POLITECNICO DI TORINO

Collegio di Ingegneria Informatica, del Cinema e Meccatronica
Corso di Laurea Magistrale Ingegneria Informatica

Tesi di Laurea Magistrale

Analisi, progettazione e sviluppo di applicazioni inerenti servizi di connettività, sia lato infrastruttura che veicolo

Relatore:
Prof. Gianpiero Cabodi

Supervisore aziendale Magneti Marelli:
Dott. Rosario Venturella

Candidato:

Francesco Triggiani

Dicembre 2018

Sommario

Questa tesi si situa nell'ambito dei Trasporti Cooperativi Intelligenti (C-ITS), sviluppati in vista di un progetto futuro che è quello della guida autonoma, più in particolare sulla tecnologia Vehicle-to-Everything (V2X) che permette ai veicoli opportunamente equipaggiati di comunicare tra loro, Vehicle-to-Vehicle (V2V), con l'infrastruttura, Vehicle-to-Infrastructure (V2I) e con gli utenti deboli della strada come pedoni e ciclisti, Vehicle-to- Pedestrian (V2P). Questa tecnologia ha lo scopo di ottimizzare i flussi di traffico e di migliorare la sicurezza stradale ed è basata sul DSRC, Dedicated Short-Range Communication e utilizza lo standard IEEE 802.11p [22] che è un'estensione dell'IEEE 802.11a. Il lavoro è stato svolto all'interno del gruppo di ricerca italiano Technology Innovation – Innovation Connectivity (TI-IC) in **Magneti Marelli**, presso la sede di Venaria Reale (TO), sotto la supervisione del Dott. Rosario Venturella.

Durante questo lavoro, è stata progettata e implementata un'architettura software, che prende il nome di **Test Site Visualizer** (TSV), che permette di ricevere i pacchetti che vengono scambiati tra i veicoli e di visualizzare in real-time su una mappa geografica la loro posizione corrente. La tesi si suddivide in due parti: la prima riguarda lo sviluppo dei tre componenti del TSV che sono il modulo software **Interpreter**, il **web server** ed il **website**. La seconda parte riguarda l'analisi delle prestazioni dell'intero framework Magneti Marelli. In particolare, viene posta l'attenzione sul tempo impiegato dai moduli software per effettuare la scrittura dei messaggi nei file di log. L'intera architettura è stata utilizzata dal gruppo di ricerca durante la dimostrazione della tecnologia V2X all'evento **Modena Automotive Smart Area** (MASA) tenutosi presso l'Autodromo di Modena, in Italia, nel Settembre 2018.

Ringraziamenti

Indice

Elenco delle figure

Elenco delle tabelle

Elenco dei simboli

1. ABS Antilock Braking System
2. BSM Basic Short Message
3. CAM Cooperative Awareness Message
4. CAN Controller Area Network
5. CCH Control Channel
6. DENM Decentralized Environmental Notification Message
7. DSRC Dedicated Short-Range Communications
8. EH Electronic Horizon
9. ESP Electronic Stability Control
10. GPS Global Positioning System
11. HMI Humane Machine Interface
12. HW Hardware
13. IVC Inter Vehicle Communication
14. LDM Local Dynamic Map
15. MAC Medium Access Control
16. OBU On Board Unit
17. POTI Positioning and Timing
18. RSU Road Side Unit
19. SCH Service Channel
20. SPAT Signal Phase And Timing
21. SW Software
22. TCP Transmission Control Protocol
23. TIM Traveler Information Message
24. TOPOM Topology Message
25. UDP User Datagram Protocol

26. V2I Vehicle to Infrastucture
27. V2V Vehicle to Vehicle
28. VDP Vehicle Data Provider

Parte I
Introduzione

Introduzione

Nello scenario di innovazione tecnologica moderno, le reti veicolari assumono un significato sempre più rilevante per quanto concerne lo sviluppo dei trasporti. Oggigiorno, infatti, la mobilità ha un ruolo centrale nel sistema sociale ed è in continua crescita su scala mondiale. Si conta che l'incidenza dei costi del trasporto sulle spese delle famiglie costituisce una voce considerevole, pari al 12% in Italia e sostanzialmente allineata alla media europea, pari al 13% [1]. Ciò porta al sorgere di tre grandi problemi che riguardano la sicurezza stradale, l'efficienza dei sistemi di trasporto e l'inquinamento.

Secondo la World Health Organization (WHO), l'organizzazione mondiale della sanità, gli incidenti stradali ogni anno causano in tutto il mondo circa 1,2 milioni di morti e circa 50 milioni di feriti. Se non verranno prese misure preventive la morte su strada è destinata a diventare la terza causa di morte nel 2020 dal nono posto nel 1990 [2]. Allo stesso tempo il settore è responsabile di circa il 33% dei consumi energetici finali, e rappresenta quindi un elemento sempre più centrale nelle politiche europee di contrasto ai cambiamenti climatici e alla riduzione dell'inquinamento nelle aree urbane. Le statistiche europee riferite ai 28 Paesi Membri evidenziano come ben il 30,4% dei gas serra e il 30,5% delle emissioni di anidride carbonica, nonché una parte considerevole dell'inquinamento atmosferico e acustico urbano, sono riconducibili ai trasporti. Questi valori per l'Italia salgono a circa il 34% [3].

Le reti veicolari, dunque, si pongono l'obiettivo di risolvere questi problemi dando vita a veicoli intelligenti che sono in grado di scambiare l'uno con l'altro molteplici informazioni che riguardano sia dettagli specifici come lo stato del veicolo, la posizione, la velocità che più generali come le condizioni del manto stradale e del traffico. Questi messaggi vengono scambiati non solo tra i veicoli ma anche tra il veicolo e l'infrastruttura, elaborati e tradotti in istruzioni per il veicolo il tutto ad una velocità dell'ordine dei millisecondi. Tale meccanismo rappresenta il concetto che sta alla base della realizzazione di un ambiente di guida sicuro e confortevole, permettendo a ciascun attore di conoscere in real-time lo scenario circostante.

Tramite lo sviluppo di questi sistemi si potranno prevenire gli incidenti dovuti alla distrazione o ad altre cause esterne, mantenendo il conducente costantemente aggiornato sullo stato del traffico, sulle condizioni del manto stradale e sulla presenza di eventuali ostacoli sulla carreggiata. Non solo, si potranno evi-

tare notevoli congestioni del traffico e ridurre i tempi di percorrenza dei tratti stradali, garantendo una maggiore efficienza nella gestione di tutto il sistema di trasporto. Quest'ultimo è un aspetto rilevante per quanto riguarda l'ambiente, dal momento che si potrà ridurre considerevolmente l'inquinamento e controllare le emissioni del veicolo, limitando così i danni causati dai gas nocivi.

Tutto ciò è reso possibile grazie al progresso tecnologico avvenuto nel campo delle telecomunicazioni e allo sviluppo in grande scala dei sistemi embedded, dispositivi specializzati a compiere determinate funzioni ed equipaggiati con un sistema operativo tipicamente della famiglia Linux e con hardware appositamente dedicato. Ciò permette di compilare il software per un determinato target hardware utilizzando una macchina, detta macchina build, differente dal target. La build ad esempio può essere un pc dotato di processore basato su architettura x86 e il target un microprocessore basato su architettura ARM.

Essi sono utilizzati all'interno di una gran varietà di prodotti come TV, tablet, ATM, elettrodomestici e hanno ottenuto un grosso impatto nel settore automotive. È stato possibile, infatti, creare sistemi che migliorano la stabilità e l'assetto del veicolo (ESP), sistemi di miglioramento della frenata (ABS), sensori di parcheggio, sistemi di navigazione assistita ed Engine Control Unit (ECU) che si occupano della gestione della formazione e combustione della miscela e del contenimento delle emissioni inquinanti di un motore a combustione interna.

Ed è proprio in questo scenario che emergono le reti veicolari che riconducibili ai sistemi ITS [17] (Intelligent Trasportation Systems) il cui obiettivo è quello di migliorare la sicurezza della guida, salvaguardare l'incolumità delle persone (safety) e la protezione del veicolo e delle merci (security) e l'efficienza del sistema di trasporto diminuendo l'impatto ambientale. Un esempio di rete veicolare è la Vehicular Ad-hoc Network (VANET), una rete wireless per la comunicazione fra i veicoli e fra l'infrastruttura e il veicolo introdotta per la prima volta nel 2001.

I sistemi ITS prevedono anche l'uso delle tecnologie ITC, Information and Communication Technologies, volte a creare una rete globale che comprende anche il sistema di trasporto ferroviario, aereo e marittimo. I sistemi di comunicazione interveicolare si suddividono in tre grandi categorie: Vehicle-to-Vehicle (V2V), Vehicle-to-Infrastructure (V2I), Vehicle-to-Pedestrian (V2P). Esse sono raggruppabili in una macro-categoria detta Vehicle-to-Everything (V2X).

Le tre grandi organizzazioni mondiali, negli ultimi dieci anni, hanno proceduto ad emanare alcuni standard per regolare i sistemi ITS. Essi sono:

- WAVE [18], Wireless Access in Vehicular Environment, emanato da Institute of Electrical and Electronics Engineers (IEEE [28]) in USA;
- ITS-G5 emanato da Europa l'European Telecommunications Standards Institute (ETSI) in Europa;

- STD-T109 [19] emanato da Association of Radio Industries and Businesses (ARIB) in Giappone.

Inoltre, c'è stata la necessità, a livello mondiale, di uniformare questi standard per normare i sistemi ITS e una delle compagnie che sta procedendo in tal senso è il gruppo italiano Magneti Marelli, il quale ha progettato un framework con l'obiettivo di unificare lo standard europeo ETSI con quello americano WAVE.

MAGNETI MARELLI

La fondazione dell'azienda risale al 1919, col nome di F.I.M.M. (Fabbrica Italiana Magneti Marelli), frutto di una joint-venture tra la FIAT e la Ercole Marelli. Il suo primo stabilimento era localizzato a Sesto San Giovanni, alle porte di Milano, ed iniziò come produttrice di magneti destinati sia all'aviazione, sia ai motori a scoppio automobilistici e motociclistici.

Nel corso della sua storia ha fabbricato alternatori, batterie per auto, bobine, centraline, navigatori, quadri di bordo, sistemi elettronici, sistemi di accensione, sistemi di scarico e sospensioni per auto e motoveicoli. Dopo varie riconversioni e trasformazioni è divenuta una multinazionale che opera in 20 paesi su 4 continenti con 85 unità produttive e 13 centri ricerca e sviluppo.

Magneti Marelli [4] opera a livello internazionale come fornitore di prodotti soluzioni e sistemi ad alta tecnologia per il mondo automotive. La sede centrale è in Italia, a Corbetta (Milano). Con 7,9 miliardi di Euro di fatturato nel 2016, circa 43.000 addetti, 85 unità produttive, 15 centri R&D, il gruppo è presente in modo capillare in 20 Paesi (Italia, Francia, Germania, Spagna, Regno Unito, Polonia, Repubblica Ceca, Romania, Russia, Serbia, Slovacchia, Turchia, USA, Messico, Brasile, Argentina, Cina, Giappone, India e Malesia) [24].

Magneti Marelli fornisce tutti i maggiori car makers in Europa, Nord e Sud America e Asia: è fornitore di componentistica e sistemi elettronici per i principali campionati motoristici internazionali, tra cui Formula 1, MotoGP e WRC. Nella Formula 1 è stata sponsor tecnico dei principali team, tra cui Scuderia Ferrari, Toyota F1 Team, Renault F1 e Red Bull Racing. Nella sua missione di fornire componentistica per il mondo automotive a livello globale, Magneti Marelli mira a coniugare qualità e offerta competitiva, tecnologia e flessibilità, con l'obiettivo di rendere disponibili prodotti d'eccellenza a costi competitivi.

Magneti Marelli punta a valorizzare, attraverso un processo di innovazione continua, il know-how e le competenze trasversali al fine di sviluppare sistemi e soluzioni che contribuiscano all'evoluzione della mobilità secondo criteri di sostenibilità ambientale, sicurezza e qualità della vita all'interno dei veicoli. Ma-

gneti Marelli fa parte di FCA.

Magneti Marelli opera a livello internazionale attraverso otto aree di business che sono: Electronic Systems, riguardante i quadri di bordo, infotainment & telematica, lighting & body electronics; Automotive Lighting, che si occupa dei sistemi di illuminazione; Powertrain, che si dedica ai sistemi di controllo del motore benzina, diesel e multifuel, e al cambio robotizzato AMT Freechoice; Suspension Systems, riguardante i sistemi sospensioni, ammortizzatori, dynamic system e i sistemi di controllo dinamico del veicolo; Exhaust systems, che si occupa dei sistemi di scarico, convertitori catalitici, silenziatori; Motorsport, riguardante i sistemi elettronici ed elettromeccanici specifici per le competizioni con leadership tecnologica in Formula1, MotoGP, SBK e WRC; Plastic Components and Modules, che si occupa dei componenti e dei moduli plastici per l'automotive e infine Aftermarket Parts and Services, che si occupa della distribuzione dei ricambi per l'Independent Aftermarket (IAM); e della Rete Assistenza e Officine Checkstar.

Obiettivo della tesi

La comunicazione veicolare prevede lo scambio di numerosi messaggi che contengono informazioni su un notevole numero di dettagli che riguardano lo scenario studiato. Uno fra questi contiene informazioni di base del veicolo e viene detto BSM, Basic Safety Message. Esso fornisce dettagli essenziali come la velocità del veicolo, la posizione in coordinate GPS, l'orientamento e molto altro.

Compito principale di questo lavoro di tesi è quello di progettare e implementare un modulo, detto INTERPRETER, che funga da sniffer di questi pacchetti. Esso si comporta come un filtro di messaggi BSM e quando questi vengono scambiati, il modulo salva dinamicamente i dati contenuti al loro interno in una struttura di appoggio per poi spedirli sul server e permettere di utilizzare tali informazioni per due scopi principali.

La prima interessante utilità dello sniffer riguarda, infatti, la capacità di filtrare ogni singolo pacchetto scambiato, di permettere l'identificazione del mittente e la rilevazione di un possibile errore avvenuto durante la comunicazione. È dunque possibile controllare il corretto funzionamento della comunicazione inter veicolare e rilevare possibili guasti o malfunzionamenti, indicando l'id univoco del veicolo dal quale non vengono ricevuti più messaggi e intervenire sul medesimo per risolvere l'imprevisto.

Il secondo fine dello sniffer riguarda la seconda parte della tesi e prevede la realizzazione di un tile server, ossia un server che sia in grado di fornire estratti di mappe geografiche, appositamente renderizzati durante una fase di prerendering, e di un sito web che accede a tali mappe e permette di visualizzare in real-time la posizione geografica corrente dei veicoli, rilevando se sono fermi o

in movimento.

È possibile così tenere traccia dei veicoli, seguirne il percorso e controllare le aree che stanno percorrendo. Il server riceve le informazioni dallo sniffer che invia un oggetto contenente le coordinate del veicolo, esse vengono estrapolate e raffigurate sulla mappa, nella sezione real time del sito web.

Terza ed ultima parte della tesi riguarda lo studio ed il test delle più veloci librerie di logging disponibili per C++, le cui caratteristiche soddisfano i requisiti di velocità richiesti dal framework. Utilizzare, infatti, librerie che riducano i tempi di log, ossia di scrittura su file dei risultati ottenuti durante l'esecuzione dei vari moduli, consente di ottimizzare e rendere più fluido il funzionamento di tutto il sistema, tenendo conto della velocità alla quale vengono scambiati i messaggi e delle caratteristiche hardware di cui si dispone.

In particolare, sono state aggiunte al progetto 6 librerie di log ed è stato creato un modulo-test che lanci un numero specifico di thread che loggano un numero specifico di volte richiamando la libreria da testare. Al termine di vari stress-test è stata trovata una libreria notevolmente più performante delle altre che è stata quindi inclusa nel framework.

Parte II
Background

Background

1 Inter-Vehicular Communication

La Inter-Vehicular Communication (IVC [5]) rappresenta una componente importare dell'architettura ITS (Intelligent Trasportation System) e una concreta applicazione delle Mobile Ad-hoc Networks (MANET [29]). La sua caratteristica principale è quella di garantire la comunicazione tra il guidatore, o il suo veicolo, con gli altri guidatori, o i loro veicoli, che sono fuori dal LOS (line of sight). Diretta conseguenza di questo sistema è un miglioramento della sicurezza stradale [25] e dell'efficienza nei trasporti. Inoltre, sfruttando le grandi potenzialità dell'auto in termini di spazio ed energia fornita, si ottiene un elevato range di comunicazione e si consente un utilizzo idealmente illimitato degli apparecchi coinvolti.

1.1 Caratteristiche e applicazioni

Le applicazioni principali della IVC possono essere raggruppate in tre grandi classi:

- Funzioni di informazione e allerta: diffusione delle informazioni stradali riguardanti incidenti, congestioni del traffico, condizioni del manto stradale a tutti i veicoli nella rete.
- Controllo communication-based: sfruttamento della capacità di look-through della IVC per evitare incidenti e congestioni.
- Sistemi di assistenza cooperativa: coordinamento tra i veicoli in punti critici del tratto stradale come zone con scarsa illuminazione o imbocchi autostradali.

L'architettura di una IVC può essere classificata in tre grandi tipologie. La prima è la rete cellulare o WLAN nella quale vengono utilizzati i gateway della rete mobile o gli access point della rete WLAN per connettersi a Internet. Questa tipologia non può però essere implementata in larga scala poiché i costi sono elevati e ci sono delle limitazioni geografiche.

La seconda è la rete ad-hoc [30] senza infrastruttura in cui la comunicazione V2V è realizzata usando la comunicazione DSRC [26], Dedicated Short-Range

Communication, e ciascun veicolo è equipaggiato con un dispositivo di rete wireless.

Infine, la terza è ibrida in quanto ha sia le caratteristiche di una rete cellulare/WLAN che quelle di una rete ad-hoc. I veicoli utilizzano la Infrastructure Unit per accedere alle informazioni dinamiche scambiate fuori dal proprio range e condividono le informazioni tramite una comunicazione V2V senza infrastruttura come mostrato in figura 1.

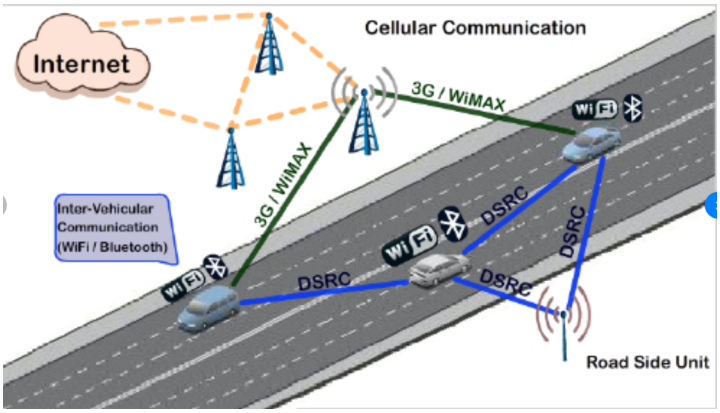

Figura 1: Wireless networking technology in V2V and V2I communication. DSRC: Dedicated Short Range Communication. WiFi: Wireless Fidelity.WiMAX: Worldwide Interoperability for Microwave Access

Vehicular Ad-hoc Network (VANET)

Utilizzate nella IVC sono le Vehicular Ad-hoc Network [16] (VANET), che rappresentano una specializzazione delle MANET, hanno alcune caratteristiche che le hanno rese competitive.

Innanzitutto, questa rete ha una topologia altamente dinamica e ciò è utile dal momento che io movimento altamente rapido dei veicoli rende difficile conservare la stessa topologia ed il numero di nodi tende ad aumentare rapidamente. La seconda caratteristica è che è una rete frequentemente disconnessa: dato il cambiamento di topologia, si può perdere la connessione in rete e i nodi possono uscire dall'area di copertura. La densità del traffico può variare notevolmente durante le diverse ore del giorno risultando spesso in una rete disconnessa.

Rilevante è modello di propagazione. Esso non rappresenta uno spazio completamente libero ma bisogna considerare alcuni ostacoli come palazzi, alberi e altri veicoli. Inoltre, c'è una notevole interferenza tra le trasmissioni wireless emesse dagli altri veicoli e dagli access point.

In questa rete, inoltre, si considera una durata idealmente illimitata della batteria e capacità di storage dei veicoli e sensori on-board. Molti veicoli, infatti, sono equipaggiati con sensori e unità GPS che servono a favorire la comunicazione e il processo di routing.

I sistemi IVC racchiudono due grandi tipologie di comunicazione: la Vehicle-to-Vehicle (V2V) e la Vehicle-to-Infrastructure (V2I). La prima riguarda la comunicazione tra i veicoli per poter usufruire di servizi come l'assistenza cooperative driver e ottenere informazioni sul traffico, la seconda riguarda la comunicazione tra il veicolo e l'infrastruttura che si trova sul bordo della strada e server per fornire al guidatore servizi come hot-spot per accesso a internet, avvisi, chat inter veicolare o anche giochi. Si possono raggruppare entrambe in comunicazione Vehicle-to-Everything (V2X).

I veicoli che percorrono una determinata strada, inviano, in modo del tutto automatico, ai propri "vicini" (ovvero a tutti i veicoli che si trovano all'interno del raggio di copertura dell'antenna trasmittente) informazioni riguardanti il proprio stato (es. velocità, posizione) ed anche informazioni provenienti dal mondo esterno come ad esempio le condizioni del manto stradale.

I veicoli che ricevono questi messaggi, acquisiscono automaticamente informazioni utili sullo stato del traffico e della viabilità. Alla ricezione di ogni messaggio, in seguito alla verifica e autenticità delle informazioni, vengono analizzate ed utilizzate come input di strategie decisionali (Es. incidente tra 100m rallenta drasticamente).

Un sistema di comunicazioni interveicolare richiede, su ogni veicolo, l'utilizzo di un dispositivo hardware dedicato, sia che si tratti di un mezzo pubblico o privato che di infrastrutture intelligenti presenti lungo le strade.

I dispositivi hardware presenti sui veicoli prendono il nome di On Board Unit (OBU), sono responsabili delle comunicazioni che avvengono tra due o più veicoli, mentre le infrastrutture a bordo strada prendono il nome di Road Side Unit (RSU). Le caratteristiche principali dei dispositivi OBU e RSU sono:

- Transceiver IEEE 802.11 necessario per comunicare tramite connessione wireless, sia per le reti infrastrutturali sia per quelle interveicolari.
- Controller Area Network (CAN): Caratteristica delle sole OBU, raccoglie i dati provenienti dai sensori del veicolo che comunicano con la centralina.
- Sistema GNSS
- Ethernet interface

- Costi contenuti: si vogliono mantenere bassi i costi di produzione in modo tale che il prezzo non scoraggi la diffusione del prodotto.

In particolare le Road Side Unit (RSU) possono essere dislocate in qualsiasi punto, tipicamente su supporti alti in modo da avere un raggio di trasmissione più ampio, ad esempio presso distributori di carburante, semafori o semplicemente a bordo strada. Alcune tra le tante applicazioni delle RSU riguardano la possibilità di propagare i messaggi provenienti dai veicoli, con i quali comunicano ad esempio la loro presenza e posizione. Un'altra interessante caratteristica riguarda la possibilità di cooperare con le altre stazioni RSU al fine di scambiare informazioni riguardanti messaggi di sicurezza, traffico, meteo ecc. Infine, è possibile trasmettere messaggi riguardanti la mappa locale o messaggi relativi ai servizi che vengono offerti in quell'area.

1.2 Sicurezza stradale

Grazie al sistema V2X, ogni veicolo è in grado di conoscere la posizione, la direzione, la velocità ed altri dati di base dei veicoli vicini. Pertanto, è possibile intuire manovre avventate, potenzialmente pericolose per la sicurezza stradale, in anticipo rispetto a quanto accade oggi senza V2X. Quando un veicolo rileva un pericolo imminente, il guidatore viene avvisato istantaneamente con un segnale acustico, visivo o anche aptico, come ad esempio quando il guidatore riceve un feedback tattile dallo sterzo vibrante.

Considerando il tempo medio di reazione dell'essere umano che è di circa 2 secondi, con un'incertezza diversa da persona a persona, con l'ausilio di questi segnali si ha un effettivo guadagno in termini di tempo di reazione, in quanto la segnalazione impiega pochi millisecondi per avvisare il guidatore che, in tal modo, avrà guadagnato secondi preziosi per poter reagire.

Grazie alla bassissima latenza delle comunicazioni wireless, è possibile comunicare in brevissimo tempo la presenza di un pedone e / o un ciclista che si trova in prossimità di un veicolo. In particolare grazie alla tecnologia Wi-Fi Direct, che permette a due dispositivi di comunicare direttamente tra di loro, consente al pedone di essere rintracciato facilmente o dalle infrastrutture a bordo strada (RSU) oppure dai veicoli stessi. Ciò ritorna molto utile sulle strade congestionate o in condizioni di scarsa visibilità.

Infine, il veicolo grazie ai suoi sensori, come il Elettronic Stability Program (ESP), riconosce ad esempio le condizioni del manto stradale precedentemente citate. Queste informazioni sono trasmesse ai veicoli vicini tramite l'utilizzo di appositi messaggi. Di conseguenza verrà generato un messaggio di allerta di tipo Decentralized Environmental Notification Message (DENM) relativo alla condizione del manto stradale, che verrà trasmesso in broadcast.

I veicoli che riceveranno questo messaggio verranno avvisati della presenza un tratto di strada pericoloso. I veicoli che li ricevono possono, sia avvisare il conducente, sia azionare automaticamente dei meccanismi per l'ottimizzazione dell'assetto del veicolo e inoltrare a loro volta le informazioni ai loro vicini.

Al fine di poter riconoscere questi messaggi provenienti dai sensori, è strettamente necessario che la OBU legga i dati dalla rete Controller Area Network (CAN) o direttamente dai sensori stessi (es. telecamera per rilevare ostacoli sulla carreggiata).

Un altro obiettivo delle interveicolari è quello di consentire una corretta gestione del traffico. Lo scambio ad alta frequenza tra veicoli ed infrastruttura, infatti, permette di garantire una viabilità intelligente, ottimizzando i costi relativi ai trasporti e diminuendo il tempo impiegato nel viaggio.

Un semaforo intelligente, ad esempio, comunica ai veicoli nelle vicinanze dell'incrocio le tempistiche semaforiche relative e le regola a sua volta in base alle comunicazioni ricevute dai veicoli in coda. L'obiettivo dell'infrastruttura è ottimizzare le code per smaltire il traffico nella maniera più efficiente.

I veicoli trasmettono informazioni di presenza e elaborano le tempistiche semaforiche suggerendo al guidatore la velocità ottimale da tenere per passare con il verde o fermarsi al rosso. Si ottiene così una guida più efficiente in termini di tempo e consumi, il che ha una diretta conseguenza positiva sull'impatto ambientale. Il sistema nel complesso risulta così essere più efficiente (tempo, meno ingorghi), più sicuro (conoscenza dello stato del semaforo) ed ecofriendly (meno consumi).

2 Framework Magneti Marelli

Il framework è stato progettato con l'obiettivo di unificare lo standard Europeo (ETSI) con quello Americano (WAVE [20]) per la comunicazione veicolare. Sono stati individuati tre gruppi software: quelli che si riferiscono solo allo standard Europeo, come ad esempio DENM e CAM, decodificati secondo lo standard EN 302 637-2 [31] e EN 302 637-3 [32], quelli che appartengono allo standard Americano (Es. BSM, TIM) e quelli che sono in comune ai due standard (Es. LDM, MAP, HMI, SPAT). Come mostrato in figura, è suddiviso in tre grandi layer: middleware, facilities e application.

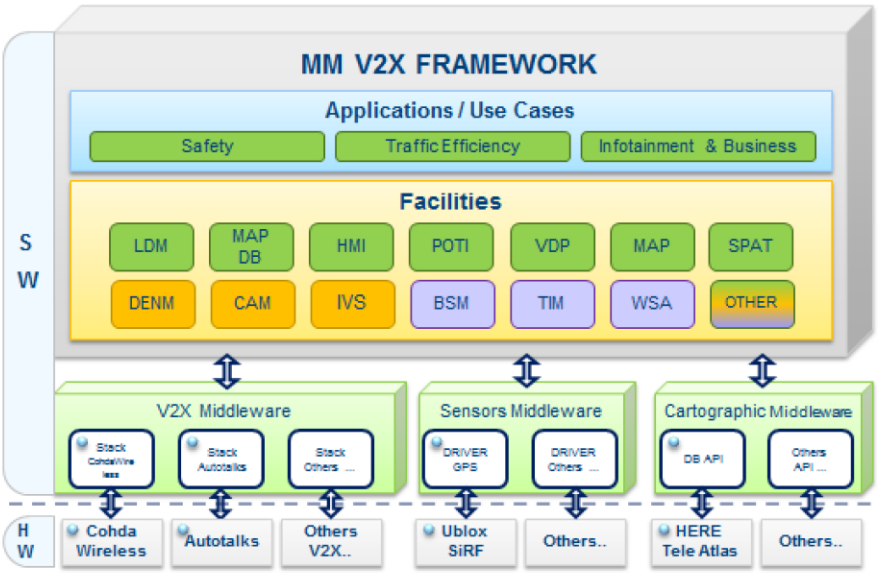

Figura 2: Framework V2X Magneti Marelli

2.1 Analisi dei layer

Middleware layer

Il middleware è un layer che crea un'interfaccia in modo da rendere i livelli superiori (facilites e application layer) indipendenti dal tipo di driver che si è scelto di utilizzare. Questo ci permette di rendere il software adattabile a qualsiasi stack V2X che si andrà a utilizzare (Cohda Wireless, Autotalks).

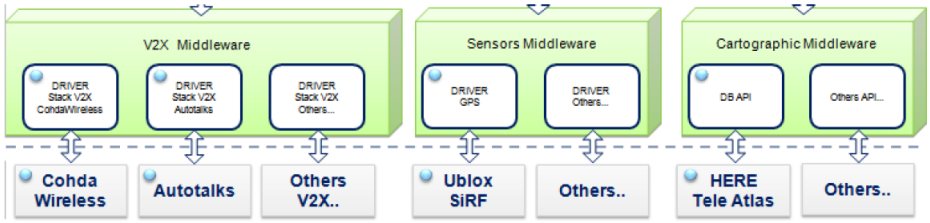

Figura 3: Middleware layer

Facilities layer

In questo livello sono definiti la maggior parte dei moduli software che compongono il framework.

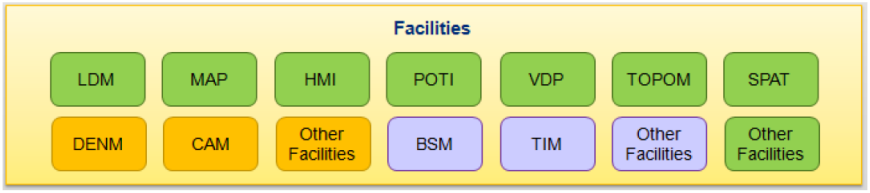

Figura 4: Facilities layer

Di seguito, vengono riportati i dettagli di ciascun modulo.

LDM

Local Dynamic Map può essere rappresentato come un modello composto da quattro strati. Partendo dallo strato più basso esso è costituito da dati statici come ad esempio le mappe. Il secondo strato è composto da dati relativamente statici, ovvero da segnali che non sono compresi sulla mappa.

Il terzo è costituito da dati relativamente dinamici come congestione del traffico o stati semaforici. In fine nel quarto strato vi sono i dati altamente dinamici come ad esempio le informazioni dei sensori automobilistici.

Local Dynamic Map è un modulo software che coopera con sistemi di trasporto intelligenti. La sua principale funzione è quella di gestire le informazioni che influenzano o sono influenzate dal traffico stradale, in particolare le fonti che generano dati possono essere diversi come veicoli, infrastrutture esterne, sensori di bordo, segnaletiche. Il modulo LDM è il fulcro del framework, dal quale

passano la stragrande maggioranza delle informazioni.

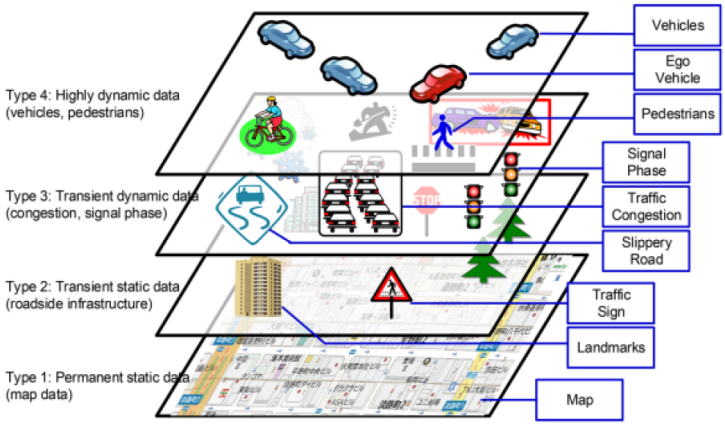

Figura 5: Modello LDM

BSM

Il Basic Safety Message è un modulo che appartiene allo standard americano, utilizzato in una varietà di applicazioni per scambiare messaggi relativi sia alla sicurezza che allo stato del veicolo. È un messaggio che viene inviato ad alta frequenza, fino a dieci volte al secondo. fornisce quasi le stesse funzionalità / informazioni dei due moduli CAM e DENM.

TIM

Il Traveler Information Message fornisce informazioni aggiuntive sullo stato del traffico o sulla segnaletica stradale. Il TIM non ha una durata infinita, esso ha un periodo di tempo limitato per cui resta valido ed inoltre può essere definita anche una area di validità fuori dalla quale esso perde di significato.

MAP

È utilizzato per trasmettere molti tipi di informazioni geografiche. Al momento attuale il suo uso primario è trasmettere la geometria di una o più intersezioni con un singolo messaggio, e quindi fornisce una mappa delle intersezioni.

Il contenuto messaggio MAP include campi come descrizioni delle intersezioni, dei tratti di strada o dei segmenti di carreggiata. Dato un singolo messaggio

di MAP esso può trasmettere le descrizioni di una o più aree geografiche o intersezioni. Il contenuto di questo messaggio spesso è utilizzato a sua volta da altri moduli per la generazione di nuovi messaggi, basti pensare al messaggio SPAT che ha bisogno di alcune delle informazioni del MAP per definire in quale corsia si trova un determinato semaforo.

SPAT

È utilizzato per trasmettere lo stato attuale di una o più intersezioni, se unito ad un messaggio di tipo MAP, il ricevitore potrà determinare lo stato del segnale e la relativa temporizzazione. Il messaggio SPAT invia lo stato corrente di ogni fase attiva del semaforo e quanto tempo manca alla prossima fase. Normalmente le informazioni sulle fasi non attive non sono trasmesse.

HMI

L' Human Machine Interface è un tramite tra il guidatore e tutti i possibili messaggi che il veicolo può comunicare a esso. In base alle esigenze applicative, l'HMI può scambiare informazioni sia in modo monodirezionale che bidirezionale. Può essere un gestore statico o dinamico delle informazioni in ordine di priorità, come il livello di emergenza delle informazioni ad esempio segnalazione di rischio di collisione, le informazioni sullo stato del traffico.

POTI

Il Positioning and Timing fornisce informazioni 3D sulla posizione (latitudine, longitudine, altitudine) e informazioni relative al tempo. Il POTI elabora i dati ricevute dai sensori, come ad esempio il GPS e altri dati provenienti dai diversi sensori dei veicoli e li fonde per ottenere informazioni di posizione sulla stazione mobile. Fornisce inoltre le informazioni sul tempo provenienti dai satelliti al fine di sincronizzare il timing del sistema.

Questo modulo necessita di una connessione permanente con un sistema GNSS (GPS, GALILEO, etc...). In caso di interruzione temporanea della connessione, alcuni meccanismi interni complementari devono essere utilizzati come backup, come ad esempio un giroscopio o un odometro, DEAD RECKONING.

VDP

Il Vehicle Data Provider è un modulo collegato con tutta la rete del veicolo e fornisce informazioni sullo stato dello stesso.

TOPOM

Il Topology Message fornisce delle informazioni dettagliate sulla tipologia della strada locale. Ad esempio, quando viene inviato un messaggio di tipo SPAT ad un veicolo, si può mandare anche un TOPOM così che il ricevente possa correlare i dati dei due messaggi e verificare la tipologia della strada.

DENM

Il Decentralized Environmental Notification Message appartiene solamente allo standard Europeo ed ha come obiettivo quello di avvertire gli utenti della strada, su un evento speciale, di pericolo, ad esempio ghiaccio su strada, la presenza di un incidente o di un'improvvisa caduta massi. Il messaggio di allerta verrà propagato in broadcast, a tutti i veicoli dell'area geografica di interesse, e verrà ripetuto per un periodo di tempo prestabilito.

Contrariamente a quanto avviene con un messaggio CAM, il DENM è relativo ad una specifica area, in cui l'evento ha preso luogo, piuttosto che a un singolo veicolo. Altra differenza con il CAM è che quest'ultimo non appena appena viene ricevuto esso non viene ritrasmesso, il DENM invece può percorrere distanze maggiori utilizzando l'inoltro a più hop tra i veicoli.

CAM

Il Cooperative Awareness Message appartiene solamente allo standard Europeo. Fornisce informazioni sulla presenza, posizione e stato del veicolo, questo messaggio viene inoltrato a tutti i veicoli presenti nel vicinato che distano soltanto un solo hop. Ogni veicolo o infrastruttura deve essere in grado di ricevere e interpretare i messaggi CAM. La generazione di quest'ultimi avviene ad altra frequenza e ad ogni istante di tempo, più precisamente da un intervallo minimo di ogni 100 ms ad un massimo di un secondo.

Un messaggio CAM viene generato se soddisfa almeno uno tra i seguenti vincoli: la direzione del veicolo cambia con almeno un angolo maggiore di quattro gradi, la distanza tra posizione del vecchio CAM e il nuovo deve essere maggiore di cinque metri oppure la velocità registrata tra il vecchio CAM e il nuovo deve essere diversa di almeno un metro al secondo.

Applications layer

A questo livello vengono definite le applicazioni che fungono da casi d'uso, i quali mostrano un esempio pratico di dove possono trovare impiego tutte le

funzionalità del framework. In particolare, ci sono tre tipi di casi d'uso. Il primo è il **Safety** e riguarda la sicurezza stradale. I veicoli possono ricevere dei messaggi di allarme da altri veicoli, anche in modalità automatica, quando sulla carreggiata è presente un ostacolo o quando si è verificato un incidente e la scarsa visibilità non consentirebbe una frenata tempestiva. In condizioni non di emergenza, tale allarme potrebbe comunque servire per scegliere percorsi alternativi che evitino lunghe code. Un segnale di cautela potrebbe essere inviato nell'abitacolo quando due veicoli si dirigono verso un incrocio cieco.

Il secondo è il **Traffic Efficiency** e riguarda la gestione del traffico. L'introduzione di veri e propri semafori intelligenti ridurrebbe le lunghe attese ad un incrocio. I semafori non dovrebbero far altro che raccogliere ed elaborare le indicazioni di distribuzione del traffico lungo le direttrici dell'incrocio e regolare di conseguenza i cicli rosso-verde. Inoltre, l'arrivo di veicoli di emergenza potrebbe essere rilevato in anticipo, forzando un segnale rosso su tutti i lati dell'incrocio ben prima che questo venga occupato da un'autoambulanza o da un veicolo dei pompieri.

Infine, il terzo è l'**Infotainment & Business** e riguarda le informazioni per i viaggiatori. Fornisce informazioni di diverso genere: turistiche come ristoranti, cinema, musei o di servizi, come ospedali, parcheggi nell'area attraversata dal veicolo.

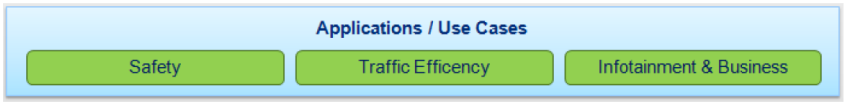

Figura 6: Applications layer

3 Linguaggi e Software utilizzati

3.1 Prima parte di tesi

3.1.1 Linguaggio C++

Il C++ è un linguaggio fatto per la programmazione orientata agli oggetti, che rende questo linguaggio una piattaforma ideale per realizzare progetti di grosse dimensioni favorendo l'astrazione dei problemi. Ciò ci consente di sviluppare software seguendo i più moderni pattern di progettazione: esistono numerosissime librerie già pronte e riutilizzabili, integrabili con i progetti.

I campi di applicazione sono i più svariati, dal gaming alle applicazioni realtime, dai componenti per sistemi operativi ai software di grafica e musica, da applicazioni per cellulari a sistemi per supercomputer. Esistono compilatori di C++ praticamente per tutte le piattaforme e sistemi operativi, con le dovute accortezze (spesso non banali) si può pensare di ricompilare un software e farlo girare in diversi contesti applicativi. Questo grazie al fatto che C++ è definito come standard [6].

3.1.2 MakeFile

Il make è l'utility, sviluppata sui sistemi operativi della famiglia UNIX, ma disponibile su un'ampia gamma di sistemi, che automatizza il processo di creazione di file che dipendono da altri file, risolvendo le dipendenze e invocando programmi esterni per il lavoro necessario. L'utility è usata soprattutto per la compilazione di codice sorgente in codice oggetto, unendo e poi linkando il codice oggetto in programmi eseguibili o in librerie. Esso usa file chiamati makefile per determinare il grafo delle dipendenze per un particolare output, e gli script necessari per la compilazione da passare alla shell.

3.1.3 Cross-Compiling

C ++ è un linguaggio portabile, quindi può essere compilato per diverse architetture. Un cross-compilatore è un compilatore capace di creare codice eseguibile per una piattaforma diversa da quella su cui il compilatore è in esecuzione. L'utilizzo fondamentale di un cross-compilatore è quello di separare l'ambiente di compilazione dall'ambiente di destinazione. GCC può essere impostato come cross-compilatore, esso supporta molte piattaforme e linguaggi di programmazione, in particolare G ++ è una delle versione di GCC per compilare C ++.
In questo lavoro di tesi è stato adoperato un sistema embedded con un processore ARM e un laptop con processore x86.

3.1.4 Apache Subversion (SVN)

Subversion (noto anche come svn, che è il nome del suo client a riga di comando) è un sistema di controllo versione per software. Gli sviluppatori di software utilizzano Subversion per mantenere versioni attuali e storici di file come codice sorgente, pagine web e documentazione. Le principali caratteristiche si SVN sono:

- Versione delle cartelle: viene tenuta traccia anche del contenuto di ogni singola directory e quindi lo spostamento di un file è a tutti gli effetti considerata una modifica, quindi rintracciabile e reversibile.
- Commits atomici: una serie di modifiche viene applicata solo in blocco, se anche solo una di esse crea errori, non viene applicata nessuna delle modifiche inviate. Questo evita la creazione di versioni incomplete che dovrebbero poi essere corrette a mano.
- Versione dei metadati: è possibile assegnare a file e directory delle proprietà personalizzate, dei veri e propri metadati e mantenere lo storico anche delle modifiche a queste proprietà.

Per convenzione, la cartella del ramo principale viene chiamata trunk e la cartella dedicata ai rami viene tipicamente chiamata branches o tags.

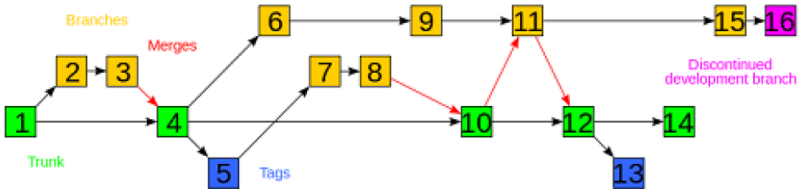

Figura 7: Schema di un flusso in Subversion

3.1.5 Enterprise Architect

Enterprise Architect è uno strumento di modellazione e progettazione basata su UML, Unified Modeling Language. La piattaforma supporta la progettazione, costruzione di sistemi software e la modellazione dei processi di business. Enterprise Architect copre diversi aspetti fondamentali del ciclo di vita delle applicazioni, dalla gestione dei requisiti fino alla progettazione, fasi di costruzione, collaudo e manutenzione, con supporto per la tracciabilità e gestione dei progetti.

Enterprise Architect è stato usato in questo lavoro di tesi per creare quattro

tipi di diagrammi. Il primo è il **Class Diagram** che è un tipo di diagramma che descrive la struttura di un sistema mostrando classi del sistema, attributi, operazioni o metodi, e le relazioni tra gli oggetti. Il secondo è il **Component Diagram** che mostra lo schema di come e cosa è composto un determinato applicativo software.

Il terzo è il **Sequence Diagram** che è un diagramma di interazione che mostra come interagiscono i processi e in quale ordine. Le interazioni sono disposte in sequenza temporale e inoltre viene rappresentata la sequenza di messaggi scambiati. Infine, l'ultimo è il **Requirements Diagram**, un diagramma personalizzato utilizzato per descrivere i requisiti e le caratteristiche del sistema. Utilizzato per illustrare i componenti dell'applicativo e quali sono i requisiti di sistema che devono essere soddisfatti

3.2 Seconda parte di tesi

3.2.1 Linguaggio Javascript

JavaScript è un linguaggio di scripting orientato agli oggetti e agli eventi, comunemente utilizzato nella programmazione Web lato client per la creazione, in siti web e applicazioni web, di effetti dinamici interattivi tramite funzioni di script invocate da eventi innescati a loro volta in vari modi dall'utente sulla pagina web in uso, come mouse, tastiera e caricamento della pagina.

Tali funzioni di script, utilizzati dunque nella logica di presentazione, possono essere opportunamente inserite in file HTML, in pagine JSP o in appositi file separati con estensione .js poi richiamati nella logica di business. Ultimamente il suo campo di utilizzo è stato esteso alle cosiddette Hybrid App (app ibride), con le quali è possibile creare app per più sistemi operativi utilizzando un unico codice sorgente basato appunto su JavaScript, HTML e CSS.

Fu originariamente sviluppato da Brendan Eich della Netscape Communications con il nome di Mochan e successivamente di LiveScript, ma in seguito è stato rinominato "JavaScript" ed è stato formalizzato con una sintassi più vicina a quella del linguaggio Java di Sun Microsystems (che nel 2010 è stata acquistata da Oracle). JavaScript è stato standardizzato per la prima volta il 1997 dalla ECMA con il nome ECMAScript. L'ultimo standard, di giugno 2017, è ECMA-262 Edition 8. È anche uno standard ISO.

Aspetti strutturali

La caratteristica principale di JavaScript riguarda l'essere un linguaggio interpretato: il codice non viene compilato, ma interpretato (in JavaScript lato client, l'interprete è incluso nel browser che si sta utilizzando). Inoltre, la sintassi è

relativamente simile a quella del C, del C++ e di Java. Javascript definisce le funzionalità tipiche dei linguaggi di programmazione ad alto livello (strutture di controllo, cicli, ecc.) e consente l'utilizzo del paradigma object oriented.

Questo è un linguaggio debolmente tipizzato e debolmente orientato agli oggetti. Ad esempio, il meccanismo dell'ereditarietà è più simile a quello del Self e del NewtonScript che a quello del linguaggio Java (che è un linguaggio fortemente orientato agli oggetti). Gli oggetti stessi ricordano più gli array associativi del Perl che gli oggetti di Java o del C++.

Altri aspetti di interesse riguardano il fatto che in JavaScript, lato client, il codice viene eseguito direttamente sul client e non sul server. Il vantaggio di questo approccio è che, anche con la presenza di script particolarmente complessi, il web server non viene sovraccaricato a causa delle richieste dei client.

Di contro, nel caso di script che presentino un codice sorgente particolarmente grande, il tempo per lo scaricamento può diventare abbastanza lungo. Un altro svantaggio è il seguente: ogni informazione che presuppone un accesso a dati memorizzati in un database remoto deve essere rimandata ad un linguaggio che effettui esplicitamente la transazione, per poi restituire i risultati ad una o più variabili JavaScript; operazioni del genere richiedono il caricamento della pagina stessa. Con l'uso di AJAX tutti questi limiti sono superati.

3.2.2 AJAX

Ajax è l'acronimo di Asynchronous JavaScript and XML, ed è una tecnica di sviluppo software per la realizzazione di applicazioni web interattive (Rich Internet Application). Lo sviluppo di applicazioni HTML con AJAX si basa su uno scambio di dati in background fra web browser e server, che consente l'aggiornamento dinamico di una pagina web senza esplicito ricaricamento da parte dell'utente.

AJAX è asincrono nel senso che i dati extra sono richiesti al server e caricati in background senza interferire con il comportamento della pagina esistente. Normalmente le funzioni richiamate sono scritte con il linguaggio JavaScript. Tuttavia, e a dispetto del nome, l'uso di JavaScript e di XML non è obbligatorio, come non è detto che le richieste di caricamento debbano essere necessariamente asincrone.

3.2.3 Node.js

Node.js è una piattaforma event-driven per il motore JavaScript V8 di Chrome UNIX like. Molti dei suoi moduli base sono scritti in Javascript, e gli sviluppa-

tori possono scrivere nuovi moduli in Javascript.

l modello di networking su cui si basa Node.js non è quello dei processi concorrenti, ma I/O event-driven: ciò vuol dire che Node richiede al sistema operativo di ricevere notifiche al verificarsi di determinati eventi, e rimane quindi in sleep fino alla notifica stessa: solo in tale momento torna attivo per eseguire le istruzioni previste nella funzione di callback, così chiamata perché da eseguire una volta ricevuta la notifica che il risultato dell'elaborazione del sistema operativo è disponibile.

Tale modello di networking, implementato anche nella libreria Event machine per Ruby e nel framework Twisted per Python, è ritenuto più efficiente nelle situazioni critiche in cui si verifica un elevato traffico di rete.

3.2.4 Websocket

Il protocollo WebSocket [8] crea una comunicazione bidirezionale in tempo reale tra client e server e risulta particolarmente adatto in questo lavoro di tesi in quanto si vuole realizzare, nella seconda parte, un'applicazione web real-time. Questi tipi di applicazioni possono sfruttare connessioni a bassa latenza e always-on, per la trasmissione rapida di informazioni in un modo precedentemente solo emulato da metodi come il polling Ajax e il long-polling, detto Comet.

I moderni sistemi di notifica si basano, infatti, su un abuso del protocollo HTTP aprendo i frame per effettuare il polling di nuovi dati dal server o utilizzando Flash per il livello di rete o per costruire tutta l'applicazione. Anche se evoluti, questi metodi hanno aspetti negativi come inefficienze o complessità. Websocket supera questi limiti, creando un protocollo che è un'estensione di quello TCP: la stessa connessione TCP che sottende l'HTTP viene usata per collegamenti full-duplex, persistenti ed efficienti, direttamente tra browser) e server web.

La specifica WebSocket è stabile e supportata da browser moderni come Chrome, Firefox e Internet Explorer 10. La connessione continua TCP consente di realizzare applicazioni molto reattive e connesse in modo molto più efficiente, sia per l'utilizzo delle risorse di client e server sia in fase di sviluppo, utilizzando un flusso "nativo" invece di un sistema di interrogazione a polling. Tecnicamente parlando, un WebSocket è una connessione TCP persistente, bidirezionale full-duplex, garantita da un sistema di handshaking client-key ed un modello di sicurezza origin-based. Il sistema, inoltre, maschera le trasmissioni dati per evitare lo sniffing di pacchetti di testo in chiaro.

La caratteristica bidirezionale indica che il client è connesso al server, e il server è connesso al client. Ciascuno può ricevere eventi come collegato e scollegato e possono inviare dati all'altro; quella full-duplex indica che il server e il client possono inviare dati nello stesso momento senza collisioni. Websocket possiede

entrambe le specifiche. Effettua anche il client-key handshake: il client invia al server una chiave segreta di 16 byte con codifica base64.

Il server poi aggiunge una stringa e rimanda il risultato, elaborato con SHA1, al client. In questo modo, il client può essere certo che il server cui aveva inviato la sua chiave è lo stesso che apre la connessione. Per quanto riguarda la sicurezza origin-based, l'origine della richiesta WebSocket viene verificata dal server per determinare che provenga da un dominio autorizzato. Il server può rifiutare le connessioni socket da domini non attendibili.

Le API offerte da WebSocket in JavaScript definiscono un oggetto che contiene: informazioni sullo stato della connessione, che può essere "connecting", "open", "closing" e "closed"; metodi per interagire con la connessione WebSocket, ossia chiudere una connessione e inviare dati; eventi che vengono sollevati all'occorrenza di un evento WebSocket, quando un socket viene aperto, chiuso o riceve in risposta un messaggio di errore.

WebSockets, essendo una connessione persistente, potrebbe richiedere molte più risorse rispetto ad un web server standard. L'impatto sul bilanciamento del carico e sul firewall può essere mitigato, la specifica WebSocket permette di trasferire le connessioni: un client può connettersi a un LoadBalancer che passa la connessione ad un application server per gestire l'elaborazione dei data frame.

3.2.5 Leaflet

Leaflet è una libreria JavaScript per sviluppare mappe geografiche interattive (WebGIS). Sviluppato dal 2010, supporta la maggior parte dei browser e degli standard HTML5 e CSS3. Leaflet permette di mostrare punti di interesse, linee o aree, o strutture dati come file GeoJSON, o livelli interattivi, su una mappa a tasselli.

Utilizzo

Un utilizzo tipico di Leaflet è l'inserimento di una mappa all'interno di un elemento HTML. Punti di interesse (markers) e livelli (layers) possono essere aggiunti successivamente.

Un'esempio d'uso della libreria Leaflet può essere visualizzato in Figura 8, in cui è evidenziato come viene impostato il layer principale che conterrà la mappa OSM, detto Tile Layer, indicando le coordinate in corrispondenza delle quali centrare la mappa e l'indirizzo del server da cui estrapolare i frammenti di essa, chiamati "tile".

```
<!DOCTYPE html>
<html>
<head>
<title>OpenStreetMap with Leaflet</title>
<link rel="stylesheet" href="http://cdn.leafletjs.com/leaflet-0.6.4/leaflet.css" type="text/css">
<script src="http://cdn.leafletjs.com/leaflet-0.6.4/leaflet.js"></script>
 <style>
        html,
        body,
        #map {
                height: 100%;
                margin: 0;
                padding: 0;
        }
 </style>
</head>
<body>
<div id="map" class="map"></div>
 <script>
        // Creazione della mappa
        var map = L.map('map').setView([45, 10], 3);

        // Settaggio layer OSM
        L.tileLayer('http://IP_SERVER/osm_tiles/{z}/{x}/{y}.png').addTo(map);
 </script>
</body>
</html>
```

Figura 8: Esempio d'uso della libreria Leaflet

Le API della libreria sono registrate nella variabile L.

Leaflet supporta i livelli Web Map Service (WMS), i livelli GeoJSON, i livelli vettoriali e i livelli a tasselli. Le funzionalità possono essere estese attraverso plugin.

Elementi

Alcuni elementi che definiscono una mappa Leaflet:

- Tipi **raster**, come Tile Layer e Image Overlay;
- Tipi vettoriali, detti **Vector**, come path, polygon e tipi specifici come Circle;
- Tipi di **raggruppamento** come Layer Group, FeatureGroup e GeoJSON;

- **Controlli** come Zoom o Layers.

Ci sono ulteriori classi per la proiezione, trasformazione e interazione col DOM.

Fornisce, inoltre, il supporto ai formati GIS Leaflet supporta alcuni formati GIS standard, estendibili attraverso plugin.

Supporti

La libreria Leaflet offre un supporto per GeoJSON completo, attraverso la funzione L.geoJson. Il plugin Leaflet-Omnivore consente di supportare KML, CSV, WKT, TopoJSON e GPX. Un altro supporto offerto è per WMS, gestito principalmente attraverso il tipo TileLayer. WFS e GML, invece, non sono supportati ma per il primo esistono plugin.

I browser supportati da Leaflet 0.7 sono Chromium, Google Chrome, Mozilla Firefox, Safari 5+ e Opera 12+ e 7-11

3.2.6 JOSM

Java OpenStreetMap Editor (JOSM) è un software libero ed è uno strumento che serve a editare geodati in formato OpenStreetMap. È stato sviluppato in Java, originariamente da Immanuel Scholz e poi viene al giorno d'oggi mantenuto da Dirk Stöcker. Ha molte funzionalità avanzate ma ha anche un'interfaccia utente abbastanza complessa rispetto a quella esistente di default online nota come editor ID.

Alcune caratteristiche interessanti di JOSM riguardano la possibilità di importare file GPX, fornendo possibilità di effettuare il tracciamento GPS e di lavorare con immagini aeree, compresi i protocolli WMS, TMS e WMTS. Offre, inoltre, un supporto per proiezioni cartografiche multiple e consente la modifica dei layer e delle relazioni.

Propone strumenti per effettuare data validation e data filtering. Offre, inoltre, la possibilità di lavorare offline e stili per effettuare presetting e rendering. JOSM, infine, offre più di 200 keyboard shortcuts per assolvere tante funzioni.

Alcune funzionalità aggiuntive offerte dal tool riguardano la possibilità di disegnare palazzi, aggiungere link per Wikipedia e visualizzare in grafica 3D sono disponibili grazie ai plugin. Per questi ultimi ci sono più di 100 repository.

4 Librerie studiate

Tutte le librerie implementate e testate, sono librerie di logging altamente veloci, appositamente create per moduli programmati in C++.

4.1 Boost

Le Librerie C++ Boost sono una collezione di librerie open source che estendono le funzionalità del C++. Molte di esse sono licenziate sotto la Boost Software License in modo da poter essere utilizzate sia in progetti open source che closed source. Alcuni dei fondatori di Boost fanno parte del comitato standard C++ (ISO/IEC 14882) e diverse librerie Boost sono state accettate per l'incorporazione sia in C++ Technical Report 1, sia in C++11.

Le librerie Boost sono pensate per essere ampiamente utili e utilizzabili in un ampio spettro di applicazioni. La licenza Boost incoraggia l'uso commerciale e non commerciale. Per assicurare efficienza e flessibilità, Boost fa un estensivo utilizzo della programmazione basata su template, e quindi sulla programmazione generica e metaprogrammazione.

Vengono fornite, inoltre, implementazioni di riferimento in modo che le librerie Boost siano adatte per l'eventuale standardizzazione. Le librerie Ten Boost sono incluse nel Report tecnico della Biblioteca del C ++ Standards Committee (TR1) e nel nuovo C ++ 11 Standard. C ++ 11 include anche molte altre librerie Boost oltre a quelle di TR1. Altre librerie Boost sono proposte per la standardizzazione in C ++ 17.

La libreria boost è largamente usata perché:

- È open-source.
- Offre un gran numero di funzionalità che STL non possiede.
- È un complemento ad STL più che un sostituto.

Molti sviluppatori Boost fanno parte del comitato C++. Infatti, molte sezioni della libreria Boost sono pensate per essere incluse nella prossima libreria standard C++. È documentata molto bene. Le sue licenze permettono di includerla sia in progetti open-source che closed-source projects. Le sue funzionalità sono spesso indipendenti l'una con l'altra per cui si possono linkare solo le sezioni interessante.

4.2 Spdlog

La libreria Spdlog è una libreria molto veloce, contiene solo gli header ed è fatta per il logging in C++. È compatibile per Linux, FreeBSD, OpenBSD, Solaris, AIX Windows (msvc 2013+, cygwin) macOS (clang 3.5+) e Android.

Caratteristiche

È davvero molto veloce. Presenta solo gli header, bisogna includerli nel progetto e la libreria è pronta per l'uso. Supporta ricche funzioni per formattare, utilizzando un'eccellente libreria fmt. È utilizzabile sia in Synchronous mode che in Asynchronous mode, quest'ultimo è molto veloce e presenta logger sia single che multi thread.

Permette di utilizzare diversi tipi di file di log. Permette di creare un file di log di tipo "basic", "rotating", "daily", "console" e "syslog". Offre, inoltre, il windows debugger, utilizzando "OutputDebugString(...)", facilmente estendibile con tipi di log customizzati; il severity based filtering grazie a cui la soglia del livello può essere modificata sia a runtime che in compile-time ed il binary data logging.

Permette il flush periodico; il log con multi-sink, ciascuno con un formato differente ed un livello di log specifico; il synchronous logging; l'asynchronous logging con multi-sink. Infine, consente all'utente di definire dei tipi e prevede un gestore degli errori.

4.3 Easylogging++

È una libreria di logging molto potente per C++. È estremamente potente, estendibile, abbastanza leggera, performante e offre il thread-safe. Permette la scrittura dei file di log in un formato personalizzato. Offre anche la possibilità di loggare classi, librerie di terze parti, STL e contenuti di terze parti.

Easylogging++ è estremamente configurabile in base all'utilizzo dell'utente e ai requisiti che lui chiede. Permette di scrivere dei sink personalizzati (richiamando funzioni come la LogDispatchCallback). Questa libreria è usata da centinaia di progetti open-source su github e altri sistemi di management anch'essi open-source.

Caratteristiche

È utile perché è utilizzabile su un progetto scritto in C++ di dimensioni sia piccole che grandi. È basata solo sugli header e richiede di effettuare un link ai file sorgente. Originariamente era header-only ma poi fu cambiata per risolvere un bug. Si può ancora utilizzare quel tipo nella versione v9.89.

Questa libreria è stata progetta per garantire:

- Portabilità
- Usabilità
- Funzionalità
- Prestazioni
- Facilità d'installazione

È configurabile a runtime, ampliabile per soddisfare i requisiti richiesti e offrendo performance elevate. Alcune librerie single-header ricorrono spesso all'utilizzo di librerie esterne come le Boost o Qt per supportare alcune funzionalità richieste dagli utenti come il multithreading. Questa libreria, invece, ha tutto i moduli implementati all'interno e non ricorre all'uso di librerie esterne e non si corre il rischio di dover dipendere da esse.

Caratteristiche principali

Easylogging++ presenta molte caratteristiche interessanti che sono spesso richieste dagli sviluppatori. Esse riguardano principalmente il fatto che siano altamente configurabili, permettano di essere estese e sono estremamente veloci. Garantiscono, inoltre, il "thread e type-safe" e sono cross-platform. Offre dei pattern per i log personalizzati e un log condizionale e occasionale.

Altre caratteristiche interessanti riguardano il fatto che permette di effettuare il tracking delle performance, di effettuare un log consistente, di gestire i crash e offre un aiuto nel controllo delle macros. Permette il logging secondo lo Standard Template Library (STL), e il syslog. Infine, offre la possibilità di linkare librerie di terze parti come Qt, Boost e WxWidgets ed è estendibile.

4.4 Nanolog

Nanolog è una libreria per il logging a latenza estremamente bassa per il C++. È davvero molto veloce, utilizza solo gli header standard e può lavorare bene con tutti i compilatori conformi al C++11. Supporta le tipiche caratteristiche di una libreria di log come molteplici livelli di log, log file di tipo rotazionale e la scrittura asincrona del file di log.

Caratteristiche

Non effettua la copia di stringhe di literals. Conversione di interi e double in ASCII. Non alloca memoria heap per le righe di log che non superano i 256 bytes.
Include header leggeri e minimali. Migliora di molto il tempo di compilazione del progetto.

Nanolog supporta due modalità di log:

- **Guaranteed logging**: i messaggi di log non vengono mai cancellati in caso di logging-rate elevato.
- **Non Guaranteed logging**: utilizza un buffer ad anello per mantenere le righe di log. In caso di logging-rate molto elevato quando il buffer è pieno (ad esempio il thread consumatore non riesce a eseguire il pop dell'oggetto con quella velocità), la riga precedente di log viene cancellata. Ciò non blocca il thread produttore anche se i buffer è pieno.

4.5 Mini-async Log

È una libreria asincrona molto performante con delle caratteristiche interessanti.

Ha una struttura minimalista e razionale ed è pensata per essere semplice. Non contiene funzionalità astratte e inutili che appesantiscono la libreria. Lascia capire cosa il codice sta elaborando e permette la modifica. Garantisce una latenza davvero bassa ed è veloce per il "consumer".

È asincrona in quanto le chiamate sincrone possono essere effettuate per messaggi speciali ma il logging diventa molto lento.

Offre formattazione minima della stringa sul thread chiamante per i casi d'uso più comuni. Non usa memoria locale per i thread, i thread dell'utente sono assunti come esterni. Presenta una funzionalità che permette di bloccare l'esecuzione finché tutti i log non sono stati scritti sul disco in caso di chiusura del programma o di malfunzionamenti.

Caratteristiche

Funziona con g++4.7 e VS 2010, garantisce performance nell'ordine dei nanosecondi ed è utilizzabile dalle librerie. L'utente può accedere all'istanza del log esplicitamente o da una funzione globale. È adatta per progetti soft-realtime e supporta il file rotating. Effettua una call per i livelli inattivi di log. Offre la possibilità di cambiare livelli di log a runtime. Non prevede ostream, effettua una formattazione delle stringhe intercettate a compile-time con un formato safe.

La soglia della severità del log può essere cambiata fuori dal processo e il meccanismo IPC è il più semplice in quanto i log interroga periodicamente alcuni file quando è in idle. Inoltre, consente l'interruzione, configurabile in base alla severità del log, in caso di coda piena.

Il logger inizia a comportarsi come un logger sincrono ma se questo comportamento non è desiderato, viene restituito un messaggio di errore ed è leggera e

si può compilare come parte del progetto.

Funzionamento

Offrendo un log asincrono, l'obiettivo principale è quello di essere veloce e avere la minore latenza possibile per il thread chiamante.
Quando l'utente sta per scrivere il messaggio di log, il thread producer serializza il dato in un chunk di memoria che poi il consumer potrà decodificare, formattare e scrivere. Sono comunque operazioni che non richiedono eccessivo tempo sul consumer.

Il formato di stringa richiesto deve essere un "literal", costante a compile-time, così nel caso in cui avvenga una chiamata di questo tipo:

```
log_error (‘‘File:a.cpp line:8 This string displays the
 next number {}’’, int32_val);
```

ciò che viene richiesto è solo un puntatore al formato stringa e alla copia del valore intero. Il lavoro del chiamante è solo quello di serializzare alcuni byte in un chunk di memoria e inserirli in una coda.

La coda è un mix delle due code famose 'lockfree' di Dmitry Vyukov. La coda è l'incontro di una coda preallocata basata su array di elementi a dimensione fissa e una coda allocata dinamicamente. La coda risultante è ancora linearizzabile.

Il formato di stringa è type-safe e validato a compile-time per i compilatori che supportano "constexpr" e "parametri di template variabili". Altrimenti vengono sollevati errori a run-time in output, soprattutto su Visual Studio 2010.

Ultima importante caratteristica è quella di permettere l'interruzione del thread chiamante finché alcuni messaggi non sono stati smaltiti dalla thread del logger.

4.6 Loguru

Logruru è una libreria di logging leggera e flessibile, scritta per C++.

Caratteristiche

I log prodotti sono human-readable e si prestano facilmente all'uso della grep.

Le sue caratteristiche principali riguardano il fatto che consente un'integrazione semplice nel codice del progetto, solo due file che sono loguru.hpp e loguru.cpp. Si può buildare e linkare loguru.cpp oppure solo effettuare l'include con #include <loguru.cpp> in uno dei file .cpp del progetto.

È leggera e semplice. Gli header sono di piccole dimensioni e non hanno gli #include per permettere una compilazione molto veloce. Non ha dipendenze ed è cross-platform.

È flessibile in quanto l'utente può utilizzare callback per il logging (per esempio per visualizzare messaggi di log sullo schermo durante l'esecuzione di un gioco). Possono essere, inoltre, utilizzate per i fatal error.

Supporta più file di output, sia in modalità truncate che append: per esempio un file di log contenente solo l'ultimo run con 'low verbosity' oppure un file di log pieno con 'highe verbosity' che viene scritto in append ad ogni run. Supporta livelli di verbosity.

Supporta le asserzioni del tipo:

```
CHECK_F(fp != nullptr, "Failed to open '%s'", filename);
```

Supporta l'abort del tipo:

```
ABORT_F("Something went wrong, debug value is %d", value;
```

Lo stack trace viene scritto in output dopo un abort e lo stack trace viene pulito. Prima di essere pulito viene eseguito:

```
some_function_name(std::__1::vector<std::__1::basic_string
<char, std::__1::char_traits<char>, std::__1::allocator<char>
>, std::__1::allocator<std::__1::basic_string<char,
std::__1::char_traits<char>, std::__1::allocator<char> > > >
const&);
```

Dopo essere pulito viene eseguito:

```
After cleanup: some_function_name
(std::vector<std::string> const&);
```

Gli stack trace vengono scritti in ordine cronologico.

Questa libreria è molto veloce. Infatti, quando è configurata in modo non bufferizzato, settando "loguru::g_flush_interval_ms = 0", impiega 6-8 us per loggare su stderr e su file, rMBP + SSD + Clang. È circa il 25%-75% più veloce della libreria GLOG. Con il settaggio "loguru::g_flush_interval_ms =100 ms" impiega 3-5 us per loggare su stderr e su file, rMBP + SSD + Clang.

Offre un completo rimpiazzo della GLOG; la possibilità di scelta tra la formattazione di tipo printf-style o std::cout-style e il controllo a tempo di compilazione

della formattazione print e sui compilatori supportati.

Permette il supporto per la formattazione di tipo fmtlib, aggiungengo la #define LOGURU_USE_FMTLIB 1 e prima di includere loguru.hpp, bisogna configurare la include directory di fmtlib per building così come per il linking oppure usare la definizione per il preprocessore FMT_HEADER_ONLY.

Errori sulle asserzioni sono marcati con "noreturn" per garantire ottimizzazione e tutti i log sono scritti su stderr e anche a colori se il terminale lo supporta. Inoltre, è thread-safe e può essere configurata in base alle necessità dell'utente, settando il numero di microsecondi dopo il quale effettuare il flush impostando il valore "loguru::g_flush_interval_ms".

Inoltre, in un thread in background viene effettuato il flush dell'output su ogni chiamata in modo da non perdere i dati in caso di crash e aggiunge preliminarmente ad ogni riga di log la data e l'ora con una precisione fino ai millisecondi, il tempo in cui è in esecuzione l'applicazione, il nome del thread utente o l'id, il file ed infine la riga di log e l'indentazione.

È in grado di controllare il valore delle variabili locali e tenerle in un catch. Esse vengono scritte in output in caso di crash. I log favoriscono l'esecuzione della grep in quanto ciascuna riga contiene tutte le informazioni di cui si necessita, come ad esempio la data, e si può facilmente applicare un filtro sui livelli di log che sono consistenti.

Parte III
Analisi e implementazione

Analisi e implementazione

5 Analisi dei prerequisiti

Il sistema completo del Test Site Visualizer (TSV) si suddivide in due grandi parti:

- Lato backend, costituita dall'Interpreter
- Lato frontend, costituita dal web server e dal web site

Entrambe le parti comunicano utilizzando la tecnologia UDP. Compito di questa prima fase di tesi è quello di andare a verificare i requisiti hardware e software richiesti per lo sviluppo del sistema TSV.

5.1 Backend TSV

Parte fondamentale per il funzionamento del Test Site Visualizer è l'Interpreter, che costituisce il backend del TSV. Esso è un modulo hardware che è stato progettato, implementato ed integrato all'interno del Framework Magneti Marelli, nella sezione WAVE [21].

Il compito principale dell'Interpreter è quello di intercettare i pacchetti che vengono scambiati tra i vari moduli, in particolare esso deve essere in grado di sniffare i messaggi BSM. Questo tipo di messaggio è molto importante poiché contiene informazioni di base del veicolo e sono quelle che servono per il funzionamento del Visualizer.

5.1.1 Architettura BS-BS

I messaggi BSM sono controllati dal modulo BS-BS, Basic Safety - Basic Service, il quale li processa all'interno del Framework V2X MM. Il BS-BS acquisisce le informazioni per configurare l'IPC da un file di configurazione.
Esso lavora sui seguenti threads:

- BSM SENDER: questo thread è responsabile della codifica dei messaggi BSM in accordo con la convenzione dello standard ASN.1 [7] e manda lo stream di byte creato su un UDP socket al S&R. Se non sono connessi né il GPS né la rete CAN, viene mandato in broadcast un messaggio BSM con

valori di default letto da file di configurazione .cfg. La frequenza del broadcast è di 10 volte al secondo ma può essere configurato da configuration file .cfg.

- BSM Receiver: questo thread è responsabile della decodifica dello stream BSM ricevuto da S&R. Una volta che viene effettuato il parse dei pacchetti in accordo con la regola dello standard ASN.1 [33], viene inviato al modulo LDM.
- UC Event Receiver: questo thread si aspetta di ricevere sul socket dalle applicazioni UC quale tipo di evento ha bisogno di essere settato safe. I flag safety permessi sono descritti nello standard.
- GPS: questo thread acquisisce le informazioni GPS dal POTI mediator, le converte in accordo con lo standard [7] e riempie la struttura del BSM che sarà inviata in broadcast.
- CAN: questo thread acquisisce i dati CAN dalla libreria esterna DriverCan. Tale libreria gestisce i segnali CAN acquisiti dal VDPmediator, li converte in accordo con lo standard [7] e riempie la struttura del BSM che sarà inviata in broadcast.

I dati gestiti dal thread CAN riguardano la "Speed", il "Transmission State Steering", il "Wheel Angle", la "Longitudinal Acceleration", la "Lateral Acceleration", la "Vertical Acceleration", il "YAW rate" ed il "Brake status".

I dati gestiti dal thread GPS riguardano la "Latitude", l'Heading. L"Elevation" e la "Longitude Accuracy" non sono ancora gestiti.

I dati che restano sempre costanti durante la trasmissione sono il "Vehicle Width" e la "Vehicle Length".

È anche possibile settare il BS-BS da file di configurazione .cfg al fine di cambiare dinamicamente i seguenti campi del pacchetto inviato dai veicoli:

Dependencies	CAN	GPS	TemporaryID (BSM_CHANGE_VEHICLE_ID_PERIODICALLY = true; BSM_CHANGE_VEHICLE_ID_PERIOD_SECOND = 5)	SecMark (BSM_NOT_CHANGE_SECMARK)	SystemTime
MsgCnt	x	x	x	x	
SecMark				x	x
TemporaryId			x		
SystemTime					x

Figura 9: Configurazione del BS-BS

n possibile scenario, ad esempio, può verificarsi con la seguente configurazione:

il MsgCnt potrebbe cambiare se un dato proveniente dal thread CAN GPS (TemporaryID o SecMark) cambiano.

Se da file di configurazione i flag sono settati in questo modo:

- BSM_NOT_CHANGE_SECMARK = **true**
- BSM_CHANGE_VEHICLE_ID_PERIODICALLY = **false**

e il sistema non è connesso né alla CAN né al GPS, allora il MsgCnt non subirà alcuna modifica.

Tempi e statistiche

Il modulo BS-BS è stato testato con più di 16 veicoli ma bisogna tener conto del limite reale delle macchine le cui capacità consentono di processare le informazioni con una velocità di 10 pacchetti al secondo, limite oltre il quale si genererebbe latenza.

Per poter migliorare queste prestazioni al fine di ridurre questo delay, bisogna modificare il thread receiver. Infatti, la fase che riguarda la decodifica e l'invio dei pacchetti all'LDM richiede un tempo ancora troppo elevato per processare più di 10 veicoli al secondo.

Architettura
Il BS-BS è un applicativo che offre un servizio interponendosi tra il Sender-&Receiver (S&R) e l'LDM. Di seguito sono riportati i collegamenti tra i vari moduli del Framework:

INPUT ricevuti da:

- Casi d'uso, per ottenere il flag da settare nel BSM safety. I casi d'uso coinvolti sono: UC-EEBL, UC-SV, UC-CLW.
- S&R, per ottenere i pacchetti BSM al fine di codificarli.

OUTPUT verso:

- LDM, per fornire i pacchetti BSM decodificati.
- S&R, per inviare in broadcast i pacchetti 10 volte al secondo.

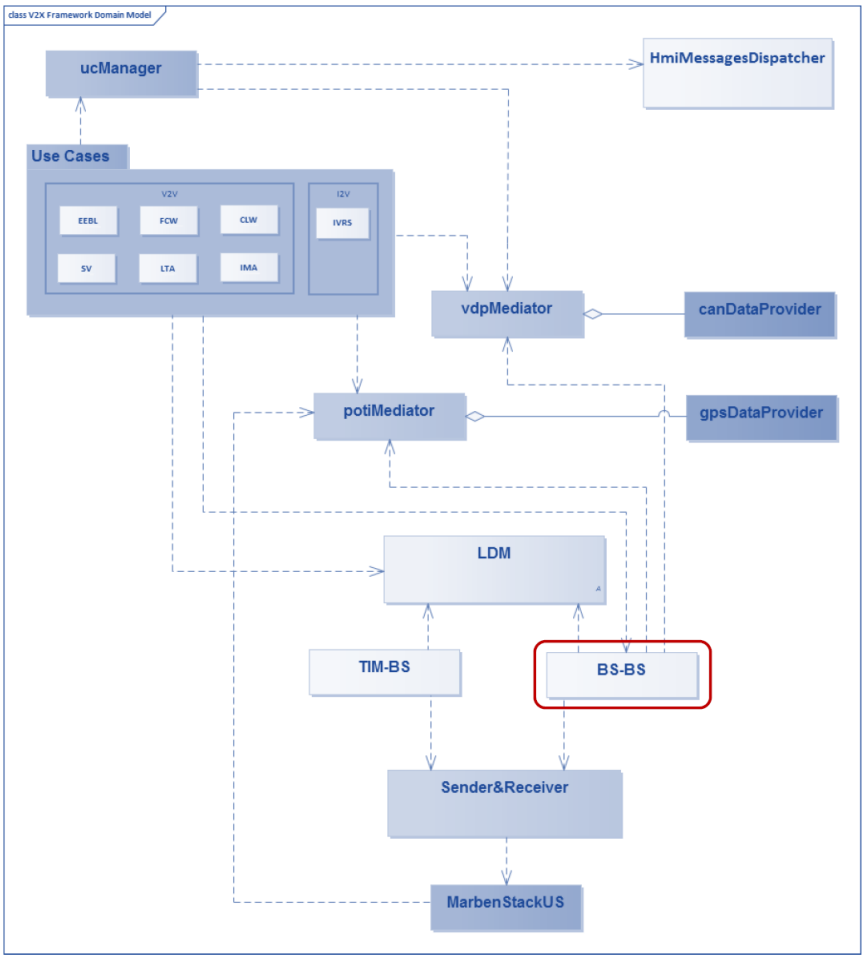

Figura 10: V2X Framework Domain Model

5.1.2 Sottoscrizione all'LDM

La Local Dynamic Map (LDM) è una struttura fondamentale all'interno dei Cooperative Intelligent Transport Systems (C-ITS). Fornisce un supporto utile alle applicazioni ITS in quanto conserva informazioni sugli oggetti che influenzano o sono influenzati dal traffico stradale. Le applicazioni ITS richiedono le informazioni essenzialmente su due grandi categorie di oggetti:

- Veicoli in movimento nelle vicinanze
- Oggetti fissi presenti sulla strada come i segnali stradali

L'LDM si comporta quindi da database situato in una stazione ITS che raccoglie i dati provenienti da altre applicazioni ITS, ne verifica la validità, controllando che provengano da una fonte autorizzata e infine li memorizza.

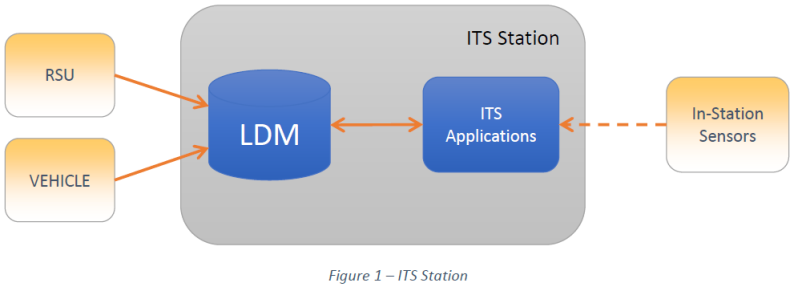

Figura 11: Stazione ITS

Le informazioni inviate o richieste all'LDM sono generate da due attori: DataProvider e DataConsumer. L'LDM fornisce un'interfaccia per abilitare un applicativo o una facility per registrarsi come LDM DataProvider e DataConsumer e per inviare/richiedere oggetti all'LDM.

Le informazioni inviate e/o richieste al modulo LDM sono originate da due attori, ossia i DataProvider e i DataConsumer. I primi sono degli applicativi autorizzati a fornire diversi tipi di dati all'LDM, i secondi invece sono degli applicativi autorizzati a richiedere dei dati dall'LDM. In particolare, i Data consumer possono richiedere i dati in due diverse modalità, la prima eseguendo una vera e propria registrazione verso l'LDM per un certo tipo di dato, se questa va a buon fine il consumatore riceverà costantemente aggiornamenti per il dato richiesto, questa modalità è denominata Notify.

La seconda modalità attraverso la quale un consumatore può richiedere un dato è la Query Puntuale, ossia la possibilità di interrogare il database su un determinato dato, tale meccanismo è utile per il dataconsumer in quanto non sempre potrebbe essere interessato a riceve costantemente gli aggiornamenti di una determinata informazione, ma soltanto in un preciso istante di tempo.

Funzionalità di memorizzazione

La funzionalità di base dell'LDM è quella di fornire un repository di informazioni per le facility e le applicazioni.
Le facility come il "Basic Safety Basic Services" (BS-BS) o il "Traveller Information Massage Basic Services" (TIM-BS) possono memorizzare informazioni all'interno dell'LDM.

La Local Dynamic Map si trova quindi tra il Middleware layer e l'Application layer. Gli input vengono ricevuti dal DataProvider con differenti messaggi V2X:

- Basic Safety Basic Services (BS-BS)
- Traveller Information Message Basic Services (TIM-BS)
- Signal Phase and Timing Basic Services (SPAT-BS)
- MAP Basic Services (MAP-BS)

L'output è diretto verso il DataConsumer: Caso d'uso (UC). Le applicazioni possono ricevere informazioni dal database che contiene una tabella per ogni tipo di messaggio. Ogni volta che l'LDM riceve un messaggio, questo va inserito o aggiornato nel database locale. Le tabelle contengono le informazioni sui vari campi del messaggio e riguardano il BSM, il TIM e lo SPATMAP.

Di seguito viene mostrata la tabella del messaggio BSM poiché il pacchetto costituisce l'oggetto di questo lavoro di tesi. Contiene, infatti, le informazioni riguardanti la latitudine e la longitudine dei veicoli e risulta indispensabile per il tracciamento e la visualizzazione dello scenario sul sito web.

BSM		
Field Name	**Field Type**	**Constraint**
msgID	UNSIGNED BIGINT	NOT NULL
MsgCnt	UNSIGNED BIGINT	NOT NULL
stationID	UNSIGNED BIGINT	NOT NULL
SecMark	UNSIGNED BIGINT	NOT NULL
Latitude	UNSIGNED BIGINT	NOT NULL
Longitude	UNSIGNED BIGINT	NOT NULL
Elev	UNSIGNED BIGINT	NOT NULL
Accuracy	UNSIGNED BIGINT	NOT NULL
Speed	UNSIGNED BIGINT	NOT NULL
Heading	UNSIGNED BIGINT	NOT NULL
Angle	UNSIGNED BIGINT	NOT NULL
AccelLong	UNSIGNED BIGINT	NOT NULL
AccelLatSet	UNSIGNED BIGINT	NOT NULL
AccelVert	UNSIGNED BIGINT	NOT NULL
AccelYaw	UNSIGNED BIGINT	NOT NULL
Brakes	UNSIGNED BIGINT	NOT NULL
VehicleWidth	UNSIGNED BIGINT	NOT NULL
VehicleLength	UNSIGNED BIGINT	NOT NULL
TransmissionState	UNSIGNED BIGINT	NOT NULL
event	UNSIGNED BIGINT	-
timestampValidity	UNSIGNED BIGINT	NOT NULL
PRIMARY KEY (stationID)	-	-

Tabella 1: Tabella pacchetto BSM

5.2 Frontend TSV

5.2.1 Mappe

Le mappe devono essere fornite da un tile server. Questo è un particolare tipo di server che è in grado di restituire tile della mappa a tutti gli host che si collegano all'indirizzo IP del server stesso. La velocità di risposta da parte del server può variare in base alla capacità hardware a disposizione e al numero di host che si collegano contemporaneamente.

Risulta particolarmente efficiente in questo senso, la possibilità di salvare in memoria cache, da parte dei dispositivi connessi tramite browser o applicativo HMI, i tile ricevuti dal server e di non richiederne altri nuovamente al server. In questo modo una volta elargite le "tessere" della mappa desiderate, il server non è più impegnato e può esaudire le richieste di altri client.

Ad ogni richiesta da parte dei client, viene visualizzato un report dei tile restituiti dal server indicando l'area geografica ed il livello di zoom selezionato. Il server deve garantire un rendering della mappa fino ad un livello di zoom impostato di 20.

La mappa che deve essere renderizzata riguarda l'estratto OSM di Venaria Reale e di Modena, città quest'ultima nella quale si è svolto l'evento ufficiale MASA nel Settembre 2018.

5.2.2 Web server e web site

Il web server deve garantire la connessione con l'Interpreter, il modulo che filtra i pacchetti BSM contenenti le coordinate GPS e che si trova sull'MK5, tramite una connessione UDP. Deve permettere di essere configurato tramite file di configurazione come tutti gli altri moduli del Framework e consentire di impostare:

- Indirizzo IP del Web Server.
- Porta su cui il Web Server riceve i dati dall'Interpreter.
- Porta su cui il Web Server riceve le richieste da parte del Web Site.

Il modulo Interpreter invia i dati ricevuti ogni 100 ms al Web Server utilizzando l'indirizzo IP e la porta settati. Il server riceve un pacchetto json in cui sono state salvate le informazioni. Vengono così estrapolate le coordinate geografiche del veicolo e sono pronte per essere inviate al web site. Ciò avviene in questo

modo: il client si invia una sola volta una richiesta al server utilizzando l'indirizzo IP e la seconda porta disponibile e si pone in ascolto. Da questo momento, ogni volta che il dato è pronto viene spedito al web site e letto.

Websocket

Per ottenere un meccanismo di scambio dati di questo tipo, è stato utilizzato il protocollo Websocket [14]. Esso serve per creare un canale bidirezionale tra il web server ed il web site, superando i limiti di HTTP e consentendo una comunicazione a bassa latenza. L'architettura è mostrata in figura:

Figura 12: Protocollo Websocket

Si riducono così i ritardi e ciò consente di ottenere dinamicamente le informazioni in tempo reale. Il client non è costretto a dover chiedere costantemente aggiornamenti al server e inoltre si evita di effettuare un numero di richieste che diventerebbe elevatissimo data la frequenza di aggiornamento dei messaggi che arrivano al server.

Il risultato è la ricezione costante dei messaggi ogni volta che il dato è pronto ottenendo un'esperienza sull'interfaccia web dinamica e fluida.

Il web site deve essere in grado di:

- Inviare una richiesta al web server per ricevere le coordinate geografiche dei veicoli.
- Visualizzare l'estratto geografico della mappa desiderata nella zona centrale del sito.
- Visualizzare menu laterali che permettono alcune configurazioni ed elenco dei layer da plottare sulla mappa.

La richiesta al web server viene inviata accedendo al menu laterale sinistro e premendo il pulsante di START. Verrà, così, inoltrata la richiesta al web server e da quel momento in poi ogni volta che il dato è disponibile, esso sarà ricevuto dal sito.

Verrà, inoltre, creato un marker e settata la posizione in real-time in base alle

coordinate dei veicoli ricevute: in tal modo sulla mappa si potrà seguire il percorso delle auto tramite i marker che si sposteranno dinamicamente sulla mappa.

Nel menu laterale destro posto in alto, si potranno notare i layer selezionabili del sito e uno di questi è nominato "SHOW CAR" e contiene tutti i marker collegati a ciascun veicolo che si sta muovendo sulla pista. Selezionando o deselezionando questo layer sarà possibile visualizzare o meno tutti i marker delle auto sulla mappa.

Tramite il menu situato sul lato sinistro, deve essere possibile effettuare alcune configurazioni. In particolare, si deve poter:

- Modificare un layer già presente di default nel sito;
- Aggiungere un layer nuovo.

L'aggiunta o la modifica dei layer avviene tramite file .xml: è possibile caricare, infatti, un file .xml che contiene le informazioni che si vogliono plottare sulla mappa e cambiare così l'aspetto di un layer predefinito o aggiungerne uno nuovo.

L'accesso al sito è possibile anche da smartphone e da tablet: grazie al demone di Apache infatti che viene messo in esecuzione sul web server, la risorsa è accessibile a tutti i dispositivi che si collegano all'indirizzo IP del server tramite browser o tramite HMI, Humane Machine Interface, applicazione Android di MM.

Anche su tali dispositivi, che possono essere sia smartphone che tablet, è possibile eseguire tutte le funzioni che offre il sito web, compreso il caricamento o la modifica dei layer. Il sito web, infatti, offre un'interfaccia grafica che si adatta allo schermo del dispositivo utilizzato.

5.3 Test librerie C++

La seconda parte di questo lavoro di tesi, riguarda l'analisi delle prestazioni dell'intero framework Magneti Marelli. In particolare, viene posta l'attenzione sul tempo impiegato dai moduli software per effettuare la scrittura dei messaggi nei file di log.

È stata effettuata, infatti, un'analisi delle prestazioni delle librerie di logging, per linguaggio C++ e durante la fase di test del framework MM, si è resa evidente la necessità di ottimizzare le prestazioni di ciascuno dei suoi moduli, date le limitazioni hardware dovute all'ambiente embedded. Ogni operazione di log, infatti, causa un ritardo, non trascurabile, che va ad inficiare la reattività di tutto il framework ed integrare una libreria di log più efficiente consente di rendere molto più veloce l'esecuzione di tutto il sistema, riducendo i ritardi e ottenendo più velocemente dati in tempo reale.

La libreria utilizzata nel framework MM è la libreria Boost che, essendo cross-platform, riusabile, open source ed efficiente nella gestione dello spazio di memorizzazione, si sposa bene con le esigenze del framework. Questa libreria di logging, inoltre, permette di effettuare la scrittura dell'output nel file di log tramite sei livelli diversi di log che sono:

- TRACE;
- INFO;
- DEBUG;
- WARNING;
- ERROR;
- FATAL.

Ogni modulo può loggare utilizzando uno di questi sei livelli e ciò permette di poter diversificare il tipo dei messaggi di output. Inoltre, tramite file di configurazione è possibile settare un parametro chiamato ***log_filter*** impostando uno di questi livelli. In questo modo nei file di log dei moduli si potranno visualizzare solo i messaggi di log di quel livello o di un livello superiore.

Ad esempio, se *log_filter* è impostato su WARNING, nel file di log del modulo verranno visualizzati esclusivamente i messaggi WARNING, ERROR e FATAL.

Lo scopo di questa parte del mio lavoro di tesi è stato quello di testare alcune librerie che potessero sostituire la libreria Boost e avessero un rendimento prestazionale più elevato. C'è stata la necessita infatti di migliorare le prestazioni del logging dei vari moduli del Framework al fine di diminuire i delay relativi

a queste operazioni di scrittura nei file di log e garantire una maggiore fluidità.

Durante l'esecuzione del Test Site Visualizer, la libreria Boost, infatti, si è rivelata poco efficiente e per ottenere un sistema realtime performante è risultato necessario dover ridurre i ritardi e migliorare la velocità di risposta di tutti i moduli del framework.

Bisogna, dunque, utilizzare librerie veloci ottimizzate per il linguaggio C++ che consentano di garantire la possibilità di loggare utilizzando quei 6 livelli di priorità e di scrivere un log di tipo rotazionale. Quest'ultimo tipo di log serve per poter creare dei file di log di dimensione moderata e di permettere la sovrascrittura dello stesso file una volta raggiunta la dimensione massima. Il log verrà così scritto sempre su questo stesso file.

Tale dimensione massima del file di log deve essere impostata da file di configurazione, impostando la dimensione in Megabyte dell'attributo **log_size_MB**. È possibile, inoltre indicare il percorso della directory in cui il file verrà salvato impostando l'attributo **log_path**.

La libreria che viene richiamata per eseguire il log è la libMMUtilityCommon che deve a sua volta richiamare la libreria oggetto di test. La libMMUtilityCommon serve per effettuare alcune configurazioni che riguardano ad esempio il nome del file di log, che è dato dal timestamp, la creazione della cartella nel path in cui salvare il file, passare il filtro di log alla libreria da testare, la dimensione massima del file espressa in MB, gestire la stringa di log creata dai vari moduli del Framework e la chiamata alle funzioni relative ai 6 livelli di log offerti.

Tramite settaggio delle *definitions* nella toolchainArm viene definita quale libreria di test deve essere richiamata dalla libMMUtilityCommon. Una volta definita la lib da richiamare, verrà inclusa quella specifica lib da testare e tutte le chiamate alla scrittura di log verranno effettuate utilizzando le funzioni di quella libreria.

6 Architettura e hardware

6.1 Test Site Visualizer

Il Test Site Visualizer è un sistema completo che permette la visualizzazione dei dati dei veicoli in tempo reale su una mappa OSM. Alla base di tutto il processo c'è l'Interpreter, che è un modulo che filtra le informazioni di base scambiate dalle vetture contenute nei messaggi BSM e li inoltra al web server. Quest'ultimo riveste un ruolo importante in quanto deve gestire la comunicazione sia con l'Interpreter sia con tutti gli host connessi in rete, inoltrando le informazioni ricevute utilizzando un oggetto json in modo che vengano visualizzate sui device.

Il web server svolge anche il ruolo di Tile Server in quanto è in grado di elargire i "tile", frammenti di dati geografici prerenderizzati, di mappa OSM in modo da mappare i veicoli utilizzando le coordinate geografiche ricevute. Questa operazione è svolta dal web site utilizzando la libreria Leaflet che permette di gestire correttamente i dati geografici. L'architettura del TSV è mostrata in figura 13.

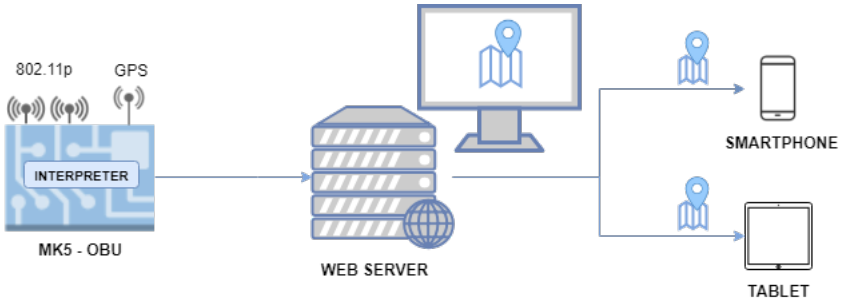

Figura 13: Architettura Test Site Visualizer

Il servizio di rendering delle mappe OSM offerto dal web server è messo in esecuzione utilizzando Renderd. Quando l'utente naviga la mappa del web site, viene effettuata una richiesta al web server dei tile desiderati, indicandone le coordinate e il livello di zoom. La richiesta viene gestita dal metodo OSMTileRequest() che elargisce il frammento di mappa richiesto. Il web site provvede, dunque a gestire la visualizzazione della mappa utilizzando il metodo della libreria Leaflet chiamato setMap().

Questo processo viene eseguito automaticamente ed è "trasparente" all'utente, mentre il servizio di tracciamento dei veicoli è attivato premendo il tasto "Start" del sito. In particolare, viene eseguita una richiesta al web server il quale salva le informazioni del client in modo che quando riceverà il dato dall'Interpreter, potrà inoltrarglielo. L'Interpreter, dunque, invia le informazioni dei veicoli al

web server che le inoltra al web site. Quest'ultimo le visualizza sulla mappa utilizzando il metodo setVehicle(). Il sequence diagram che rappresenta questo processo è mostrato in figura 14.

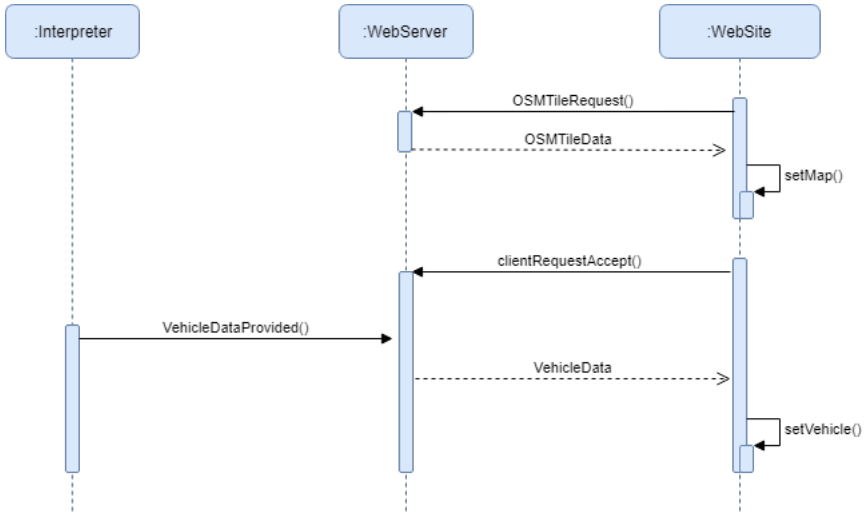

Figura 14: Sequence diagram TSV

6.2 Backend TSV

6.2.1 Hardware

Il modulo Interpreter viene eseguito su una board MK5-OBU, On Board Units, di Cohda Wireless che costituirà l'hardware utilizzato in questa prima parte della tesi. La board possiede le seguenti caratteristiche:

- IEEE 802.11p
- IEEE 1609
- Processore Embedded
- Sistema GNSS
- Security co-processor
- USB 2.0 host e interfaccia OTG

Figura 15: MK5-OBU

- CAN bus interfaces
- Ethernet interface
- HDMI
- Micro CD card slot
- 12V & 24V operation

Questa board è stata progettata per le comunicazioni Vehicle-to-Vehicle (V2V) e per le interazioni tra veicolo e le infrastrutture esterne, Vehicle-to-Infrastucture (V2I). Tale comunicazione è appunto basata sullo standard IEEE 802.11p che risulta essere la tecnologia più performante in questo campo.

6.2.2 Modulo INTERPRETER

L'Interpreter costituisce il fulcro del backend del Test Site Visualizer e viene eseguito sulla board Cohda appena descritta. Il modulo si comporta da sniffer dei messaggi BSM scambiati tra i veicoli. Per fare ciò si registra all'LDM per poter ottenere queste informazioni con un particolare ID che deve essere quello di un caso d'uso appartenente alla lista di ID validati dall'LDM. Una volta effettuata la registrazione, l'Interpreter diventa effettivamente un DataConsumer e riceve i pacchetti BSM ogni 100 ms.

Requisiti funzionali

Il modulo deve:

- Ricevere pacchetti BSM ogni 100 ms;
- Controllare l'origine dei pacchetti;
- Interpretare i pacchetti ed estrapolarne le informazioni necessarie;
- Creare un oggetto json contenente i dati necessari per il web site;
- Comunicare con il web server;
- Inviare l'oggetto json utilizzando un socket UDP.

L'esigenza di inviare i pacchetti ogni 100 ms viene soddisfatta eseguendo una richiesta, da parte del modulo software Interpreter, di tipo Notify all'LDM.

A differenza della richiesta effettuata tramite una Query Puntuale, che fornisce un determinato dato in un preciso istante di tempo, la modalità Notify permette all'Interpreter, dopo aver effettuato la registrazione, di ricevere una notifica

ogni volta che il dato nell'LDM è pronto.

6.2.3 Architettura MK5-OBU con Interpreter

L'Interpreter si comporta da DataConsumer, sottoscrivendosi all'LDM ed effettuando una query puntuale per ottenere i dati desiderati. Richiede i messaggi scambiati dai veicoli di tipo BSM. Essi contengono le informazioni di base dei veicoli, come le loro coordinate geografiche. Queste vengono ottenute dal POTI.

Esso è un modulo costituito dal POTI Mediator, che inoltra le informazioni ai moduli BSM e SPAT, e dal GPS Data Provider che invia i dati al primo.

Ciascuno affidato ad un thread. In questa fase i pacchetti vengono gestiti e vengono estrapolate le informazioni volute. Questi dati confluiscono in un unico metodo che si occupa della schedulazione per poter poi essere inviati. L'architettura che presenta l'MK5-OBU, con l'integrazione dell'Interpreter, è mostrata in Figura 16.

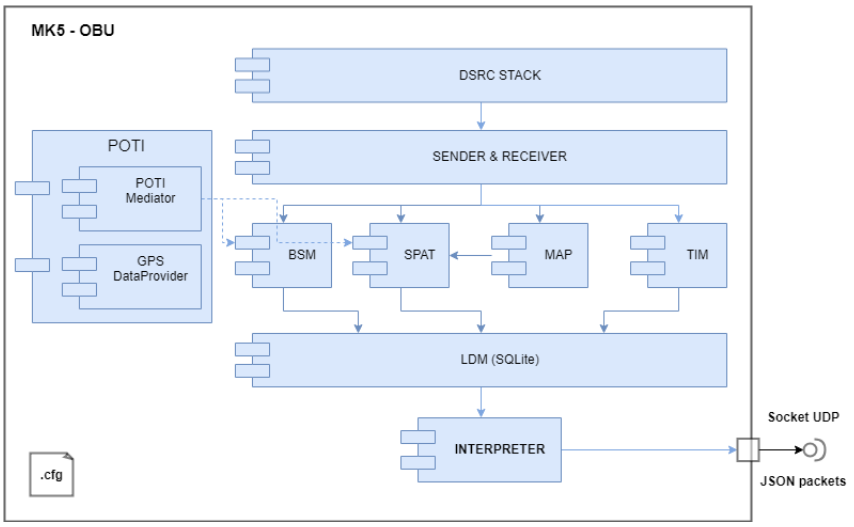

Figura 16: Architettura MK5-OBU con Interpreter

Registrazione Data Consumer

L'Interpreter invia una richiesta all'LDM utilizzando un ID specifico per potersi registrare. La richiesta di autorizzazione è quindi inoltrata al Security Manager che autorizza la ricezione dei dati richiesti.

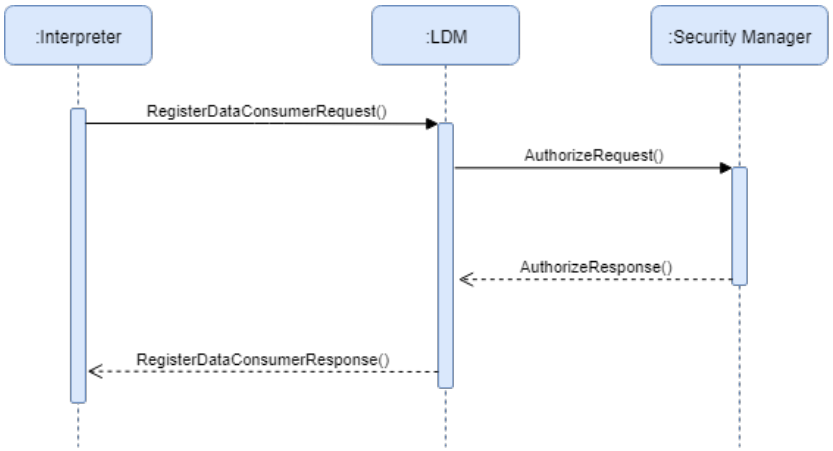

Figura 17: Registrazione Data Consumer

Il "sequence diagram" che rappresenta il processo di sottoscrizione dell'Interpreter è mostrato in Figura 17. Questo modulo esegue la sottoscrizione al Local Dynamic Map (LDM), modulo del framework MM che gestisce il database contenente le informazioni ricevute o richieste da due tipi di attori: i Dataprovider e i DataConsumer.

I primi sono gli applicativi autorizzati a fornire diverse tipologie di messaggi all'LDM, i secondi sono gli applicativi autorizzati advdfvdvdd interrogare il database, richiedendo il dato in modalità Notify oppure eseguendo una Query Puntuale. L'Interpreter esegue il primo tipo di richiesta e riceve una notifica ogni volta che un dato aggiornato è disponibile nell'LDM. Si ottiene, dunque, un Binary Large Object (BLOB), contenente le informazioni desiderate.

Questo processo è alla base del funzionamento dell'Interpreter, permettendogli in questo modo di svolgere la sua funzione di sniffer. Grazie ai dati ricevuti dall'LDM, può costruire un oggetto json che verrà inviato al web server e permetterà la visualizzazione dei veicoli.

6.3 Frontend TSV

Il frontend del Test Site Visualizer è composto da due grandi parti: il web server ed il web site.
Il web server deve essere in grado di assolvere i seguenti compiti:

- Ricevere i dati provenienti dal modulo Interpreter.
- Effettuare una connessione con il web site.
- Inviare al web site i dati riguardanti i veicoli.
- Inviare al web site i tile della mappa per permettere la visualizzazione dell'area geografica.

6.3.1 Mappe

OpenStreetMap è utile per la creazione di dati geografici e la loro fornitura gratuita a chiunque li desideri: negli anni è stata realizzata una mappa della superficie terrestre rilasciata con una licenza libera per l'utilizzo e la modifica. I dati raccolti dal progetto vengono memorizzati in un database appositamente predisposto: la conversione degli stessi sotto forma di informazione visuale può avvenire mediante diverse piccole immagini, dette "tessere" (o tile), la cui generazione e diffusione pubblica può gravare significativamente sulle prestazioni di un server.

Sebbene esistano tile server, come "tile.openstreetmap.org", gratuiti e liberamente accessibili, essi sono in genere resi disponibili da volontari e non hanno a disposizione grandi risorse di calcolo e di connettività di rete, per cui uno sviluppatore potrebbe dover rivolgersi a servizi a pagamento oppure mettere su un proprio tile server. Quest'ultimo riguarda lo svolgimento della seconda parte di tesi e svolge un ruolo essenziale per il funzionamento del sito web.

Nella sezione implementativa descriverò il processo di installazione e la configurazione di un tile server OpenStreetMap su una macchina Ubuntu Linux 18.04.1 LTS e la creazione dell'applicazione che mostra una mappa utilizzando i tile messi a disposizione dal server stesso.

Al termine si otterrà un tile server perfettamente funzionante a partire da un sistema installato con un profilo minimale: si inizierà scaricando ed installando le diverse componenti per la memorizzazione delle informazioni OpenStreetMap, la loro resa sotto forma di immagini e la gestione delle priorità di utilizzo. Un particolare modulo del server HTTP Apache sarà utilizzato come interfaccia pubblica del sistema.

Un ruolo essenziale nella creazione del tile server lo riveste la scelta della macchina server (fisica o virtuale) su cui effettuare l'installazione, al fine di poter

portare a termine il processo di installazione dello stesso. Si richiede un modesto quantitativo di spazio di archiviazione, un disco veloce e grande quantitativo di RAM, con processore almeno dual core: in questo progetto di tesi è stato utilizzato il car-PC, un pc ubuntu dotato di 8 processori e 30 Gib di RAM. L'intero processo di installazione ha richiesto circa un'ora.

Il web site deve:

- Visualizzare i dati ricevuti su una mappa.
- Caricare sia staticamente che dinamicamente i layer sulla mappa.
- Visualizzare o nascondere i layer e i dati del veicolo.

Car-PC

Il web server ed il web site sono eseguiti sul car-PC, un pc basato su Ubuntu che ha le seguenti caratteristiche:

- **OS**: Ubuntu 18.04.1 LTS
- **Memoria RAM**: 30,4 GiB
- **Processore**:
  Intel® Core™ i7-6820EQ CPU @ 2.80 GHz x 8
- **Tipo di sistema**:
  sistema operativo a 64 bit.
- **Disco**: 491,2 GB

Figura 18: Car-PC

Il web server offre due servizi: il primo è quello di gestire la comunicazione con l'Interpreter e con gli host connessi in rete, a cui inoltrare le informazioni ed il secondo è quello di agire come Tile Server, utile per fornire i tile prerenderizzati della mappa OSM e viene collegato ad un monitor su cui è possibile visualizzare il sito web.

In Figura 19, viene riportata l'architettura dei moduli utilizzati nel Car-PC e verrà spiegato come avviene la comunicazione tra essi.

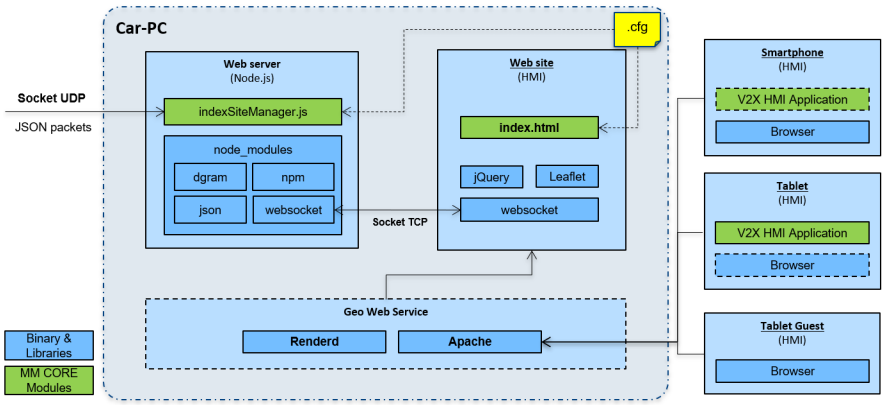

Figura 19: Architettura Test Site Visualizer - Web server e web site

La parte di comunicazione riguardante i dati dei veicoli è gestita dal modulo Node.js. Il web server è stato infatti programmato in javascript per assolvere tale compito. Il file di start è "indexSiteManager.js" e viene lanciato per far partire l'esecuzione di questa sezione del server.

Per gestire la comunicazione con l'Interpreter utilizza il modulo "dgram", che permette di effettuare uno scambio di pacchetti utilizzando il protocollo UDP.

Npm è il gestore dei pacchetti per javascript ed è stato utilizzato per installare alcuni moduli utili come json e websocket. Il primo server per poter leggere i dati provenienti dalla board, dal momento che sono inviati in formato stringa json, e per creare un pacchetto da inoltrare ai client.

Il secondo permette di utilizzare il protocollo Websocket che serve per poter effettuare una comunicazione TCP con il client a bassa latenza. Infatti, crea un canale bidirezionale per mezzo del quale il client può effettuare una richiesta univoca al server e costui può procedere all'inoltro delle informazioni ogni volta che il dato è pronto.

In questo modo si evitano continue richieste da parte del client rendendo la comunicazione molto più efficiente, evitando inutile overhead sul server.

I due servizi offerti dal server sono possibili avviando due demoni: renderd e apache. Il primo si occupa di effettuare il rendering dei tile di mappa e il secondo permette di accedere alle risorse web, tra cui il web site.

Ciascun client che si trova nella stessa rete del web server, può collegarsi all'indirizzo della risorsa web ed accedere al web site. Quest'ultimo è accessibile utilizzando un browser o l'applicazione V2X HMI Magneti Marelli utilizzando sia uno smartphone che un tablet.

La pagina di start del web site è "index.html" tramite la quale gestisce la mappa OSM, la comunicazione con il server e le impostazioni volute dall'utente. L'interfaccia grafica offre la possibilità di effettuare le configurazioni e di cambiare gli elementi impostati di default utilizzando la libreria jQuery che permette di implementare le funzionalità Ajax.

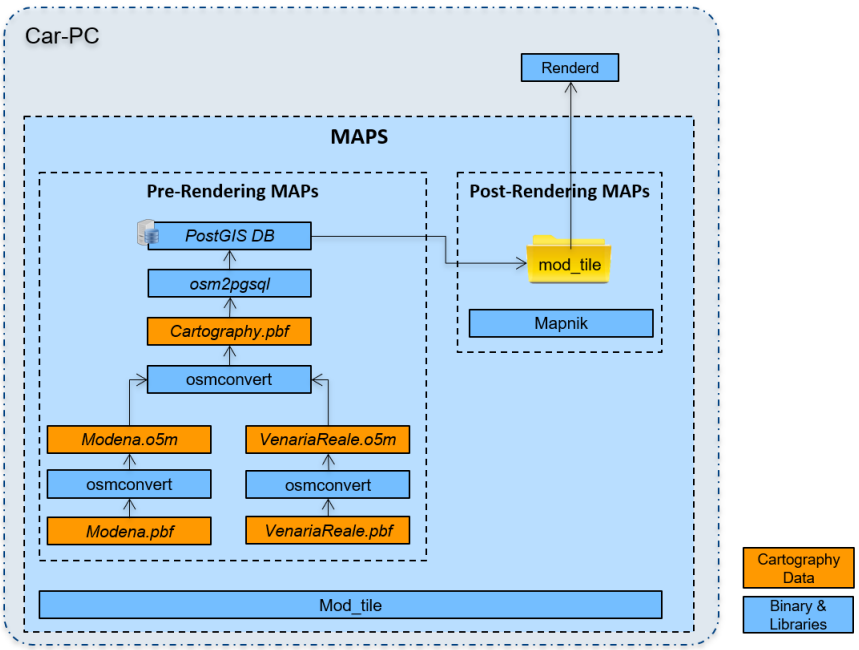

Figura 20: Architettura Test Site Visualizer - Mappa

Per la gestione della mappa OSM viene utilizzata la libreria Leaflet, che

permette di utilizzare i metodi che servono a mappare correttamente i "tile" di mappa OSM ricevuti dal tile server. Infine, la comunicazione con il web server, effettuata per ricevere i dati dei veicoli, si basa su websocket, lo stesso protocollo utilizzato lato server.

L' architettura del Test Site Visualizer che concerne la mappa OSM è mostrata in Figura 20. Renderd è il demone che effettua il servizio di rendering, permettendo al web server di svolgere la funzione di tile server, elargendo i tile di mappa OSM richiesti dai client, che possono essere device come smartphone e tablet. Questi tile sono stati ottenuti in più fasi.

Sono stati scaricati da OpenStreetMap, progetto open source che offre mappe modificabili con licenza libera, gli estratti di mappa OSM di Modena e di Venaria Reale in formato .pbf e utilizzando il tool osmconvert, sono stati convertiti in formato .o5m in modo da poterne effettuare il merge, utilizzando lo stesso tool.

Si ottiene dunque il dato cartografico contenente gli estratti desiderati e, utilizzando il tool osm2pgsql, viene salvata nel database per mappe PostGIS. Per elargire i tile, Renderd utilizza Mapnik che permette di salvare i dati cartografici in mod_tile, effettuando il caching e permettendo di eseguire il rendering in tempo reale dei tile e di fornirli ai client.

Per quanto riguarda la creazione dei layer da raffigurare sulla mappa OSM ottenuta, il circuito e gli Overlay Layer sono stati realizzati utilizzando il tool JOSM, un editor estendibile di OpenStreetMap, grazie al quale sono state costruite le polyline, tracciandole in corrispondenza dei punti di riferimento le cui coordinate geografiche sono state ottenute dai rilevamenti effettuati sulla pista di Modena.

Grazie all'ausilio del tool JOSM è possibile quindi creare oggetti, come polyline e layer, che possono essere aggiunti alla mappa in modo da poter visualizzare informazioni aggiuntive sulla pista ad un livello di dettaglio maggiore.

6.4 Test librerie C++

Per effettuare il test sulla velocità di logging delle librerie C++ è stata utilizzata la Step3, una board che ha le seguenti caratteristiche:

- **CPU**:
  Processore ARv7, 4 core
- **RAM**: 1 GB
- **Disco**: 512 MB

Figura 21: Step 3

Sulla board viene caricato tutto il Framework MM e viene fatto partire per testarne le prestazioni. Ciascun modulo logga il proprio output su di un file di log che viene salvato in un percorso impostato da file di configurazione e di dimensione settata anch'essa nello stesso modo.

Il processo di log da parte dei moduli avviene in questo modo: il modulo all'avvio controlla se è abilitato il logger, controllando la definition MMLogger, e in tal caso richiama initLogger, passandogli il nome del modulo e il livello di log.

Una volta inizializzato il logger, il modulo può loggare il proprio output richiamando la funzione relativa al livello di log desiderato. La gestione di queste chiamate è affidata alla libreria libMMUtilityCommon che si occupa di gestire sia l'inizializzazione che la scrittura dei log, ma non solo, si occupa anche di effettuare alcune configurazioni iniziali: essa, a sua volta, per effettuare la scrittura vera e propria del messaggio di log, richiama la libreria di logging C++ Boost, utilizzando le sue funzioni offerte per effettuare il log.

Ogni volta che un modulo richiama il logger per scrivere su file, verrà richiamata la funzione corrispondente.

7 Implementazione

7.1 Backend TSV

7.1.1 Interpreter

La prima parte di questo lavoro di tesi prevede la creazione di un modulo software, chiamato Interpreter, capace di ricevere i pacchetti BSM, Basic Safety Message, scambiati tra i veicoli e l'infrastruttura di rete e tra i veicoli stessi, secondo lo standard SAE j2735. Questi messaggi contengono le informazioni di base del veicolo che ha generato il pacchetto.

L'Interpreter esegue la sottoscrizione al Local Dynamic Map (LDM), modulo del framework MM che gestisce il database contenente le informazioni ricevute o richieste da due tipi di attori: i *Dataprovider* e i *DataConsumer*.

I primi sono gli applicativi autorizzati a fornire diverse tipologie di messaggi all'LDM, i secondi sono gli applicativi autorizzati ad interrogare il database, richiedendo il dato in modalità *Notify* oppure eseguendo una *Query Puntuale*. L'Interpreter esegue il primo tipo di richiesta e riceve una notifica ogni volta che un messaggio BSM aggiornato è disponibile nell'LDM.

Si ottiene, dunque, un Binary Large Object (BLOB), contenente le informazioni desiderate. Ciò avviene ogni 100 ms. Una volta ottenuto il BLOB, il modulo ne estrapola i campi interessati ai fini della mappatura dei veicoli sul sito web.

Vengono, dunque, ottenuti l'ID, la latitudine e la longitudine del veicolo, dopo aver effettuato una opportuna conversione. Ottenute queste informazioni, il modulo ha il compito di costruire un pacchetto JSON in cui memorizzarle. Una volta creato l'oggetto JSON, questo deve essere inviato al web server che è fondamentale per il funzionamento del sito web.

In particolare verrà utilizzata una connessione UDP poiché è l'ideale per poter inviare pacchetti quel numero elevato di volte. L'oggetto viene così inviato al server.

In Figura 22 è mostrato il "component diagram" dell'Interpreter che rappresenta l'architettura software del Test Site Visualizer. Dalla figura si evincono i moduli utilizzati del Framework e l'operazione di invio dei dati scritti in un pacchetto JSON sul socket UDP.

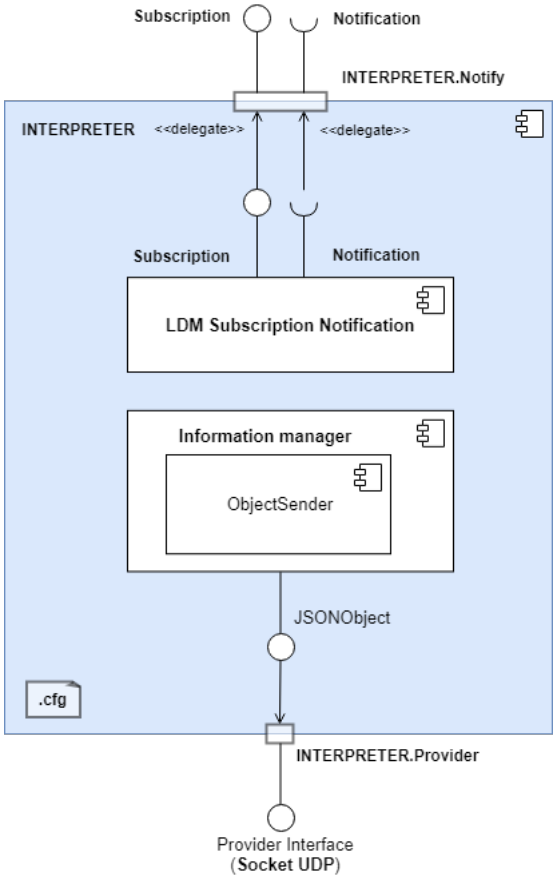

Figura 22: Component diagram Interpreter

Funzionamento

Il modulo Interpreter legge da file di configurazione il proprio Application ID, l'indirizzo IP e la porta dell'LDM e l'indirizzo IP e porta del web server.

Nella funzione principale vengono settati i parametri per iscriversi all'LDM come *DataConsumer*: viene impostato un filtro pari a NULL, viene istanziato un dataObject di tipo BSM e creato un DataConsumer passandogli come parametri l'id del modulo Interpreter, il dataObject, l'indirizzo IP e porta dell'LDM.

Da notare che l'Application ID è univoco in quanto l'LDM accetta sottoscrizioni da parte dei vari moduli controllando una lista di ID e non si possono effettuare iscrizioni da parte di due moduli con lo stesso ID.

Pertanto ciascun caso d'uso ha un ID assegnato e quello impostato per l'Interpreter da file di configurazione è l'ID 9.

Settato il DataConsumer, viene lanciato un thread, chiamato "*threadfunc*", richiamando il metodo requestSubscription e passandogli il filtro. Dopodiché si effettua una sleep di 2 secondi necessaria affinché il DataConsumer completi la registrazione prima di effettuare richieste.

A questo punto, si entra in un ciclo while, interrompibile durante l'esecuzione del modulo sulla board tramite un "*signal*", in cui si esegue la richiesta del dataObject, come definito dallo standard TS 102 894-2 [15], e quando il dato è pronto, ne si seleziona il tipo di dato BSM ottenendo così il BLOB, Binary Large Object, contenente tutte le informazioni del messaggio Basic Safety Message.

Dal Blob si estraggono: l'ID, la latitudine e la longitudine del veicolo che sta trasmettendo i dati. Questi dati vanno convertiti utilizzando la conversione ntohl (network to host long) e si dividono "lat" e "long" per il valore "10000000.0" per ottenerne il valore corretto.

Questi dati vengono ottenuti utilizzando la decodifica secondo lo standard SAE J2735.

Una volta che le informazioni desiderate sono state estrapolate, si crea un oggetto JSON in cui salvare ordinatamente i dati, utilizzando la libreria libjson inclusa nel Framework. L'oggetto json creato ed inviato è raffigurato in Figura 23.

```
jsonObject =
{
        "ID" : "113",
        "LAT" : "45.1437329",
        "LONG" : "7.62328920"
}
```

Figura 23: Oggetto JSON inviato dall'Interpreter

L'oggetto creato viene formattato creando una stringa json e viene, dunque, eseguito il thread che invia il messaggio al web server, chiamato *threadSenderBSM*, passandogli l'oggetto stesso.

In questo thread viene utilizzato un socket UDP di tipo MMipcSocket settando indirizzo IP e porta del web server. Il messaggio contenente le coordinate geo-

grafiche del veicolo viene così spedito al web server.

Eseguendo una richiesta di tipo *Notify*, l'Interpreter riceve una notifica ogni volta che un messaggio BSM aggiornato è disponibile nell'LDM e ciò accade con una frequenza di 10 volte al secondo. Il messaggio fornirà quindi continui aggiornamenti sulla posizione dei veicoli. In particolare, vengono inviati i messaggi di tutti i veicoli che sono presenti nell'area di interesse e verranno poi filtrati dal web server in base all'ID.

Infine, il modulo software viene integrato nel Framework MM e viene eseguito utilizzando il file di configurazione "config_MM_WAVE_TSV.cfg" che contiene le configurazioni iniziali necessarie per essere inizializzato correttamente.

L'Interpreter viene eseguito su una board e, durante il test in laboratorio, esso riceve i messaggi BSM che vengono inviati da una seconda board su cui è stato avviato il framework. Filtrati i messaggi, si ottiene l'ID della board in esecuzione e le informazioni che contengono le coordinate del veicolo, simulando in talo modo il funzionamento del sistema in vettura.

Osservando l'output dell'Interpreter, inoltre, è possibile evincere se c'è un malfunzionamento di uno dei moduli del framework, in quanto è possibile rilevare la mancata ricezione delle informazioni da un particolare ID in esame.

7.2 Frontend TSV

In questa seconda parte di tesi mi sono occupato dell'implementazione del servizio web direttamente connesso al modulo Interpreter. Il sistema è composto da un web server, che con l'integrazione del servizio di Rendering delle mappe OSM svolge anche il ruolo di Tile Server, e da un web site.

7.2.1 Web server

Il web server è stato implementato utilizzando Node.js, piattaforma open source utile per l'esecuzione di codice javascript lato server. È stata effettuata questa scelta perché Node.js ha un'architettura event-driven e permette di gestire I/O asincroni, ottimizzando il throughput e la scalabilità dell'applicativo web che prevede numerosi operazioni di input/output e la gestione della comunicazione real-time con il web site.

Node richiede al sistema operativo di ricevere notifiche al verificarsi di determinati eventi, e rimane quindi in sleep fino alla notifica stessa: solo in tale momento torna attivo per eseguire le istruzioni previste nella funzione di callback, così chiamata perché da eseguire una volta ricevuta la notifica che il risultato dell'elaborazione del sistema operativo è disponibile. Tale modello di networking è risultato più efficiente nelle situazioni critiche in cui si verifica un elevato traffico di rete.

Il web framework utilizzato per Node.js è stato **Express**. Esso costituisce un miro-framework che offre strumenti veloci per l'applicazione in Node. È stata creata, dunque, una express app: utilizzando npm, il gestore di pacchetti per Javascript, è stato creato il file package.json per l'applicazione.

Il framework Express, inoltre, permette di specificare alcune informazioni come ad esempio il nome e la versione dell'app. Ma il campo più importante è l'entry point: questo sarà il file principale che verrà usato per lanciare l'intera applicazione.

È stato specificato come entry point il file indextsvserver.js. Infine, è stata installato Express nella directory app ed è stato salvato nell'elenco delle dipendenze, lanciando il comando:

```
$ npm install express --save
```

L'applicazione legge da file di configurazione, chiamato configTSV.json, l'indirizzo IP del server, la porta su cui ricevere i dati provenienti dall'Interpreter e la porta su cui ricevere la richiesta da parte del web site. Per assolvere il primo compito è stato utilizzato il modulo "**dgram**". Esso permette di creare un socket UDP, in quanto la comunicazione MK5-WebServer avviene utilizzando questo protocollo, dal momento che essendo connectionless consente di ottenere velocità e una minore congestione della rete rispetto a TCP.

Creato il socket UDP il server si pone in ascolto sulla prima porta, chiamata "TSV_Port". Verranno ricevuti qui i dati inviati dall'Interpreter con una frequenza di 1 messaggio ogni 100 ms. Quando il server viene avviato, il nome del processo utilizzato è "NODEJS_WEBSERVER", utile in caso si voglia interrompere la sua esecuzione da riga di comando.

Per implementare la sezione real-time e gestire lo scambio dati con il web site è stato incluso il modulo "**websocket**". Websocket è stato utilizzato perché è un protocollo che permette di creare un canale bidirezionale veloce tra server e browser.

Esso, infatti, supera i limiti del protocollo HTTP, riuscendo a creare una comunicazione in tempo reale veloce e a bassa latenza. Il server deve consentire infatti l'invio del dato appena esso è pronto, evitando di attendere che sia il client a farne richiesta continuamente.

Tramite Websocket il server riconosce l'evento e lo notifica al client appena si verifica: ciò rende l'esperienza della navigazione del sito web continua e ininterrotta poiché il browser comunica costantemente con il server, consegnando i dati aggiornati immediatamente piuttosto che quando richiesto, ottenendo così un sistema dinamico e fluido.

Websocket si basa su TCP, ma ne costituisce un'implementazione. Creato il socket, dunque, il server si pone in ascolto sulla seconda porta, quella destinata a ricevere richieste da parte dei client. La comunicazione è impostata in modo tale che dal web site parta una sola volta la richiesta di ricevere i dati in tempo reale, ricevuta la quale il server salva il riferimento in una lista di client: appena il dato è ricevuto dall'Interpreter, esso dal momento che è sotto forma di stringa json viene ripulito eliminando i caratteri speciali. Si crea un nuovo oggetto in cui si settano il tipo, impostato su "message", e l'object che contiene la stringa con i dati geografici utili al web site.

Quest'ultima viene dunque inviata al client utilizzando la lista riempita precedentemente quando è stata ricevuta la richiesta di connessione. Ogni volta che il dato è pronto, dunque, esso verrà subito notificato al client, realizzando così l'applicazione real-time.

7.2.2 Mappe - Tile server

Per poter rendere disponibili i tile della mappa il web server è stato implementato in modo che assolvesse anche il ruolo di Tile server. Esso è un particolare tipo di server che tramite opportune configurazioni è in grado di accettare connessioni dagli host e distribuire i tile di mappa richiesti.
Il tile server è stato installato sul car-PC su cui è presente un sistema operativo Ubuntu 18.04.1 LTS. I moduli essenziali alla creazione del tile server sono:

- **Mapnik**, Mapnik utilities
- **Apache**
- Mod_tile
- Python
- Yaml
- Openstreetmap Carto
- PostgreSQL e PostGIS
- Osmconvert e Osm2pgsql
- **Renderd**

Mapnik [9] è un toolkit open source che ho utilizzato per effettuare il rendering dei tile della mappa. Viene utilizzata da Renderd. Inoltre, tramite Mapnik è stato possibile personalizzare tutti gli aspetti grafici di essa come le caratteristiche dei dati, icone, caratteri motivi ed alcuni effetti grafici 3d, definendo le regole di stile in linguaggio XML.

Apache HTTP Server [10] è stato utilizzato per creare un server web. Viene fatto partire il demone tramite il comando

```
$ sudo service apache2 start
```

il quale permette l'accesso al sito web che si trova in /var/www/html/ e di gestire alcuni aspetti della sicurezza. La configurazione viene effettuata modificando il file httpd.conf.

Mod_tile [11] è un modulo di Apache. Viene utilizzato per poter fornire i tile della mappa desiderati e di effettuare il caching. Permette di effettuare il rendering in real-time e la sua strategia di caching garantisce performance elevate e consente di gestire quasi mille richieste al secondo.

Openstreetmap Carto [12] è un preprocessore utilizzato per i fogli di stile Mapnik. I fonts che ho installato necessari per Openstreetmap Carto sono:

- noto-emoji

- noto-fonts

PostgreSQL [23] è un modello di base di dati ad oggetti (ODBMS) e PostGIS è un'estensione spaziale di esso. Una volta creato il database

```
$ psql -U postgres -h $HOSTNAME -c "CREATE DATABASE gis
ENCODING 'UTF -8' LC_COLLATE 'en_GB.utf8' LC_CTYPE
'en_GB.utf8' TEMPLATE template0"
```

si crea l'utente e si garnatisce l'accesso a gis DB

```
$ psql -U postgres -c "create user marelli;grant
all privileges on database gis to marelli;"
```

e si abilita l'accesso remoto a PostgreSQL modificando il file di configurazione /etc/postgresql/10/main/pg_hba.conf aggiungendo la riga

```
host all all 0.0.0.0/0 md5
```

e il file /etc/postgresql/9.5/main/postgresql.conf settando l'indirizzo di ascolto come '*':

```
listened addresses = '*'
```

Terminato il processo si ottiene un database di questo tipo:

Name	**Owner**	**Encoding**	**Collate**	**Ctype**	**Access privileges**
gis	postgres	UTF8	en_GB.utf8	en_GB.utf8	=Tc/postgres postgres=CTc/postgres marelli=CTc/postgres

Tabella 2: Database gis

Per consentire un rendering ad alte prestazioni, sono stati settati i seguenti parametri:

- shared_buffers = 128MB
- min_wal_size = 1GB
- max_wal_size = 2GB
- work_mem = 32MB
- maintenance_work_mem = 256MB
- autovacuum = off
- fsync = off

Una volta completata la configurazione del database, si procede l'immissione dei dati geografici all'interno del database. Per effettuare questa operazione vengono utilizzati i due tool Osm2pgsql [13] e Osmconvert [34].

Il primo è un programma basato su riga di comando e converte i dati OpenStreetMap in un formato idoneo alla memorizzazione nel database PostgreSQL appena creato ed esteso con PostGIS. In questo modo i dati OSM saranno resi visivamente utilizzando Mapnik dal momento che PostgreSQL è uno dei formati più efficienti e flessibili che Mapnik può utilizzare per interrogare grandi quantità di dati.
Il secondo è servito per convertire ed elaborare i dati OpenStreetMap.

Sono stati quindi scaricati dal sito ufficiale http://osm-estratti.wmflabs.org/-estratti/regioni/pbf/ gli estratti di mappa di Venaria Reale e di Modena e sono state effettuate le seguenti operazioni:

- conversione dei due file .pbf in file con estensione .o5m

```
$ osmconvert VenariaReale.pbf -o=extractVenariaReale.o5m
$ osmconvert Modena.pbf -o=extractModena.o5m
```

- esecuzione del merge dei due estratti in un'unica mappa .pbf

```
$ osmconvert extractVenariaReale.o5m extractModena.o5m
-o=mergedAllMap.pbf
```

Ottenuta la mappa finale, essa deve essere salvata nel database PostGIS. La procedura di caricamento dei dati sul database, che viene detta ***infasamento***) è particolarmente intensa dal punto di vista dell'I/O, e potrebbe richiedere diverse ore, se non giorni, a seconda delle prestazioni del server e della dimensione del file scelto.

In questo lavoro di tesi, tenendo conto dell'hardware a disposizione, il processo di installazione dei tile riguardanti Venaria Reale e Modena ha richiesto pochi minuti, mentre l'installazione della mappa contenente tutta la regione italiana ha richiesto circa un'ora.

Per effettuare questa operazione viene lanciato il comando seguente:

```
$ osm2pgsql --slim -d gis -C 8000 --number-processes 3
-c -G --hstore --style openstreetmap-carto.style
--tag-transform-script openstreetmap-carto.lua -d gis
-H $HOSTNAME -U postgres mergedAllMap.pbf
```

Bisogna porre l'attenzione su due settaggi in particolare:

- **-C 8000**: indica il numero di chunk

- **--number-processes 3**: in media il numero ottimale è dato dal numero di core del processore - 1

Al termine di questo processo, i dati sono stati caricati nel database PostGIS. Ora bisogna creare degli indici per rendere più veloci le query e garantire l'accesso a tutte le tabelle gis per evitare di provocare errori quando l'utente 'marelli' tenta di effettuare l'accesso alle stesse. Infine bisogna aggiungere gli indici di geometria parziali indicati da openstreetmap-carto. Il databse conterrà dunque le seguenti tabelle:

Schema	Name	Type	Owner
public	planet_osm_line	table	postgres
public	planet_osm_nodes	table	postgres
public	planet_osm_point	table	postgres
public	planet_osm_polygon	table	postgres
public	planet_osm_rels	table	postgres
public	planet_osm_roads	table	postgres
public	planet_osm_ways	table	postgres
public	spatial_ref_sys	table	postgres

Tabella 3: Tabelle database gis

Da questo momento i tile della mappa sono pronti per essere forniti in rete. Questa operazione è svolta da Renderd, un demone che una volta lanciato fornisce un sistema di code con priorità per l'effettuazione del rendering dei tile.

Bisogna effettuare le seguenti modifiche al file /usr/local/etc/renderd.conf:

```
[renderd]
stats_file=/var/run/renderd/renderd.stats
;socketname=/var/run/renderd/renderd.sock
num_threads=4
tile_dir=/var/lib/mod_tile

[mapnik]
plugins_dir=/usr/lib/mapnik/3.0/input/
font_dir=/usr/share/fonts
font_dir_recurse=true

[default]
URI=/osm_tiles/
TILEDIR=/var/lib/mod_tile
XML=/home/marelli/src/openstreetmap-carto/style.xml
HOST=localhost
```

```
TILESIZE=256
```

e modificare il file /etc/init.d/renderd nel modo seguente:

```
DAEMON=/usr/local/bin/$NAME
DAEMON_ARGS="-c /usr/local/etc/renderd.conf"
RUNASUSER=marelli
```

Ora si fa partire il servizio di rendering lanciando Renderd:

```
$ sudo -u username renderd -f
-c /usr/local/etc/renderd.conf
```

L'ultimo passaggio è quello di fa partire Apache così da rendere tutto disponibile sull'indirizzo IP del server e lo si aggancia al servizio di Renderd. Si procede a modificare, pertanto, il file /etc/apache2/sites-enabled/000-default.conf aggiungendo dopo la riga <VirtualHost *:80> le seguenti righe:

```
#Load all the tilesets defined in the configuration file into
#this virtual host
LoadTileConfigFile /usr/local/etc/renderd.conf

#Socket where we connect to the rendering daemon
ModTileRenderdSocketName /var/run/renderd/renderd.sock

# Timeout before giving up for a tile to be rendered
ModTileRequestTimeout 3

# Timeout before giving up for a tile to be rendered that
#is otherwise missing
ModTileMissingRequestTimeout 60
```

e si esegue il restart di Apache con il comando

```
$ sudo systemctl restart apache2
```

In questo modo rendiamo il server web attivo e il sito web disponibile on-line. Vengono fatti partire due demoni: Renderd e Apache. Il primo permette di effettuare la fornitura dei tiles a tutti gli host che si collegano tramite un meccanismo di caching, elargendo i dati caricati sul database PostGIS e sfruttando il servizio di Mapnik e di Mod_tile. Apache permette l'accesso al sito web da parte di tutti gli host che si collegano all'indirizzo IP del server web.

Per testare il funzionamento del tile server senza ricorrere all'utilizzo del sito web, si può semplicemente collegarsi tramite un browser all'url:

```
http://IP_Server/osm_tiles/0/0/0.png
```

e si potrà visualizzare l'immagine di un planisfero come rappresentato in figura:

Figura 24: Planisfero

Per visualizzare alcune statistiche alcune informazioni riguardanti i tile restituiti dal server, basta collegarsi all'url:

```
http://IP_Server/mod_tile
```

Infine, i tile vengono creati come 'metatile' e salvati nella cartella /var/lib/-mod_tile.

7.2.3 Web site

Il web site è stato sviluppato in Javascript e HTML5; è stato utilizzato Ajax per leggere i file di input, il linguaggio XML e i fogli di stile CSS. Per quanto riguarda le librerie, è risultato essenziale l'utilizzo di leaflet.js e jquery.js.

La prima è una libreria javascript open source che ha permesso di gestire le mappe OSM da visualizzare nel sito e di crearne una ottimizzata per diversi tipi di dispositivi mobili, consentendo di mostrare non solo punti di interesse, linee e aree ma anche livelli interattivi, chiamati layer, da visualizzare sulla mappa.

Layout

Il sito web è diviso in tre sezioni: quella principale e le due laterali. La prima è quella più importante, posta nella zona centrale del sito e che occupa quasi interamente il display: consente la visualizzazione della mappa, di scaricare e di rappresentare i layer, e di effettuare uno zoom di 20 livelli, superando quello di default di 18, per consentire di visualizzare anche dettagli maggiori del circuito.

Il lato destro presenta un menu con delle checkbox contenenti il tipo di mappa che si vuole ottenere, versione stradale o satellitare, e i layer che si possono visualizzare. Questi oggetti sono stati creati utilizzando i layer group per creare i diversi layer e il layer control per gestirli. Sono stati utilizzati due tipi di layer:

- Base layer
- Overlay layer

I **base layer** sono mutualmente esclusivi in quanto solo uno di essi può essere visibile sulla mappa nello stesso momento, prestandosi pertanto ad essere utilizzato per selezionare il tipo di mappa, stradale o satellitare, da visualizzare. Gli **overlay layer** sono invece i livelli che si possono sovrapporre alla mappa e servono ad aggiungere dei dettagli su di essa: si possono infatti evidenziare aree geografiche e aggiungere marker o polyline.

La mappa satellitare è generata richiedendo i tile da google, indicando l'indirizzo "http://{s}.google.com/vt/lyrs=s&x={x}&y={y}&z={z}", mentre la mappa stradale è generata ottenendo i tile restituiti dal Tile server implementato in questa seconda parte di tesi.

Ci si collega all'indirizzo IP del web server e tramite la libreria leaflet, in base alla posizione in cui si vuole visualizzare la mappa geografica, vengono riconosciute

le coordinate dei tile da scaricare, salvate nelle variabili x, y, z, si effettua la richiesta di quei frammenti al tile server e vengono caricati nel layer Osm_Street in questo modo:

```
var Osm_Street = L.tileLayer(
        "http://localhost/osm_tiles/{z}/{x}/{y}.png",
        {
          maxNativeZoom:20,
          maxZoom:24
        }
);
```

Viene impostato un livello di zoom nativo pari a 20, superiore a quello di default pari a 18. Per rendere attiva questa modifica bisogna configurare il tile server ed in particolare il servizio di Renderd: nel file /src/mod_tile/includes/render_config.h bisogna settare la definizione #define MAX_ZOOM 20 e nel file /usr/local/-etc/renderd.conf bisogna aggiungere il parametro MAXZOOM=20.

In questo modo, si ricevono i tile di mappa anche ad un livello di zoom superiore e si ottengono dettagli maggiori sul circuito. Nell'interfaccia web si può effettuare lo zoom fino ad un livello pari a 24.

Per creare gli overlay layer si utilizzano i layer group, associandone ciascuno ad un nome univoco. Dopodiché si crea un control layer che accetta come primo parametro il base layer e come secondo l'overlay layer e si aggiunge alla mappa. Il risultato è il seguente:

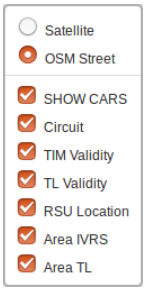

Figura 25: Control layer

Gli overlay layer aggiunti di default nella pagina web sono sette, essi sono: Circuit, TIM Validity, TL Validity, RSU Location, Area IVRS, Area TL ed infine Show cars layer. Il primo rappresenta la pista che è stata utilizzata per l'evento ufficiale MASA. Il TIM-Validity rappresenta l'area di validità del messaggio TIM, Traveler Information Message, il quale fornisce informazioni aggiuntive sullo stato del traffico o sulla segnaletica stradale.

Il TIM non ha una durata infinita, esso ha un periodo di tempo limitato per cui resta valido ed inoltre può essere definita anche una area di validità fuori dalla quale esso perde di significato; è rappresentato dall'area rettangolare di colore blu. Il TL-Validity indica la posizione dei due traffic light presenti sulla pista; è rappresentato da un'icona semaforica. L'RSU-Location indica la posizione della Road Side Unit, il dispositivo posto sul ciglio della strada che fornisce supporto di connettività ai veicoli in transito; è rappresentato da un piccolo cerchio verde.

L'area IVRS, indica la "In Vehicle Road Sign", rappresentata da un'area rettangolare evidenziata di colore blu ed infine l'area TL che indica la validità del semaforo ed è rappresentata da un'area rettangolare evidenziata di colore verde. Una rappresentazione della mappa con tutti i layer sovrapposti può essere osservata nella figura seguente, in cui è impostato come base layer quello stradale OSM, utilizzato durante la prova, contenente i tile ricevuti dal web server:

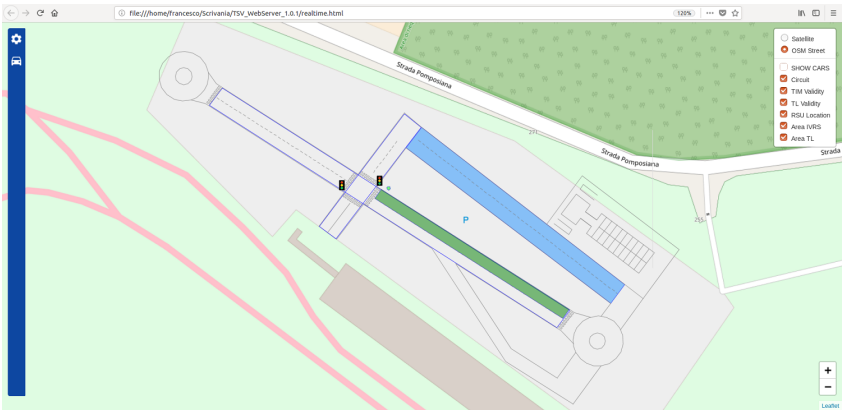

Figura 26: Visualizzazione base e overlay layer

Ogni layer viene "costruito" e raffigurato a run-time, leggendo in input un file .xml. Per ottenere un file .xml che si interpretabile dal browser e che rispetti i punti geografici desiderati, è stato utilizzato il tool JOSM. Esso è un editor che legge i dati OSM e permette di poter realizzare delle polyline da visualizzare sulla mappa.

Creazione del file di input

Nell'editor JOSM viene scaricata la mappa satellitare OSM relativa alla zona di interesse, riguardante la pista dove sarà effettuata la demo, accanto all'Autodromo di Modena. Una volta ottenuta la mappa, si può ora procedere a segnare i punti geografici sulla mappa e disegnare una polyline in modo che rappresenti l'area che si vorrà visualizzare poi come layer. I punti geografici sono stati

rilevati sul circuito e sono stati segnati nell'editor, utilizzando la latitudine e la longitudine: in questo modo è stato possibile ricreare esattamente il circuito tramite polyline.

Il risultato è un file .xml che presenta tutti i nodi utilizzati per rappresentare il circuito, indicati dai tag e da un id univo. In questo file ci sono, inoltre, i riferimenti per collegare i nodi e costruire le figure geometriche desiderate. Affinché questo file venga correttamente interpretato dal browser, bisogna effettuare un processo di pulizia in modo che si eliminino le informazioni superflue e si estrapolino esclusivamente i tag <node> che contengono le coordinate geografiche dei punti e i tag <way> che indicano quali sono i nodi che costituiscono figure geometriche. È stato creato un script che esegue questa operazione, chiamato Script_OSMtoXML.sh, e genera un file come il seguente:

```
<description>
        <node id='-39145' lat='44.63518900506' lon='10.81394853293'/>
        <node id='-39618' lat='44.63516883631' lon='10.81395303642'/>
        <node id='-39638' lat='44.63518459131' lon='10.81394383907'/>
        <way id='-89562'>
                <nd ref='-39145'/>
                <nd ref='-39618'/>
                <nd ref='-39638'/>
                <nd ref='-39145'/>
        </way>
</description>
```

Figura 27: Input file .xml

Il tag <**description**> funge da container per tutti i tag. È indispensabile per la lettura di tutto il file. Il tag <**node**> include le informazioni relative ai nodi e contiene l'id del nodo che è univoco, la latitudine e la longitudine. Il tag <**way**> è descritto da un id univoco e server per elencare tutti i nodi che compongono una polyline: presenta come sottolivello, infatti, un inner tag <**nd**> che contiene l'id univoco del nodo. L'insieme di tutti questi nodi contenuti in un tag way rappresenta una polyline. Qualora non si voglia utilizzare l'editor JOSM, bisogna creare un file .xml di questo tipo affinché esso possa essere correttamente letto dal browser.

Creazione della polyline e caricamento del file

I file di input .xml vengono tutti salvati nel percorso /inputFilesXML. Vengono letti da questa cartella utilizzando un metodo Ajax di tipo GET che effettua il parsing del file .xml. Cancellate eventuali vecchie polyline create, si estrae il con-

tenuto dei tag <way>, <nd> e <node> tramite il metodo "querySelectorAll" e si individuano innanzitutto i poligoni, indicati dai tag <way>. Dopodiché si leggono i sottotag <nd> che contengono il riferimento a tutti i nodi del poligono e si accede così al nodo specifico confrontando il riferimento al nodo contenuto nel tag <nd> e l'id del nodo stesso contenuto nel tag <node>, ottenuto utilizzando il metodo "getAttribute".

Individuato il nodo, si estraggono le coordinate geografiche "lat" e "lon" e si costruisce la polyline utilizzando il Vector layer "Polyline" offerto dalla libreria Leaflet oppure il Vector layer "Polygon". Nel caso di caricamento di immagini si utilizza un marker, la cui icona corrisponde all'immagine desiderata, e si posiziona in corrispondenza delle coordinate lette dal file .xml.

Nel menu laterale sinistro, nella sezione Configuration, è possibile effettuare alcune modifiche sui layer: in particolare, è possibile modificarne uno preesistente oppure caricarne uno nuovo; entrambe le operazioni richiedono di caricare un file .xml.

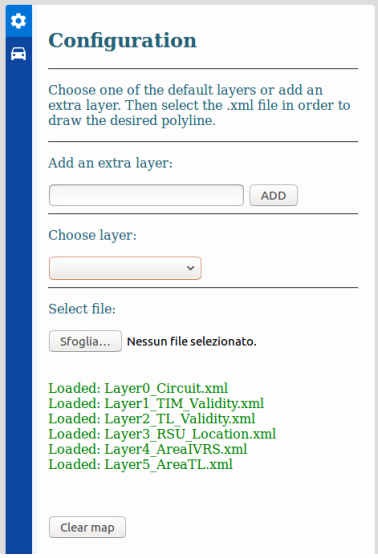

Figura 28: Menu Configuration

Nel caso in cui si voglia modificare un layer preesistente, bisogna scegliere il layer da modificare dal menu a tendina, chiamato "**Choose layer**", selezionarlo e caricare un file .xml costruito come descritto nel paragrafo precedente, selezionandolo con il tasto Sfoglia. La polyline associata al layer da modificare verrà cancellata, sarà ricalcolata la nuova ed associata al layer. Si può inoltre aggiungere altri layer, scrivendone il nome nella casella "**Add an extra layer**"

e caricare il file .xml associato. Il nuovo layer verrà aggiunto alla lista dei layer disponibili e verrà mostrato anche in una nuova checkbox nel menu laterale destro posto in alto.

Comunicazione real-time

Nel menu laterale sinistro, nella sezione Real time, si può inizializzare la comunicazione con il web server, premendo il tasto "**Start**".

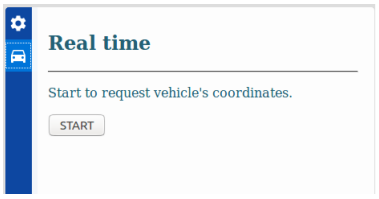

Figura 29: Menu Real time

Quando viene premuto il tasto Start, ha inizio la comunicazione con il web server e dinamicamente si inizieranno a ricevere informazioni in tempo reale da visualizzare sulla mappa. In particolare, viene utilizzato anche dal lato browser il protocollo **websocket**. Esso è risultato particolarmente efficiente ed essenziale per la creazione dell'applicazione real-time. Tramite websocket è possibile ottenere una comunicazione asincrona e full-duplex attraverso una singola connessione TCP.

Tramite questa sessione di comunicazione interattiva tra il browser e il server si possono mandare messaggi al server e ricevere risposte event-driven senza doverle richiedere al server. Ed è ciò che server per la realizzazione di questo applicativo: dal momento che il dato sul server è ricevuto con una frequenza di 1 volta ogni 100 ms, il numero richieste che dovrebbe fare il client al server sarebbe elevatissimo, comportando uno spreco di banda e un overhead sul server, in quanto quest'ultimo riceverebbe numerose richieste ed anche da più client contemporaneamente.

Da considerare anche che il dato non sarebbe ricevuto in tempo reale. I dati vengono trasferiti tramite una WebSocket come messaggi, ognuno dei quali è costituito da uno o più frame contenente ciascuno i dati scambiati. Ogni frame è preceduto da 4-12 byte di dati sul carico utile e ciò consente di ridurre la quantità di dati non a carico utile trasferiti, con conseguente riduzione significativa della latenza. Tale connessione è mantenuta per ogni singolo client, permettendo di inviare in real-time dati a uno o più client appena il server lo ha disponibile.

Si ottiene così un vero real-time: l'unico delay che rimane è quello del trasferimento dati. L'overhead lato server è ridotto: se non ci sono nuovi dati

disponibili il server non viene continuamente bombardato da inutili richieste. Infine, è garantita una bassa latenza e nessun spreco di banda.

Funzionamento pagina web

Il layer principale del sito web è impostato come base layer di tipo stradale, chiamato Osm_Street, il quale richiede e visualizza i tile di mappa OSM elargiti dal Tile Server. Il centro della mappa è impostato sulla pista dove è stata effettuata la demo ufficiale durante l'evento MASA, a Modena, settando le coordinate geografiche del circuito e uno zoom pari a 18. Il layer Osm_Street è aggiunto alla mappa in questo modo:

```
var map = L.map("map",
{ layers:
              [Osm_Street]
}).setView(
              [44.6351696, 10.8142081], 18
);
```

Oltre al settaggio del layer principale, in fase di load della pagina web avviene il caricamento di tutti gli overlay layer: vengono creati i group layer associandone ognuno ad un nome univoco per ciascun layer e vengono aggiunti al control layers in modo tale che essi compaiano nel menu laterale destro che fornisce la lista dei layer visualizzabili.

Ora bisogna costruire le polyline da associare a ciascun layer. Per fare questo viene richiamato un metodo, detto "loadAllInputFiles" che si occupa di leggere da file di configurazione quali sono i file .xml associati a ciascun layer di default, li legge dalla cartella /inputFilesXML e richiama il metodo "creaCircuito" per ognuno di essi. Questo metodo procede ad effettuare il parse, descritto precedentemente, del file e associa la polyline che viene costruita durante la lettura dei dati del file, al group layer associato all'overlay layer da visualizzare. L'aggiunta viene fatta in questo modo:

```
nome_Polyline.addTo(nome_GroupLayer);
```

si può inoltre settare il colore della polyline che verrà raffigurata sul circuito, lo spessore della linea e l'opacità. Anche i poligoni e le immagini possono essere aggiunte in questo modo al group layer. Le icone e le immagini venongo caricate anch'esse nella fase di loading, dalla cartella /images.

La modifica di un layer preesistente avviene scegliendo il layer desiderato ed il file .xml che si vuole sostituire: sarà richiamato il metodo che crea la polyline relativa, ma prima dovrà essere cancellata quella preesistente in questo modo:

```
nome_Layer.clearLayers();
```

Su ciascun layer può essere effettuata questa operazione di cancellazione, tramite la quale vengono eliminate tutte le polyline associate ad essi. Ciò è possibile premendo il tasto "Clear map", il quale richiama un metodo che effettua la cancellazione su tutti gli overlay layer, lasciando inalterato quello di base contenente la mappa OSM.

Per quanto riguarda la creazione di layer aggiuntivi bisogna effettuare alcune operazion preliminari. In particolare, dopo aver ottenuto dal form il nome del layer che si vuole aggiungere, si controlla innanzitutto se non è NULL o se un layer con lo stesso nome è già esistente, ed in tal caso si chiede di immettere un nome diverso, e si crea un array di layer aggiuntivi, detti extraLayers, in cui si aggiunge il layer desiderato.

Tale array viene utilizzato per aggiungere i layer aggiuntivi al control layer, in modo che essi vengano aggiunti al menu laterale destro e siano selezionabili per essere visualizzati sulla mappa. Vengono così inseriti come overlay ed aggiunti a quelli preesistenti, tramite il seguente metodo:

```
controlLayers.addOverlay(
       array[nome_extraLayer], nome_extraLayer
);
```

L'array viene utilizzato, inoltre, per aggiungere i layer aggiuntivi al rispettivo group layer, in modo che possonano poi essere eliminati anch'essi quando si effettua la cancellazione di tutti gli overlay layer, tramite il tasto "Clear map". Infine, l'array è utile quando viene richiamato il metodo "creaCircuito", in quanto la polyline che viene creata viene aggiunta al group layer associato all'extra layer che si vuole aggiungere, estrapolandolo dall'array.

Per quanto riguarda il funzionamento real-time, una volta premuto il tasto "Start" viene richiamato il metodo "getCoordInterpreter" che è quello che si occupa di instaurare la connessione con il web server ed ottenere le coordinate inviate dal modulo Interpreter. Da file di configurazione, chiamato configTSV.json e situato nella cartella /cfg, viene letto l'indirizzo IP del web server e la porta sulla quale accetta le richieste del client.

Viene dunque aperta la connessione utilizzando il protocollo websocket, per i motivi precedentemente descritti, impostando l'url: "ws://" + configTSV.WEBSITE_IP + ":" + configTSV.WEBSITE_Port e inviata la richiesta al server, contenente i dati del client. Il server potrà così dunque salvare le informazioni del client in una lista di client connessi e inviare i dati richiesti.

Il client sarà in ascolto, tramite il metodo "connection.onmessage", dell'oggetto JSON inviato dal server, senza dover effettuare numerose richieste al server: sarà il server che notificherà al client il dato in tempo reale, non appena esso è ricevuto dal modulo Interpreter, realizzando così un'esperienza d'uso per l'utente continua ed ininterrotta e permettendo di visualizzare il dato costantemente

aggiornato, riducendo drasticamente i delay, che saranno solo quelli dovuti all'effettiva trasmissione del dato. Una volta che il pacchetto è stato ricevuto, si controlla il campo type che deve essere settato su "message" e si procede così a leggerne il payload.

Per visualizzare la posizione dei veicoli, si utilizza un marker con il simbolo di un autoveicolo, la cui posizione sarà settata in tempo reale non appena il dato aggiornato viene ricevuto. Si utilizza un array di marker in cui salvare un marker per ogni veicolo utilizzando come indice univoco l'ID del veicolo stesso.

Ricevuto il dato, si effettua, dunque, il parse dell'oggetto JSON e si estrapola l'ID del veicolo che ha originariamente inviato i dati. Si controlla che non sia già stato salvato nell'array un marker relativo a quel veicolo, controllando che l'ID non sia presente, e se ne crea uno: tale marker sarà salvato nell'array e sarà utilizzato per tracciare gli spostamenti del veicolo, settandone la posizione.

Quando viene ricevuto il dato aggiornato con le nuove coordinate dello stesso veicolo, viene utilizzato lo stesso marker, estrapolandolo dall'array, e ne viene settata la posizione.

Le coordinate si trovano nei campi "LAT" e "LONG" dell'oggetto JSON, vengono estrapolate, utilizzando rispettivamente il metodo "parseFloat(JSON.parse(mex).-LAT)" e "parseFloat(JSON.parse(mex).LONG)", e usate per settare la posizione del marker sulla mappa. Il tutto viene effettuato in questo modo:

```
arrayMarkers[ID_veicolo].setLatLng(
 L.latLng(LATITUDINE, LONGITUDINE));
```

Infine, il marker viene aggiunto all'overlay layer associato alla visualizzaZione di questi elementi, chiamato "Layer_coord", il quale viene creato come layer group e aggiunto agli altri overlay layer, con il nome di "Show cars". Nel menu laterale destro, il layer control, si potrà quindi selezionare e deselezionare tale layer per far visualizzare o meno i marker sulla mappa.

Dal momento che ogni dato aggiornato viene ricevuto con una frequenza di una volta ogni 100 ms, si ottiene una fluidità elevata e la visualizzazione e il tracciamento del percorso dei veicoli appaiono entrambi chiari e puntuali.

Visualizzazione

Di seguito, viene riportata una schermata del web site che mostra la visualizzazione dei veicoli, tramite marker, sulla mappa, mostrando la posizione geografica esatta sul circuito. I marker vengono posizionati dinamicamente e seguono il percorso dei veicoli in tempo reale. Per ottenere questo risultato, l'intero sistema del Test Site Visualizer, TSV, è stato messo in esecuzione.

In particolare, lato backend, l'Interpreter riceve i pacchetti BSM contenenti le coordinate geografiche che indicano la posizione corrente del veicolo che li sta generando. Tali informazioni sono inviate al web server, il quale, oltre a inoltrarle costantemente a tutti i client che effettuano la connessione sulla sua seconda porta, mantiene attivi due servizi necessari: il demone di Renderd, che serve per poter elargire i tile della mappa OSM e quello di Apache, che serve per rendere possibile l'accesso alla pagina del sito web.

Infine, sulla pagina web, dopo aver premuto il tasto "Start" ed abilitato il layer per visualizzare i veicoli, chiamato "Show cars", si ottiene il tracciamento in tempo reale dei veicoli che stanno percorrendo il circuito.

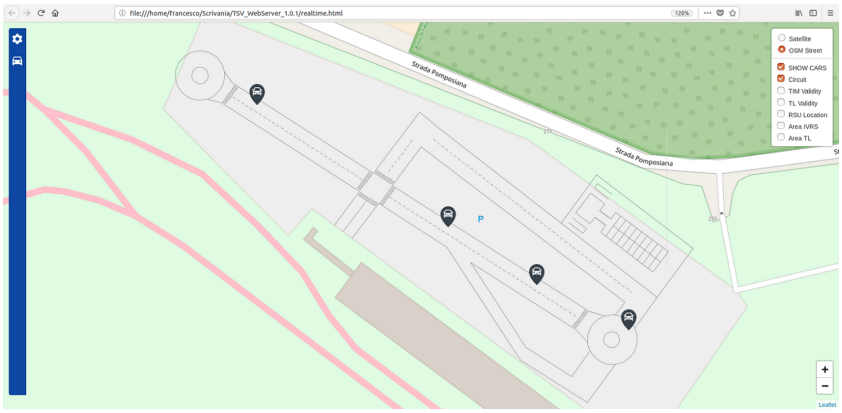

Figura 30: Visualizzazione veicoli in tempo reale

7.3 Test librerie C++

In quest'ultima parte di tesi, ho effettuato l'implementazione di 6 librerie di logging C++ al fine di effettuare dei test sull'impatto prestazionale di ognuna di esse su tutto il Framework Magneti Marelli. In particolare, l'utilizzo di una libreria di log veloce è essenziale in quanto si vanno a ridurre i delay riscontrati nella fase di log su tutti i moduli del Framework, andando a migliorare le prestazioni complessive di quest'ultimo.

Il primo passo è stato quello di andare a progettare e implementare un modulo, chiamato "modulotest", da integrare nel Framework il cui compito sia quello di "stressare" la libreria di logging da testare effettuando un numero di operazioni di log settabile da file di configurazione e utilizzando un numero di thread anch'esso settabile dallo stesso file. L'operazione ha comportato uno stress per la CPU della board sulla quale è stato effettuato il test ed ha comportato a volte anche l'utilizzo della stessa per divere ore, in base alla velocità della libreria di logging testata.

Il "modulotest", come tutti i moduli del Framework, richiama la libreria libMMUtility che è quella che gestisce tutto il processo di logging. In particolare, richiama la funzione initLogger che serve per inizializzare e configurare il logger all'avvio e vengono passati due parametri: il nome del modulo e il livello di log. Dopodiché il modulo può richiamare le funzioni offerte dalla libUtility per loggare, utilizzando quella opportuna per il tipo di log.

Nella funzione principale viene eseguito un numero di thread pari al valore di MAX_THREAD settato da file di configurazione, dopodiché si attende la terminazione di ciascuno di essi con una operazione di join. Ogni thread lanciato fa partire un numero di scritture su file di log pari al valore di MAX_LINES: verrà scritta una stringa contente il nome del modulo, il numero del thread che sta scrivendo e il numero della riga. Viene eseguito il calcolo in secondi del tempo richiesto per scrivere tutte le righe desiderate dal thread in questione e si salva il risultato in un array.

Una volta che tutti i thread hanno terminato, si esegue il calcolo della media tra tutti i tempi ottenuti da tutti i thread per scrivere tutte le righe di log e si fornisce il risultato finale.

La libMMUtility è la libreria che gestisce il processo di log: permette di eseguire la initLogger, la configurazione e i settaggi iniziali, e di richiamare le funzioni che scriveranno nel log. Ho aperto un branch di questa libreria, dal momento che essa richiamava esclusivamente la libreria Boost, in cui includo la libreria di test voluta in base alle definition: in particolare, utilizzando la toolchainArm per compilare la libreria, viene aggiunto in questo file tramite add_definition, la definizione della libreria che vogliamo testare cosicché quando la libMMUtility verrà compilata la richiamerà, si potranno così utilizzare le sue funzioni e le sue

configurazioni.

Nella initLogger viene letto il file di configurazione e viene impostato il path in cui salvare il file di configurazione, letto dal parametro log_path, viene creata la cartella con il nome del modulo che ha richiamato la init, la dimensione massima del file espressa in MB, letta da log_size_MB, il nome del file di log che corrisponde al timestamp corrente, e il filtro di log, letto da log_filter. Quest'ultimo si rivela particolarmente utile in quanto serve a settare un livello di filtro dei messaggi di log oltre il quale essi non verranno scritti sul file.

Sono impostati 6 diversi livelli di log ordinati secondo la gravità, essi sono: trace, debug, info, warning, error e fatal. Se il filtro di log è impostato sul livello info, ad esempio, tutti i messaggi che richiamano il log per scrivere messaggi di tipo trace o debug non verranno scritti nel file di log. Un'altra caratteristica che viene garantita è la possibilità di effettuare una scrittura del log di tipo rotazionale: in questo modo, raggiunta la dimensione massima del file di log, esso viene riscritto ottenendo così un file di log che non sfori mai la dimensione massima impostata da file di configurazione. L'output del log è impostato in modo da scrivere il timestamp, il livello di log ed il messaggio di output del modulo. Un esempio di output finale, in cui è utilizzato il livello di log di tipo INFO, è il seguente:

```
[2018-10-31 09:54:05.969326][INFO]MW:Packet received  ...
[2018-10-31 09:54:05.969458][INFO]******* MAP RECEIVED ********
[2018-10-31 09:54:05.969604][INFO]MW:Payload: 30 81 8c 80 01 07 81 01 - 00 83 01 03 84 01 01 a5 - 7a 30 78 81 04
 00 00 0c - 2e a2 10 80 04 02 a9 14 - 5f 81 04 00 a5 01 c8 82 - 02 00 00 a7 5e 30 2d a2 - 2b 80 0a 4e 6f 72 74 6
8 - 62 6f 75 6e 64 81 01 01 - a2 1a 30 18 80 01 01 81 - 02 01 f4 82 01 02 a3 0c - 04 04 02 90 fe eb 04 04 - 1c 7
2 ed c6 30 2d a2 2b - 80 0a 53 6f 75 74 68 62 - 6f 75 6e 64 81 01 02 a2 - 1a 30 18 80 01 01 81 02 - 01 f4 82 01
02 a3 0c 04 - 04 fc da 00 b1 04 04 ec - 25 0c 7f 87 02 00 00  - dim 143
[2018-10-31 09:54:05.969621][INFO]*******  END ********
```

Figura 31: Output libreria Nanolog C++

Ad ogni livello di log corrisponde dunque una funzione offerta dalla libMMUtility e richiamata dal modulo che vuole effettuare il log a quel livello specifico.

Infine, vengono compilate le librerie da testare che devono essere incluse nella libMMUtility. Si setta la definition della libreria che si vuole testare, si compila la libMMUtility in modo che includa solo quella e si carica sulla board il "modulotest", la libMMUtility e la libreria da testare. Si setta il file di configurazione del modulo in modo che esamini diversi scenari di test e si lancia l'esecuzione sulla board. Nel path di log relativo al "modulotest" verrà creato il file di log relativo alla libreria da testare.

Parte IV
Conclusioni

Conclusioni

In questa tesi è stata progettata ed implementata un'architettura software, che prende il nome di **Test Site Visualizer** (TSV), che permette di ricevere i pacchetti che vengono scambiati tra i veicoli e di visualizzare in real-time su una mappa la loro posizione corrente. Questo lavoro si suddivide in due parti: la prima riguarda lo sviluppo dei tre componenti del TSV e la seconda l'analisi delle prestazioni dell'intero framework Magneti Marelli. In particolare, viene posta l'attenzione sul tempo impiegato dai moduli software per effettuare la scrittura dei messaggi nei file di log.
Il TSV è stato utilizzato dal gruppo di ricerca durante la dimostrazione della tecnologia V2X all'evento **Modena Automotive Smart Area** (MASA) tenutosi presso l'Autodromo di Modena, in Italia, nel Settembre 2018.

8 Risultati

8.1 Test Site Visualizer

I tre componenti del **Test Site Visualizer** sono il modulo software **Interpreter**, il **web server** ed il **web site**. Il primo riceve i messaggi BSM, contenenti le informazioni di base dei veicoli, e le invia al server, il secondo le inoltra ai device come smartphone e tablet ed elargisce i "tile" di mappa geografica OSM e, infine, il terzo permette la visualizzazione ed il tracciamento in tempo reale dei veicoli in movimento sul circuito. L'architettura software è mostrata in Figura 32.

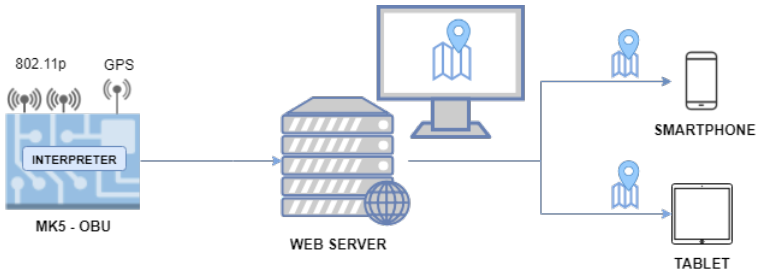

Figura 32: Architettura Test Site Visualizer

Il web site è stato progettato in modo che presenti un'interfaccia principale che contenga la mappa geografica, una checkbox posta sul lato destro utile per poter impostare il tipo di mappa, **stradale** o **satellitare**, e i *layer* da visualizzare sulla mappa, e da un menu contestuale posto sul lato sinistro, apribile a scorrimento per poter gestire alcune impostazioni di configurazione, utili per poter caricare nuovi *layer* o cambiare quelli esistenti, e un tab in cui è presente un tasto "Start" tramite il quale è possibile avviare la ricezione dei dati contenenti le coordinate geografiche dei veicoli, ottenendo la raffigurazione dei relativi *marker* sul circuito.

La mappa OSM, opportunamente prerenderizzata, comprende l'autodromo di Modena, il circuito ad esso adiacente e la sezione di Venaria Reale.

È stata realizzata, grazie all'utilizzo del protocollo **websocket**, una connessione a bassa latenza e senza spreco di banda. In questo modo, vengono evitate continue richieste del dato da parte del client al server, ottenendo molto meno overhead sul web server ed un vero real-time in quanto l'unico ritardo riscontrato è quello dovuto al trasferimento dei dati.

Inoltre, sono stati utilizzati dati cartografici, grazie all'ausilio di **OpenStreetMap**, e sono stati modificati e aggiornati, tramite l'editor **JOSM**, in modo da poter accedere ad un livello di zoom più elevato, assolvendo la necessità di visualizzare i veicoli ad un livello di dettaglio maggiore.

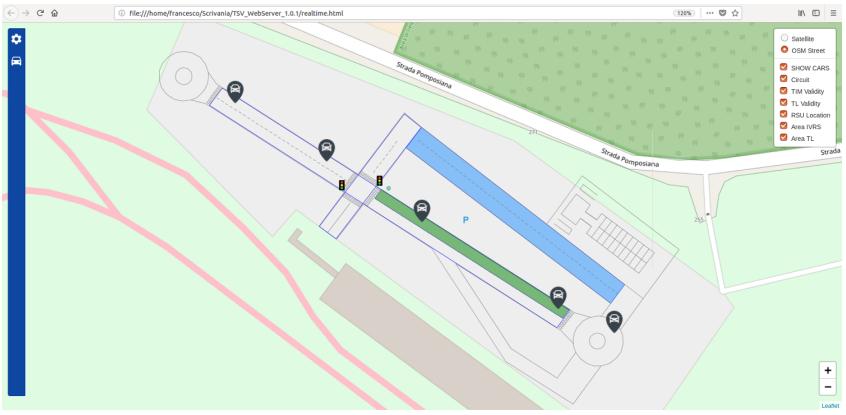

Figura 33: TSV - Risultato finale tracciamento veicoli

8.2 Test librerie di log

La seconda parte della tesi ha permesso di ottenere dei dati relativi alle prestazioni delle più **performanti** librerie di logging per il linguaggio C++. Questo lavoro è stato effettuato perché durante il test dell'intero framework MM si è

resa evidente la necessita di ottimizzare le prestazioni di ciascuno dei suoi moduli, date le capacità hardware a disposizione.

Ogni operazione di log, infatti, causa un ritardo, seppur minimo, che va ad inficiare la reattività di tutto il framework ed implementare una libreria di log più efficiente consente di rendere molto più veloce l'esecuzione di tutto il sistema, riducendo i ritardi, migliorando la velocità di risposta dei moduli e ottenendo più velocemente dati in tempo reale.

Ogni modulo del framework scrive i propri risultati su un file di log, il quale è univoco per ciascuno di essi, pertanto questo studio è servito per individuare quale libreria di sposasse meglio con le esigenze dei moduli che costituiscono il framework Magneti Marelli al fine di migliorare le performance complessive durante l'esecuzione.

La libreria utilizzata nel framework MM è la libreria **Boost**, in quanto possiede alcune caratteristiche che si sposano bene con le esigenze di cross-compilazione da ambiente x86 ad ARM del framework, come la riusabilità, la portabilità e l'essere cross-platform.

Offre, inoltre, buone prestazioni a runtime ed efficienza dello spazio rispetto alla leggibilità del codice di implementazione delle stesse, offrendo blocchi predefiniti che possono essere combinati più liberamente senza dettare o dominare la progettazione dell'applicazione.

È stato effettuato un test preliminare sulle prestazioni di questa libreria riscontrando dei ritardi che potevano essere ridotti. Sono state, dunque, testate altre cinque librerie di logging per C++ tutte open source, senza dipendenze esterne e scaricabili da repository GitHub, che sono: Spdlog, Easylog, Nanolog, Loguru e Miniasynclog.

Il log da effettuare è di tipo ***rotazionale*** in modo tale che quando viene raggiunta la dimensione massima del file di log, esso venga riscritto. Ciò permettere di non saturare la memoria dell'MK5 e di mantenere comunque le performance elevate. È stato, infine, posto un limite di dimensione configurabile da file di configurazione. Il modulo con il quale sono state testate è un modulo progettato appositamente per effettuare alcuni stress-test.

Sono stati creati diversi scenari per poter analizzare il comportamento delle librerie, studiando il loro comportamento e individuando quale risponda in maniera migliore alle esigenze dei moduli. Il modulo software deve:

- Generare un numero di thread, impostabile da file di configurazione, da eseguire contemporaneamente;
- Scrivere sul file di log un numero di righe impostabile anch'esso da file di configurazione.

Il modulo software di test offre la possibilità di testare i seguenti nove tipi di scenari:

- 1 thread e 10.000 righe di log;
- 1 thread e 100.000 righe di log;
- 1 thread e 1.000.000 di righe di log;
- 10 thread e 10.000 righe di log;
- 10 thread e 100.000 righe di log;
- 10 thread e 1.000.000 di righe di log;
- 100 thread e 10.000 righe di log;
- 100 thread e 100.000 righe di log;
- 100 thread e 1.000.000 di righe di log.

Il test è stato effettuato su una board, Step 3, con le seguenti caratteristiche:

- ***CPU***: Processore ARv7, 4 core
- ***RAM***: 1 GB
- ***Disco***: 512 MB

8.2.1 Tempi ottenuti

I risulati di ogni singola libreria sono stati utilizzati per determinare quale tra esse si sposasse meglio con le esigenze del Test Site Visualizer. I tempi sono espressi in secondi e negli scenari in cui vengono eseguiti 10 o 100 thread, è stata effettuata una media sui tempi di ciascun thread, dal momento che da ognuno di essi sono stati ottenuti tempi pressoché simili.

Di seguito, si riportano le tabelle che mostrano i risultati ottenuti durante la fase di test.

Libreria Boost

N. RIGHE DI LOG	N. THREAD		
	1	10	100
10.000	1.557 s	8.866 s	86.864 s
100.000	12.876 s	90.946 s	883.972 s
1.000.000	124.892 s	1055.360 s	8439.447 s

Tabella 4: Risultati libreria Boost

Libreria Spdlog

N. RIGHE DI LOG	N. THREAD		
	1	10	100
10.000	0.466 s	3.799 s	35.566 s
100.000	3.240 s	39.407 s	378.892 s
1.000.000	29.549 s	397.373 s	3809.084 s

Tabella 5: Risultati libreria Spdlog

Libreria Easylog

N. RIGHE DI LOG	N. THREAD		
	1	10	100
10.000	1.627 s	18.426 s	180.518 s
100.000	15.257 s	180.374 s	1921.440 s
1.000.000	156.529 s	1798.158 s	18932.889 s

Tabella 6: Risultati libreria Easylog

Libreria Nanolog

N. RIGHE DI LOG	N. THREAD		
	1	10	100
10.000	0.206 s	0.404 s	2.270 s
100.000	1.224 s	2.632 s	24.342 s
1.000.000	10.367 s	25.490 s	246.947 s

Tabella 7: Risultati libreria Nanolog

Libreria Loguru

N. RIGHE DI LOG	N. THREAD		
	1	10	100
10.000	0.862 s	5.712 s	54.959 s
100.000	5.885 s	55.539 s	539.428 s
1.000.000	57.844 s	554.488 s	5267.314 s

Tabella 8: Risultati libreria Loguru

Libreria Miniasynclog

N. RIGHE DI LOG	N. THREAD		
	1	10	100
10.000	1.980 s	11.483 s	139.383 s
100.000	12.874 s	112.450 s	1281.274 s
1.000.000	69.610 s	1032.889 s	10643.112 s

Tabella 9: Risultati libreria Miniasynclog

8.2.2 Confronto dei risultati

In tutti gli scenari di test, in cui vengono eseguiti rispettivamente un thread, 10 thread e 100 thread, la libreria che risulta più **performante** è la Nanolog, ottenendo dei risultati sorprendenti, che si distaccano notevolemente da quelli di tutte le altre.

Seguono le librerie Spdlog e Loguru che risultano, rispettivamente, la seconda e la terza libreria di logging più efficiente. La libreria Boost, come riscontrato nella fase preliminare del test, ottiene risultati che non si sposano adeguatamente con le esigenze del Test Site Visualizer, in quanto la necesiità è quella di ottenere un sistema real-time molto performante.

Infine, si notano le librerie Easylog e Miniasynclog per le quali si riscontrano tempistiche molto elevate, risultando inefficienti.

Nella Tabella 10 sono mostrate le performance delle librerie testate eseguendo un solo thread.

N. THREAD	Nanolog	Spdlog	Loguru	Boost	Easylog	Miniasynclog
10.000	0.206 s	0.466 s	0.862 s	1.557 s	1.627 s	1.980 s
100.000	1.224 s	3.240 s	5.885 s	12.876 s	15.257 s	12.874 s
1.000.000	10.367 s	29.549 s	57.844 s	124.892 s	156.529 s	69.610 s

Tabella 10: Comparazione librerie - Un thread

Di seguito, in Tabella 11, i risultati ottenuti testando le librerie eseguendo 10 thread.

N. THREAD	Nanolog	Spdlog	Loguru	Boost	Easylog	Miniasynclog
10.000	0.404 s	3.799 s	5.712 s	8.866 s	18.426 s	11.483 s
100.000	2.632 s	39.407 s	55.539 s	90.946 s	180.374 s	112.450 s
1.000.000	25.490 s	397.373 s	554.488 s	1055.360 s	1798.158 s	1032.889 s

Tabella 11: Comparazione librerie - 10 thread

Infine, vengono comparate le performance, mostrate nella Tabella 12, eseguendo 100 thread.

N. THREAD	Nanolog	Spdlog	Loguru	Boost	Miniasynclog	Easylog
10.000	2.270 s	35.566 s	54.959 s	86.864 s	139.383 s	180.518 s
100.000	24.342 s	378.892 s	539.428 s	883.972 s	1281.274 s	1921.440 s
1.000.000	246.947 s	3809.084 s	5267.314 s	8439.447 s	10643.112 s	18932.889 s

Tabella 12: Comparazione librerie - 100 thread

Un grafico riepilogativo delle prime tre librerie più performanti si può osservare in figura 33, in cui si mostrano le performance ottenute nella scenario più significativo, ossia quello che prevede l'esecuzione di 100 thread.

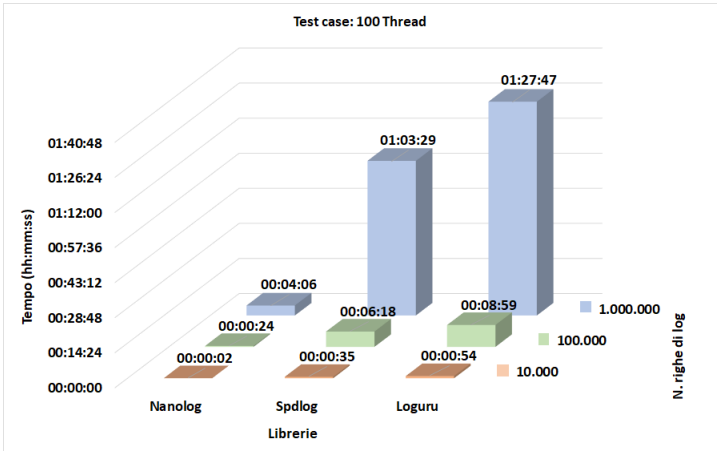

Figura 34: Riepilogo comparazione librerie

I risultati migliori dal punto di vista delle performance sono stati ottenuti dalla libreria **Nanolog** ed essa è stata compilata ed integrata all'interno del framework MM, ottimizzando il processo di log di tutti i moduli e garantendo fluidità durante la loro l'esecuzione.

Ciò ha migliorato le prestazioni di tutto il framework, rispondendo adeguatamente alle esigenze del **Test Site Visualizer** di ottenere un sistema **real-time** altamente performante, permettendo di ottenere un'esperienza dinamica e fluida sull'interfaccia del site web.

9 Sviluppi futuri

Questo lavoro di tesi prevede possibili sviluppi futuri su molti aspetti, poiché il Test Site Visualizer si sviluppa su più fronti. Innanzitutto, per quanto riguarda il lato backend le possibilità applicative dell'Interpreter sono molteplici.

La sua funzione di sniffer, infatti, non solo è utile per poter richiedere ed estrapolare dati importanti dalla comunicazione V2X, ma può essere utilizzata anche per testare il funzionamento di tutti i moduli che prendono parte all'esecuzione del Framework. Dai file di log generati dall'Interpreter, infatti, si possono evincere eventuali problematiche o guasti che potrebbero sorgere durante l'utilizzo del sistema, verificando l'ID univoco del veicolo che ha interrotto la trasmissione dei dati.

Si procederà ad analizzare possibili soluzioni che potrebbero riguardare la rete CAN o il modulo BS-BS. La funzione di sniffer dell'Interpreter può essere, inoltre, estesa per estrapolare dati dalla comunicazione V2X riguardanti gli **eventi** generati dai veicoli

Il web site è aperto a molteplici implementazioni. Si potranno applicare i filtri sui pacchetti ricevuti, in modo da visualizzare solo i dati di uno specifico veicolo. Si può estendere la sezione della configurazione, permettendo non solo di modificare o aggiungere nuovi *layer*, ma anche di poter aggiungere tipi diversi di oggetti sia statici che dinamici quali possono essere semafori, cartelli stradali, divieti e tutto quel che concerne la gestione del traffico stradale.

Si possono, infine, visualizzare su schermo da remoto, tramite "**Card-layer**", gli eventi che sono stati generati dai singoli veicoli, come ad esempio l'avviso di una frenata brusca o di un veicolo nelle vicinanze, rappresentando il quadro **HMI**, Human Machine Interface, del singolo veicolo sul quale si sta verificando lo specifico evento.

Riferimenti bibliografici

[1] MCKINSEY & COMPANY - BLOOMBERG NEW ENERGY FINANCE, “An integrated perspective on the future of mobility”, October 2016.

[2] M. Peden; Richard Scurfield; D. Sleet; D. Mohan; et al. "World report on road traffic injury prevention" (PDF). World Health Organization. Retrieved 2008-02-29.

[3] INTERNATIONAL ECONOMIC DEVELOPMENT COUNCIL (a cura di), “Creating the Clean Energy Economy - Analysis of the Electric Vehicle Industry”, Washington 2013.

[4] https://www.magnetimarelli.com/it/azienda

[5] An Overview of Inter-Vehicular CommunicationSystems, Protocols and Middleware, JOURNAL OF NETWORKS, VOL. 8, NO. 12, DECEMBER 2013

[6] http://www.cplusplus.com/info/description/

[7] REQ-BS-BS-006, Standard, Validate packets decoded consistency according ASN.1 rules, 3.0.8

[8] http://buildnewgames.com/websockets/, “WebSockets: A Guide” di Jack Lawson

[9] https://mapnik.org/

[10] https://httpd.apache.org/

[11] https://github.com/openstreetmap/mod_tile.git

[12] https://github.com/gravitystorm/openstreetmap-carto.git

[13] https://github.com/openstreetmap/osm2pgsql

[14] https://www.maxcdn.com/one/visual-glossary/websocket/

[15] ETSI TS 102 894-2 V1.1.1: “Intelligent Transport Systems (ITS); Users and applications requirements; Part 2: Applications and facilities layer common data dictionary”

[16] Journal of Wireless Networking and Communications 2013, 3(3): 29-38

[17] ETSI EN 302 637-2 V1.3.1: “ Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 2: Specification of Cooperative Awareness Basic Service “

[18] IEEE Vehicular Technology Society “ Standard for Wireless Access in Vehicular Environments (WAVE)

[19] Overview of ARIB Standards (STD-T109), https://www.arib.or.jp/english/std_tr/telecommunications/desc/std-t109.html

[20] IEEE Vehicular Technology Society " Standard for Wireless Access in Vehicular Environments (WAVE) - Security Services for Applications and Management Messages", 2006

[21] IEEE Vehicular Technology Society " Standard for Wireless Access in Vehicular Environments (WAVE) - Networking Services" , 2007

[22] D.Jiang, L.Delgrossi, " IEEE 802.11p: Towards an International Standard for Wireless Access in Vehicular Environments"

[23] http://www.postgresql.org/

[24] http://www.omniauto.it/magazine/27675/magneti-marelli-storia-retrospettiva/

[25] Anna Maria Vegni , Mauro Biagi, Roberto Cusani " Smart Vehicles, Technologies and Main Applications in Vehicular Ad hoc Networks", 2013

[26] SAE J2735 Dedicated Short Range Communications (DSRC) Message Set Dictionary, SAE, DSRC Committee, Nov. 2009.

[28] https://www.ieee.org/

[29] https://www.cisco.com/c/en/us/products/ios-nx-os-software/mobile-ad-hoc-networking/index.html

[30] Claudia Campolo, Antonella Molinaro. "Multichannel communications in vehicular ad hoc networks: A survey". IEEE Communication Magazine, 2013

[31] ETSI EN 302 637-2 V1.3.1: " Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 2: Specification of Cooperative Awareness Basic Service "

[32] ETSI EN 302 637-3 V1.2.1: " Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 3: Specifications of Decentralized Environmental Notification Basic Service "

[33] http://www.obj-sys.com/asn1tutorial/node1.html

[34] https://github.com/7890/osmconvert