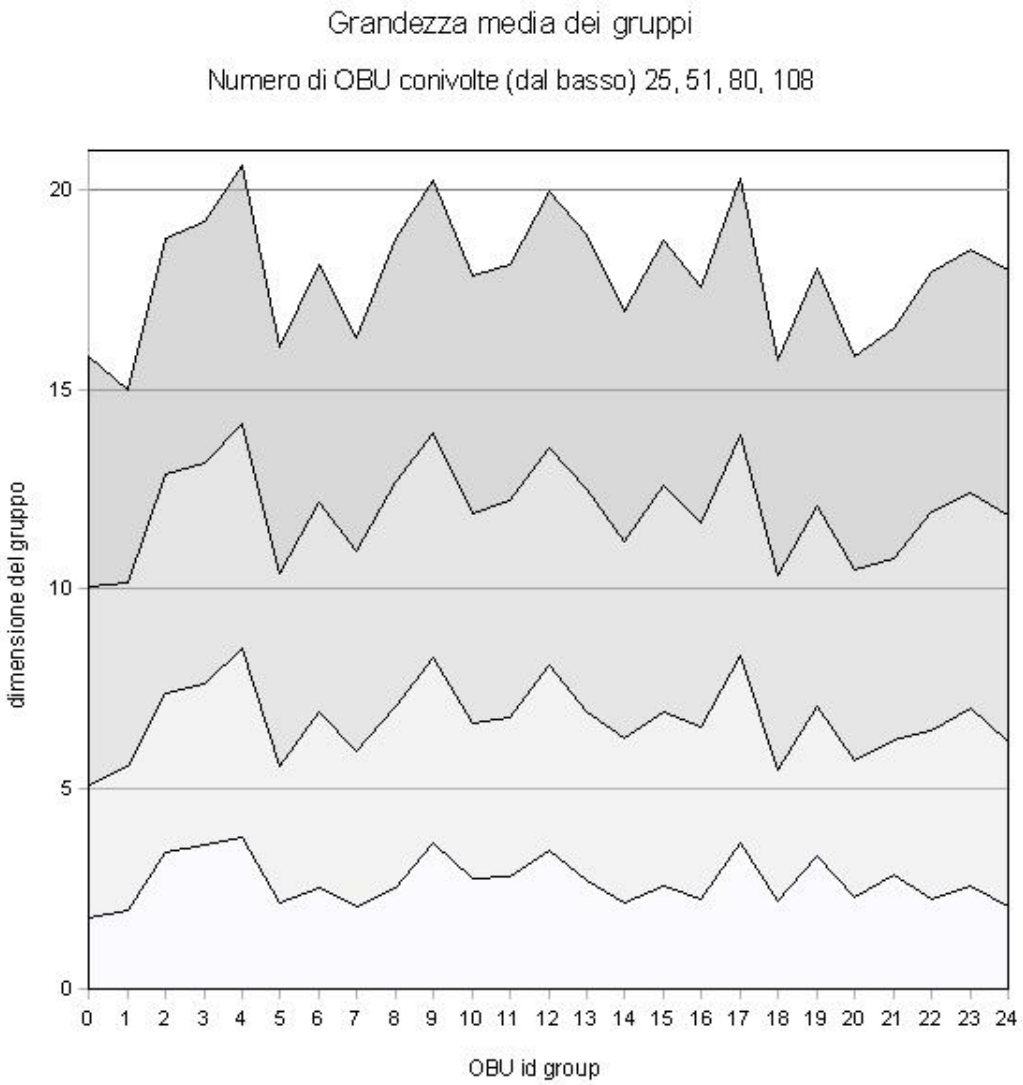

Figura 5.80: Grafico - Dimensione dei singoli gruppi per fasce

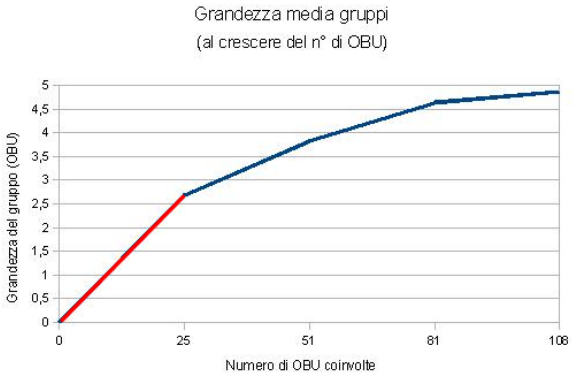

Figura 5.81: Grafico - Grandezza media dei gruppi, aumento n° OBU

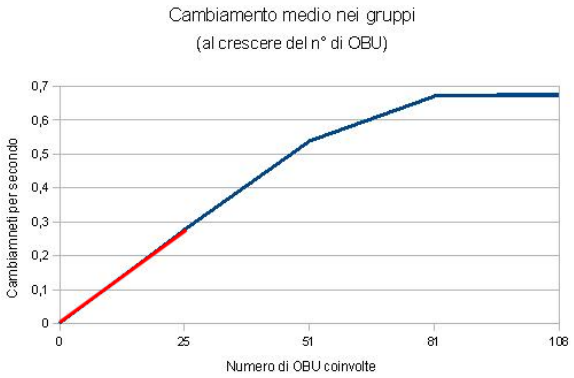

Figura 5.82: Grafico - Cambiamento medio nei gruppi, aumento n° OBU

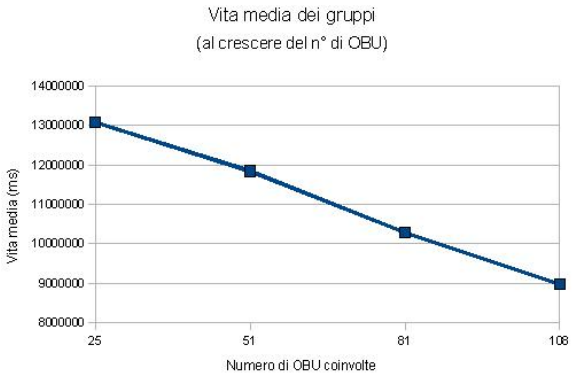

Figura 5.83: Grafico - Vita media dei gruppi, aumento n° OBU

5.4.4 Analisi al variare della portata WiFi

Dall'analisi dei dati contenuti nelle tabelle 5.68 e 5.69, e dai relativi grafici 5.84, 5.85, 5.86, 5.87, 5.88 e 5.89 di seguito presentati, si deduce che, dipendentemente dalla portata dell'antenna WiFi:

- i cambiamenti all'interno dei gruppi sono costanti
- la dimensione del gruppo aumenta in modo lineare
- di conseguenza alle prime due deduzioni, anche il tempo di permanenza del veicolo all'interno del gruppo cresce in modo lineare.

La prima deduzione fatta, ovvero che i cambiamenti all'interno dei gruppi sono costanti, era prevedibile in quanto il raggio di copertura dell'antenna non influenza il tasso di entrata/uscita del veicolo, che, pur dovendo percorrere una distanza maggiore per entrare od uscire dal gruppo, lo farà egualmente, ma in maggior tempo, come riportato nell'ultima deduzione.

La seconda considerazione, che riguarda l'aumento lineare della dimensione del gruppo all'aumentare della portata dell'antenna WiFi, risulta assolutamente la più importante tra quelle fatte. Certo la pendenza della retta dell'aumento non è equivalente a quella della portata, ma, a meno di una costante moltiplicativa, l'andamento è lo stesso.

Tabella 5.68: Dati V2V al variare della portata WiFi - Parte 1 di 2

scenario	sim time (s)	camp (s)	obu tot	wifi range
beijing	1200	0,5	5728	100
roma	3600	0,5	10666	100
toll plaza	3600	0,5	7902	100
beijing	1200	0,5	5728	120
roma	3600	0,5	10666	120
toll plaza	3600	0,5	7902	120
beijing	1200	0,5	5728	140
roma	3600	0,5	10666	140
toll plaza	3600	0,5	7902	140
beijing	1200	0,5	5728	250
roma	3600	0,5	10666	250
toll plaza	3600	0,5	7902	250

Tabella 5.69: Dati V2V al variare della portata WiFi - Parte 2 di 2

grp len	grp changeXs	v in grp (s)	step life	tot life (ms)
29,98	0,16	176,69	203,06	101530,6
1,43	0,04	47,74	164,91	82454,2
3,2	0,06	60,65	306,37	153186,5
33,12	0,17	195,8	203,06	101530,6
1,53	0,04	48,26	164,91	82454,2
3,63	0,06	64,67	306,37	153186,5
35,42	0,17	209,66	203,06	101530,6
1,64	0,04	49,33	164,91	82454,2
4,05	0,06	68,46	306,37	153186,5
40,87	0,17	241,82	203,06	101530,6
2,24	0,04	60,39	164,91	82454,2
6,2	0,07	91,1	306,37	153186,5

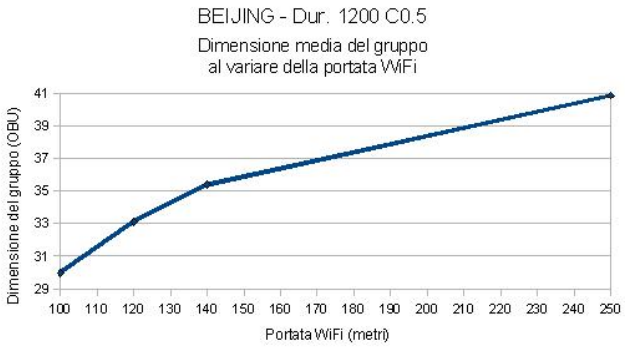

Figura 5.84: Grafico - Beijing, dimensione gruppo, portata WiFi

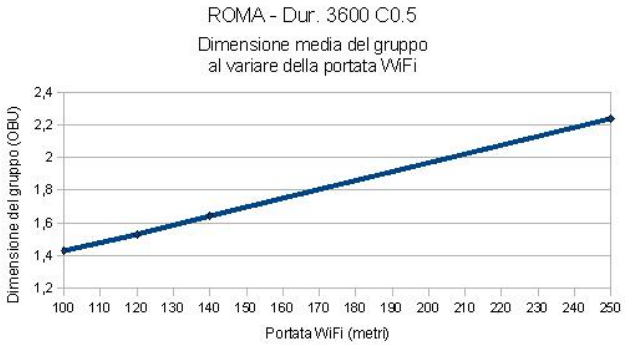

Figura 5.85: Grafico - Roma, dimensione gruppo, portata WiFi

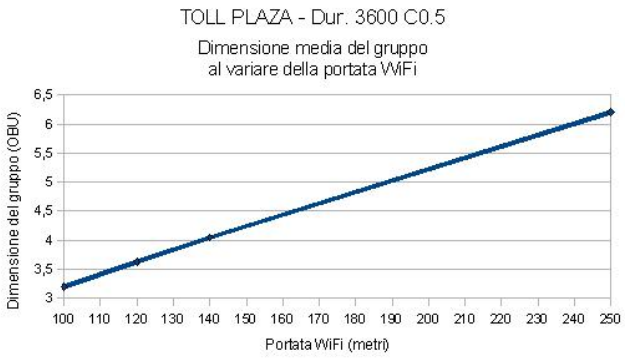

Figura 5.86: Grafico - Toll plaza, dimensione gruppo, portata WiFi

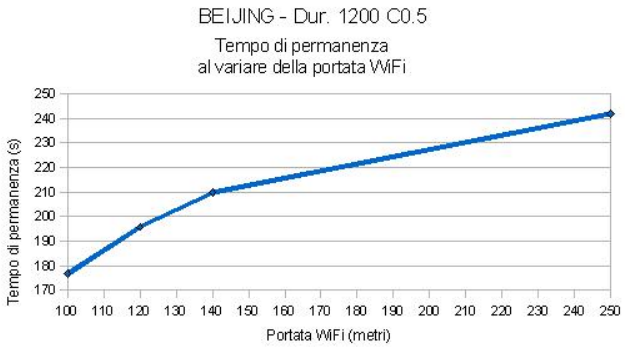

Figura 5.87: Grafico - Beijing, permanenza veicolo, portata WiFi

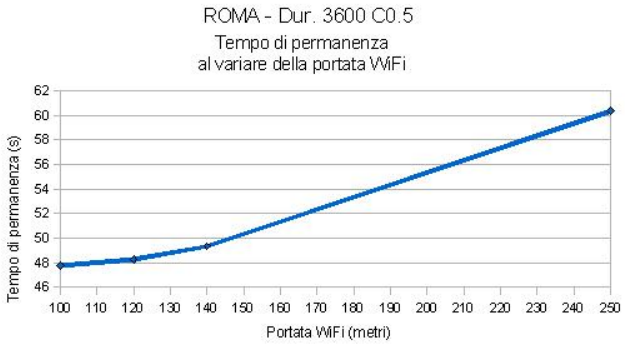

Figura 5.88: Grafico - Roma, permanenza veicolo, portata WiFi

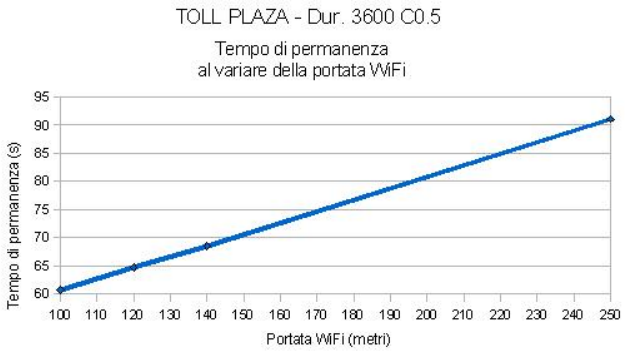

Figura 5.89: Grafico - Toll plaza, permanenza veicolo, portata WiFi

5.4.5 Note e deduzioni conclusive

Dalle varie analisi svolte, relative alla creazione di gruppi di comunicazione tra OBU, con particolare riferimento alle sezioni conclusive di ognuna, si riportano le seguenti note e conclusioni relative all'analisi V2V:

- per quanto riguarda l'analisi al variare del campionamento base, la dimensione media del gruppo rimane pressoché invariata, con differenze nell'ordine della seconda cifra decimale, mentre la variazione del gruppo, ovvero il numero di veicoli che entrano ed escono dal gruppo per ogni secondo, sale con il campionamento base più fine; come diretta conseguenza dei punti precedenti la permanenza media dell'OBU all'interno del gruppo diminuisce con il campionamento base più fine. Per quanto riguarda la variazione del gruppo, si constata che vi sono molti più cambiamenti se l'OBU può entrare od uscire dal gruppo ad intervalli più brevi.

- Variando lo scenario emergono differenze nella creazione dei gruppi legate alla conformazione dello stesso, in particolare, gli incroci trafficati, le colonne, il basso numero di strade ortogonali percorribili, le zone del centro e le basse velocità, permettono l'aumento della dimensione dei gruppi.

- Il numero di veicoli circolanti influisce sull'andamento della dimensione dei gruppi, che segue un comportamento standard. Questo comportamento è caratterizzato da tassi di entrata ed uscita, dipendenti dal numero di veicoli coinvolti nella simulazione, ma sempre collegati ad una curva ben definita. La dimensione dei gruppi altresì aumenta in modo simil-logaritmico con l'aumentare del numero di OBU coinvolte, mentre la variazione media dei gruppi aumenta in modo molto simile alla dimensione media dei gruppi; infine, il tempo medio di permanenza di un veicolo in un gruppo diminuisce, fino a stabilizzarsi, sempre all'aumentare del numero di OBU coinvolte. Da queste deduzioni fatte si può asserire che i dati di ogni scenario convergono ad un risultato decisamente stabile e misurabile.

- Dipendentemente alla crescita della portata dell'antenna WiFi, i cambiamenti all'interno dei gruppi sono costanti, mentre la dimensione del gruppo aumenta in modo lineare e di conseguenza anche il tempo di permanenza del veicolo all'interno del gruppo cresce in modo lineare. Il risultato per cui i cambiamenti all'interno dei gruppi sono costanti era prevedibile, in

quanto il raggio di copertura dell'antenna non influenza il tasso di entrata ed uscita dei veicoli, che, pur dovendo percorrere una distanza maggiore per entrare od uscire dal gruppo, lo faranno egualmente, ma in maggior tempo. Per quanto riguarda invece l'aumento lineare della dimensione del gruppo all'aumentare della portata dell'antenna WiFi, questo risulta assolutamente di importanza strategica, non per la questione del solo aumento che era assolutamente prevedibile, ma per la linearità di quest'ultimo.

La longevità di un veicolo all'interno di un gruppo, misurata in tutte le simulazioni analizzate e sempre compresa tra 6 e 250 secondi, ha media intorno ai 110 secondi nell'ultima analisi con scenari quasi completamente reali.

La dimensione media dei gruppi, misurata in numero di OBU, sempre superiore alle 1,5 unità circa, con punte intorno alle 41 unità, nell'ultima analisi con scenari quasi completamente reali ha una media di circa 14 unità.

La combinazione di queste ultime deduzioni, in corrispondenza con le altre osservazioni fatte e l'analisi empirica della realtà, porta a concludere che il raggruppamento delle OBU in *cluster*, previa strategie V2V, diminuisce in media di 14 volte il numero di comunicazioni verso il centro di controllo.

Lo scambio delle informazioni intra veicoli risulta altresì attuabile al fine di migliorare anche la circolazione, considerando che la maggiore necessità di ciò è in corrispondenza di traffico sostenuto, ovvero di più OBU circolanti.

Conclusioni e sviluppi futuri

Durante lo svolgimento del progetto sono stati portati a termine tutti gli obiettivi prefissati e sono state sviluppate delle funzionalità aggiuntive che hanno permesso l'analisi di tecniche *communication-saving* maggiormente avanzate, al fine di ricercare le migliori da applicare nel contesto reale, in particolare:

- il numero delle comunicazioni V2I è stato ridotto utilizzando tecniche *communication-saving* senza compromettere il *trade-off* dati inviati - errore commesso;
- sono stati codificati, compressi e studiati i flussi di dati provenienti dalle OBU verso il centro di controllo, al fine di conoscere le potenziali quantità di dati con cui si avrà a che fare nella realtà;
- dopo un'attenta analisi sono state individuate le migliori strategie *communication-saving* utilizzabili nei contesti reali richiesti;
- sono state studiate ed implementate nuove tecniche *communication-saving* al fine di apportare miglioramenti a quelle inizialmente proposte;
- l'implementazione delle tecniche descritte è stata svolta in modo da rendere agevole la portabilità del codice sull'architettura reale;
- sono state confrontate le diverse tecniche al fine di individuare pregi e difetti di ognuna;
- sono stati analizzati alcuni scenari reali per mostrare le quantità necessarie di veicoli che rendono possibile un efficiente utilizzo delle comunicazioni V2V;
- sono state definite strategie e politiche di clustering tra veicoli;

- è stata effettuata la stima della riduzione del carico sulle comunicazioni V2I, derivata dell'utilizzo di comunicazioni V2V.

Considerando che tutto il progetto è stato svolto in un ottica di estensione e usabilità orientate agli sviluppi futuri del progetto Pegasus, sono di seguito presentate alcune proposte. In particolare, per quanto riguarda le tecniche *communication-saving*:

- queste si dovrebbero testare su scenari molto più estesi, con dati sul traffico aggiornati e forniti da enti certificati, al fine di validare definitivamente le analisi svolte;
- la codifica sviluppata avvantaggia un'ulteriore codifica in stile Huffmann, la quale potrebbe essere implementata insieme ad altre tecniche di compressione, al fine di ridurre i dati inviati alla centrale;
- sullo stile della tecnica *Linear Regression*, la quale ha ottenuto ottimi risultati, potrebbero essere sviluppate tecniche più complesse quali *Multivariate adaptive regression splines* [16] o *Segmented regression* [34];

Per quanto riguarda invece le possibili estensioni delle funzionalità dell'OBU:

- si potrebbero sviluppare ulteriormente gli algoritmi per il *clustering* V2V al fine di ottimizzare le comunicazioni ed i gruppi formati;
- si potrebbe studiare una tecnica di auto-selezione adattativa, in modo da ottimizzare l'utilizzo delle tecniche *communication-saving*; la validità di questo algoritmo sarebbe verificabile direttamente dall'OBU stessa, essendo questa a conoscenza dei dati inviati alla centrale.

La conclusione ultima di questa ricerca vedrebbe l'implementazione di quanto sviluppato in questo progetto sull'architettura reale di ogni OBU, a loro volta impiantate su veicoli realmente circolanti negli scenari urbani delle nostre città.

Ulteriori ringraziamenti

Dal mio punto di vista, l'inclusione dei ringraziamenti nella parte iniziale dei documenti è stata pensata per non dover sopperire all'elencazione di tutti coloro che meritano di essere ringraziati; detto ciò, preferisco citare chi effettivamente risulta nei miei pensieri in questo momento.

Il primo ringraziamento è sicuramente per i miei famigliari, senza i quali non avrei mai potuto iniziare e finire (con questo progetto) gli studi universitari. Un grazie di cuore a Giampaolo, Giuliana e Stefano. Grazie anche a Giuliano, Nicola, Lucia, Leonardo, Gabriella, Vittorio e Simone.

A questi ringraziamenti aggiungo anche Sandra, Daniele, Mara, Monica e Davide, i quali sono in qualche modo uniti al gruppo di prima.

Un particolare ringraziamento ai miei professori universitari, ad alcuni per aver condiviso con me le loro importanti conoscenze e ad altri, che oltre ad aver fatto ciò, mi hanno anche procurato un futuro lavorativo migliore. Un grande grazie a Michele, Riccardo, Luca, Mauro, Roberto, Giacomo, Mauro, Claudia, Alessandro, Paolo e Marco.

Grazie anche a tutti i miei compagni di corso e ai miei amici, sia per l'aiuto e la fiducia dati che per l'ottima la compagnia. Grazie Andrea, Alessandro, Christian, Chiara, Elisabetta, Elena, Enrico, Emmanuele, Enrico, Davide, Fabio, Francesco, Federico, Filippo, Francesco, Francesca, Giulia, Guido, Giacomo, Giovanni, Luca, Michele, Nicola, Luca, Nicolò, Manuele, Marco, Matteo, Rita, Roberto, Simone, Stefania, Stefano e Vincenzo.

Ringrazio infine tutti coloro i quali non ho avuto la pazienza di citare, ma che sanno di meritarsi i miei ringraziamenti. Grazie a tutti.

Capitolo 6

Appendice

6.1 Appendice 1: Bozza di algoritmo per il V2V

Si presenta una bozza di esempio di algoritmo per il V2V, includibile nel progetto al fine di testare il comportamento dei gruppi di OBU formati durante la simulazione. Si precisa che questo esempio non vuole riportare dettagli implementativi ma solo essere una guida di partenza per successivi sviluppi.

```
// function create_group() - [TBD]
// function join_group() - [TBD]
// function have_group() - [TBD]
// function get_group() - [TBD]
// function eventually_leave() - [TBD]

for( i=0; i < ( OBU_number - 1); i++ )
  for( j=0; j < OBU_number; j++ )
    if( Point.distance( OBU[i].getPosition() , OBU[j].
       getPosition() ) <= WiFi_range )
    {
      if( OBU[i].have_group() && (! OBU[j].have_group()) )
        OBU[j].join_group( OBU[i].get_group() );
      else
        if( OBU[j].have_group() && ( ! OBU[i].have_group()
           ) )
          OBU[i].join_group( OBU[j].get_group() );
        else
          if( (! OBU[j].have_group()) && (! OBU[i].
             have_group()) )
          {
            OBU[i].create_group( OBU[j].[TBD] );
            OBU[j].create_group( OBU[i].[TBD] );
          }
    }
    else{
      OBU[i].eventually_leave( OBU[j].get_group() );
      OBU[j].eventually_leave( OBU[i].get_group() );
    }
```

6.2 Appendice 2: Dati sulla compressione

Tramite la funzione di seguito riportata si è provato a comprimere i messaggi inviati dalle OBU verso la centrale (V2I); ciò che è risultato è stato un'aumento di dimensione, principalmente dovuto all'esigua lunghezza dei messaggi stessi e agli header introdotti dall'ulteriore codifica fatta dall'algoritmo di compressione. Da notare che l'algortimo di compressione è stato utilizzato variando tutti i parametri consentiti.

```
/**
 * Compress a string using Java ZIP compression.
 * @param in String to compress
 * @return Compressed string
 */
public static String Compress( String in )
{
    // Create the compressor with best level of compression
    Deflater compressor = new Deflater();

    // Eg. other paramters: BEST_COMPRESSION
    compressor.setLevel(Deflater.FILTERED);

    // Give the compressor the data to compress
    compressor.setInput(in.getBytes());
    compressor.finish();

    // Create an expandable byte array to hold the
       compressed data.
    // You cannot use an array that's the same size as the
       orginal because
    // there is no guarantee that the compressed data will
       be smaller than
    // the uncompressed data.
    ByteArrayOutputStream bos = new ByteArrayOutputStream(
       in.length());
```

```
    // Compress the data
    byte[] buf = new byte[1024];
    while (!compressor.finished())
    {
        int count = compressor.deflate(buf);
        bos.write(buf, 0, count);
    }

    try{
        bos.close();
    }
    catch( Exception e )
    {
        Test.Debug.log("Error in compression", e+"");
    }

    // Get the compressed data
    return bos.toString();
}
```

6.3 Appendice 3: Esempio di test

```
package Test;

/**
 * Testing GPSconverter class.
 *
 * @author Marcello Pietri - pietri.marcello@yahoo.it
 * @version 0.1
 */
public class Test_GPSconverter {

    public Test_GPSconverter(){

        Test.Debug.debug = true;
        Test.Debug.debug_forcelog = false;
        Test.Debug.debug_tmp = false;
        Test.Debug.debug_forcelog_tmp = false;

        test1();
        test2();
        test3();
        test4();
        // missing function from test5 to test12
    }

private void test1(){
Test.Debug.println("\n---\n");
Test.Debug.println( "STANDARD - Must be 0.366 km -->
"+OBU.GPS_2Dconverter.toKmetersDistance(46.06716, 11.12331,
    46.07033, 11.12462, false) );
        Test.Debug.println( "STANDARD - Must be 0.299 km
           -->
"+OBU.GPS_2Dconverter.toKmetersDistance(46.06833, 11.12388,
    46.06848, 11.12776, false) );

        Test.Debug.println("\n---\n");
```

```
        Test.Debug.println( "ACCURATE - Must be 0.366 km
           -->
"+OBU.GPS_2Dconverter.toKmetersDistance(46.06716, 11.12331,
    46.07033, 11.12462, true) );
        Test.Debug.println( "ACCURATE - Must be 0.299 km
           -->
"+OBU.GPS_2Dconverter.toKmetersDistance(46.06833, 11.12388,
    46.06848, 11.12776, true) );
Test.Debug.println("\n---\n");
}

private void test2(){
Test.Debug.println("\n---\n");
Test.Debug.println("Tour Eiffel - Colosseo : 1.109,96 Km (
   calcolata da Google Earth)");
        Test.Debug.println( "STANDARD - Must be 1109.96 km
           -->
"+OBU.GPS_2Dconverter.toKmetersDistance(48.857924,
   2.294026, 41.890164, 12.492346, false) );
        Test.Debug.println( "ACCURATE - Must be 1109.96 km
           -->
"+OBU.GPS_2Dconverter.toKmetersDistance(48.857924,
   2.294026, 41.890164, 12.492346, true) );
Test.Debug.println("\n---\n");
}

private void test3(){
Test.Debug.println("\n---\n");
Test.Debug.println( "ACCURATE - 0.00001 --> "+OBU.
   GPS_2Dconverter.toKmetersDistance(0, 0,
0.00001, 0, true) );
        Test.Debug.println( "ACCURATE - 0.00001 --> "+OBU.
           GPS_2Dconverter.toKmetersDistance(0, 0, 0,
0.00001, true) );
        Test.Debug.println( "ACCURATE - 0.00001 --> "+OBU.
           GPS_2Dconverter.toKmetersDistance(0.00001, 0,
0, 0, true) );
```

```
        Test.Debug.println( "ACCURATE - 0.00001 --> "+OBU.
           GPS_2Dconverter.toKmetersDistance(0, 0.00001,
0, 0, true) );
Test.Debug.println("\n---\n");
}

private void test4(){
Test.Debug.println("\n---\n");
Test.Debug.println( "ACCURATE - 0.0001 --> "+OBU.
   GPS_2Dconverter.toKmetersDistance(0.0001, 0, 0,
 0, true) );
        Test.Debug.println( "ACCURATE - 0.001 --> "+OBU.
           GPS_2Dconverter.toKmetersDistance(0.001, 0, 0,
0, true) );
        Test.Debug.println( "ACCURATE - 0.01 --> "+OBU.
           GPS_2Dconverter.toKmetersDistance(0.01, 0, 0, 0,
true) );
        Test.Debug.println( "ACCURATE - 0.1 --> "+OBU.
           GPS_2Dconverter.toKmetersDistance(0.1, 0, 0, 0,
true) );
        Test.Debug.println( "ACCURATE - 1 --> "+OBU.
           GPS_2Dconverter.toKmetersDistance(1, 0, 0, 0,
           true)
);
        Test.Debug.println( "ACCURATE - 10 --> "+OBU.
           GPS_2Dconverter.toKmetersDistance(10, 0, 0, 0,
true) );
Test.Debug.println("\n---\n");
}

// missing function from test5 to test12
}
```

Esempio di output:

```
STANDARD - Must be 0.366 km --> 0.3671005084413005
STANDARD - Must be 0.299 km --> 0.3001304303402162

---

ACCURATE - Must be 0.366 km --> 0.366747114329361
ACCURATE - Must be 0.299 km --> 0.2998415058508927

---

Tour Eiffel - Colosseo : 1.109,96 Km (calcolata da Google
   Earth)
STANDARD - Must be 1109.96 km --> 1110.6693165140118
ACCURATE - Must be 1109.96 km --> 1109.6001161509917

---

ACCURATE - 0.00001 --> 0.0011121237993707868
ACCURATE - 0.00001 --> 0.0011121237993707868
ACCURATE - 0.00001 --> 0.0011121237993707868
ACCURATE - 0.00001 --> 0.0011121237993707868

---

ACCURATE - 0.0001 --> 0.011121237993707868
ACCURATE - 0.001 --> 0.11121237993707868
ACCURATE - 0.01 --> 1.1121237993707866
ACCURATE - 0.1 --> 11.121237993707869
ACCURATE - 1 --> 111.21237993707868
ACCURATE - 10 --> 1112.1237993707869
```

Bibliografia

[1] G. Adomavicius and A. Tuzhilin. Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions. *IEEE TKDE*, 17(6):734–749, 2005.

[2] Arvind Arasu, Brian Babcock, Shivnath Babu, John Cieslewicz, Mayur Datar, Keith Ito, Rajeev Motwani, Utkarsh Srivastava, and Jennifer Widom. STREAM: The Stanford Data Stream Management System. In M. Garofalakis, J. Gehrke, and R. Rastogi, editors, *Data Stream Management: Processing High-Speed Data Streams*. Springer, 2007.

[3] D. Ayala, J. Lin, O. Wolfson, N. Rishe, and M. Tanizaki. Communication Reduction for Floating Car Data-based Traffic Information Systems. In *Proc. of Advanced Geographic Information Systems, Applications and Services*, pages 44–51, 2010.

[4] I. Botan, G. Alonso, P. M. Fischer, D. Kossmann, and N. Tatbul. Flexible and scalable storage management for data-intensive stream processing. In *Proc. of EDBT*, pages 934–945, 2009.

[5] R. Bruno, M. Conti, and E. Gregori. Mesh networks: commodity multihop ad hoc networks. *IEEE Communications Magazine*, 43(3):123–131, 2005.

[6] Co-founders: Paolo Tiberio, Federica Mandreoli, Riccardo Martoglia. Information Systems Group, Università degli studi di Modena e Reggio Emilia. Web site: http://www.isgroup.unimo.it.

[7] R. Ding and Q. Zeng. A Clustering-based Multi-channel Vehicle-to-Vehicle (V2V) Communication System. In *Proc. of ICUFN*, pages 83–88, 2009.

[8] Document Object Model (DOM). Web sites: http://www.w3.org/DOM/ http://www.w3.org/2003/01/dom2-javadoc/index.html.

[9] F. Mandreoli, R. Martoglia, W. Penzo, S. Sassatelli. Data Management Issues for Intelligent Transportation Systems . In *18th Italian Symposium on Advanced Database Systems (SEBD 2010)*, pages 198–209, 2010.

[10] Maresca M. Fornasa M., Zingirian N. Extensive GPRS Latency Characterization in Uplink Packet Transmission from Moving Vehicles. In *Vehicular Technology Conference, 2008. VTC Spring 2008. IEEE* , pages 2562–2566, 2008.

[11] S. Goel, T. Imielinski, and K. Ozbay. Ascertaining Viability of WiFi based Vehicle-to-Vehicle Network for Traffic Information Dissemination. In *Proc. of IEEE ITSC*, 2004.

[12] L. Golab and M. T. Özsu. Update-Pattern-Aware Modeling and Processing of Continuous Queries. In *Proc. of SIGMOD*, pages 658–669, 2005.

[13] The Google Geocoding API.
Web site:
http://code.google.com/intl/it-IT/apis/maps/documentation/geocoding/.

[14] GPRS from Wikipedia.
Web site: http://it.wikipedia.org/wiki/General_Packet_Radio_Service.

[15] C. Guestrin, P. Bodik, R. Thibaux, M. Paskin, and S. Madden. Distributed Regression: an Efficient Framework for Modeling Sensor Network Data. In *Proc. of IPSN*, pages 1–10, 2004.

[16] Friedman J. H. Multivariate Adaptive Regression Splines. *Annals of Statistics*, 19(1):1–67, 1991.

[17] Guy Steele James Gosling, Bill Joy and Gilad Bracha. *The Java Language Specification*. Sun Microsystems, third edition edition, 1996-2005. ISBN 0-321-24678-0.

[18] Java. Web site: http://www.java.com/it/about/.

[19] Easily convert between latitude/longitude, Universal Transverse Mercator (UTM) and Ordnance Survey (OSGB) references with Java using the Jcoord package. Web site: http://www.jstott.me.uk/jcoord.

[20] C. S. Jensen and R. T. Snodgrass. Temporal data management. *IEEE TKDE*, 11(1):36–44, 1999.

[21] O. Kennedy, C. Koch, and A. J. Demers. Dynamic Approaches to In-network Aggregation. In *Proc. of ICDE*, 2009.

[22] U. Lee and M. Gerla. A survey of urban vehicular sensing platforms. *Computer Networks*, 54(4):527–544, 2010.

[23] L. Liu, C. Pu, and W. Tang. Continual Queries for Internet Scale Event-Driven Information Delivery. *IEEE TKDE*, 11(4):610–628, 1999.

[24] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TinyDB: An Acquisitional Query Processing System for Sensor Networks. *ACM TODS*, 30(1):122–173, 2005.

[25] A. Melki, S. Hammadi, Y. Sallez, T. Berger, and C. Tahon. Advanced approach for the public transportation regulation system based on cybercars. *RAIRO-Oper. Res.*, 44(1):85–105, 2010.

[26] Meta System S.p.A, Via Majakovskij 10/b/c/d/e - 42124 Reggio Emilia - Italy. Web site: http://www.metasystem.it.

[27] S. E. Middleton, N. R. Shadbolt, and D. C. De Roure. Ontological User Profiling in Recommender Systems. *ACM TOIS*, 22(1):54–88, 2004.

[28] Octo Telematics S.p.A. Via Lamaro, 51 00173 Roma - Italy. Web site: http://www.octotelematics.com/.

[29] PEGASUS: ProgEtto per la Gestione della mobilità Attraverso Sistemi infotelematici per l'ambito Urbano, per la Sicurezza di passeggeri, veicoli e merci. Web site: http://pegasus.octotelematics.com.

[30] Pegasus partners.
Web site: http://pegasus.octotelematics.com/web/guest/partners.

[31] PTV AG, Stumpfstr. 1 - 76131 Karlsruhe, Germany
. Web site: http://www.english.ptv.de.

[32] C. Ré, J. Letchner, M. Balazinska, and D. Suciu. Event queries on correlated probabilistic streams. In *Proc. of SIGMOD*, pages 715–728, 2008.

[33] Regressione Lineare.
Web site: http://automatica.ing.unibs.it/mco/ms/regressione/paragrafi teoria/regressione lin.html.

[34] H. P. Ritzema. Frequency and regression analysis. *Drainage Principles and Applications (ILRI)*, 16(6):175–224, 1994.

[35] Simjava. Web site: http://www.dcs.ed.ac.uk/home/hase/simjava/.

[36] L. Speičvcys, C.S. Jensen, and A. Kligys. Computational data modeling for network-constrained moving objects. In *Proc. of GIS*, pages 118–125, 2003.

[37] SRS - Software Requirements Specifications. Web site: http://standards.ieee.org/reading/ieee/std_public/description/se/830-1998_desc.html.

[38] UML - Unified Modeling Language. Web site: http://www.uml.org/.

[39] ALMA MATER STUDIORUM - Università di Bologna , Centro Interdipartimentale L.Galvani. Web site: http://www.centrogalvani.unibo.it.

[40] VISSIM - Multi-Modal Traffic Flow Modeling.
Web sites:
http://www.english.ptv.de/software/transportation-planning-traffic-engineering/software-system-solutions/vissim/
http://www.tpsitalia.it/software/ingegneria_traffico/vissim.php.

[41] The World Wide Web Consortium (W3C). Web site: http://www.w3.org.

[42] F. Wang, S. Liu, and P. Liu. Complex RFID event processing. *The VLDB Jour.*, 18(4):913–931, 2009.

[43] IEEE 802.11TM WIRELESS LOCAL AREA NETWORKS. Web site: http://www.ieee802.org/11/.

[44] IEEE 802.11 from Wikipedia.
Web site: http://it.wikipedia.org/wiki/IEEE_802.11.

[45] The Xerces Java Parser. Web site: http://xerces.apache.org/xerces-j/.

[46] Y. Yao and J. Gehrke. Query Processing in Sensor Networks. In *Proc. of CIDR*, 2003.

[47] P. Zhuang, Q. Qi, and Y. Shang. Wireless sensor networks in intelligent transportation systems. *Wireless Communications and Mobile Computing*, 9(3):287–302, 2009.