ALMA MATER STUDIORUM
UNIVERSITÀ DEGLI STUDI DI BOLOGNA

Scuola di Ingegneria e Architettura
Corso di Laurea Magistrale in Ingegneria Informatica

SIMULAZIONI DI NAMED DATA NETWORKING PER RETI VEICOLARI

Tesi nel corso di: Reti di Telecomunicazioni LM

Relatore:
Prof. FRANCO CALLEGATI
Co-relatori:
PROF. WALTER CERRONI
PROF. GIOVANNI PAU

Tesi di Laurea di:
CHIARA CONTOLI

ANNO ACCADEMICO 2012–2013
SESSIONE II

PAROLE CHIAVE

Named Data Networking

Reti

Mobilità

Veicolare

Simulazioni

“Il gran giorno capita sempre prima o poi... ”
Un mercoledì da leoni

Indice

Introduzione **ix**

1 Named Content **1**

1.1 Introduzione 1
1.2 Il modello CCN 2
1.2.1 Identificazione dei dati: sequencing 6
1.2.2 Routing 9
1.3 Named Data Networking su reti veicolari 12
1.3.1 Reti e servizi per il veicolare 13
1.3.2 Applicabilità di NDN alle reti veicolari 15
1.3.3 Sfide nell'applicare NDN a reti veicolari 18
1.3.4 Un esempio di applicazione 20

2 Strumenti per la mobilità **25**

2.1 Linee guida per la costruzione di una mobilità veicolare . . . 25
2.1.1 Un framework per modelli di mobilità veicolare . . . 26
2.1.2 Classi di modelli di mobilità 28
2.1.3 Synthetic Models 28
2.1.4 Survey-based Models 29
2.1.5 Trace-based Models 29
2.1.6 Traffic Simulator-based Models 30
2.1.7 Validazione 30
2.2 Modelli di mobilità e simulatori di rete 31
2.3 Criteri di scelta derivanti dai requisiti dell'applicazione . . . 32
2.3.1 Simulatori di rete 36
2.4 Modelli di mobilità 36
2.4.1 Isoleted Vehicular Model 36

2.4.2 Embedded Vehicular Mobility Model 37
2.4.3 The Federated Mobility Models 38
2.4.4 Alcune considerazioni 38

3 Strumenti utilizzati 39
3.1 TIGER 39
3.2 Simulation of Urban MObility (SUMO) 40
3.3 CORNER 41
3.4 VERGILIUS 46
3.5 Lettura delle mappe tramite TIGER 47
3.6 Generatore di scenario 47
3.7 Simulatore di rete: NS-3 + ndnSIM 49
3.8 Mettere insieme gli strumenti, dalla configurazione alla mobilità 52

4 Scenari di Simulazione 57
4.1 Casi d'uso 57
4.2 Analisi dei dati e risultati 62
4.2.1 Singolo Produttore Singolo Consumatore - Scenario 1 63
4.2.2 Singolo Produttore Singolo Consumatore - Scenario 2 64
4.2.3 Singolo Produttore Singolo Consumatore - Scenario 3 76
4.2.4 Singolo Produttore Multi Consumatore - Scenario 1 . 82
4.2.5 Singolo Produttore Multi Consumatore - Scenario 2 . 88
4.2.6 Multi Produttore Singolo Consumatore - Scenario 1 . 88
4.2.7 Multi Produttore Singolo Consumatore - Scenario 2 . 101
4.2.8 Multi Produttore Singolo Consumatore - Scenario 3 . 107
4.2.9 Multi Produttore Multi Consumatore - Scenario 1 . . 113
4.2.10 Multi Produttore Multi Consumatore - Scenario 2 . . 113
4.3 Considerazioni 126

5 Conclusioni 129

A Funzioni di utilità e script 131
A.1 Esempio di file di configurazione, modalità 1 131
A.2 Esempio di file di configurazione, modalità 7 132
A.3 Funzionalità 132
A.4 Configurazione e codice simulazione 145
A.5 Analisi dei dati 162

Introduzione

Le reti veicolari, anche dette VANET, sono da tempo oggetto di studio. Durante il periodo di ricerca svolto presso l'Università della California Los Angeles (UCLA) è stato possibile studiare i protocolli adatti allo scambio di contenuti fra i veicoli secondo il paradigma del Named Data Networking (NDN). Il Named Data Networking rappresenta un nuovo modello di comunicazione per il reperimento dei contenuti all'interno della rete. Nelle VANET ogni veicolo è potenzialmente un fornitore di contenuti, oltre che un richiedente.
L'infrastruttura di riferimento posta all'interno del campus universitario permette il reperimento di dati necessario allo studio del problema, non solo da un punto di vista pratico ma anche da un punto di vista teorico. Infatti, data la tipologia dei test e le difficoltà intrinseche che essi comportano, l'attività di simulazione svolge un ruolo importante per lo sviluppo e lo studio del protocollo all'interno delle reti veicolari.
L'attività di ricerca svolta si articola nei seguenti aspetti:

- introduzione al nuovo paradigma di comunicazione: principi del Named Data Networking, funzionamento di NDN, reti veicolari, applicabilità di NDN alle VANET;

- modelli di mobilità per le reti veicolari: linee guida per la costruzione di un modello di mobilità, situazione attuale dei modelli disponibili, simulatori di rete, strumenti utilizzati e il loro funzionamento;

- attività di simulazione: pianificazione e implementazione di diverse tipologie di scenari di reti veicolari;

- analisi dei dati raccolti dalla fase precedente: vengono elaborati i dati raccolti e si cerca di catturarne gli aspetti più significativi.

L'obiettivo è quello di condurre uno studio di fattibilità sull'applicazione di NDN alle reti mobili, in particolare alle reti veicolari in ambito urbano. Al momento in cui è iniziata la collaborazione con il gruppo di ricerca del Network Research Lab di UCLA, era da poco stata rilasciata la prima versione di NDN contenente l'estensione pensata per il veicolare, quindi non erano presenti in letteratura studi condotti per questo tipo di scenari. Lo scopo è quello di estrarre informazioni e ricavarne significative indicazioni sulle prestazioni del sistema.

Capitolo 1

Named Content

1.1 Introduzione

Oggi giorno la rete vede il suo maggiore utilizzo nella distribuzione/richiesta di contenuti e servizi. L'accesso di contenuti e servizi richiede un mapping tra *cosa* interessa agli utenti e *dove* risiede nella rete il desiderata. Con l'avvento di Internet negli anni 1960-'70 il networking aveva l'obiettivo di risolvere il problema della condivisione delle risorse. Questo portò ad un modello di comunicazione in cui il dialogo è tra due macchine, una che richiede l'utilizzo di una risorsa, e l'altra che fornisce l'accesso ad essa.
Quando gli utenti utilizzano Internet sono interessati a *cosa* questo contiene e non a *dove* il contenuto risiede. Al giorno d'oggi però, la comunicazione è ancora in termini di *dove* risiede l'informazione. La comunicazione host-to-host è un'*astrazione* di rete scelta per risolvere i problemi e le esigenze emerse negli anni '60-'70. Una soluzione che prevede un modello che sposta l'attenzione sul *cosa* anziché sul *dove*, fornisce una migliore astrazione per i problemi di comunicazione odierni. Viene quindi presentata una architettura di comunicazione basata sul *named data*, chiamata *Content-Centric Networking* (*CCN*). Questo modello non ha alcuna nozione di host al suo livello più basso (non c'è concetto di "indirizzo" del pacchetto). Tuttavia, il modello mantiene le decisioni progettuali che hanno reso celebre il TCP/IP e trae molta ispirazione da IP. Nel seguito verrà presentata l'architettura e le operazioni di CCN, che vedrà il suo utilizzo non solo nell'ambito delle reti fisse, ma anche nell'ambito delle reti mobili, in particolare nell'ambito delle reti veicolari. In un mondo dove sempre più dispositivi sono connessi alla

rete, con opportune varianti, si vedrà infatti come questo nuovo modello di comunicazione sia particolarmente adatto negli scenari di mobilità urbana.

1.2 Il modello CCN

Facendo riferimento a quanto descritto in [1], il modello CCN prevede una comunicazione di tipo *consumer driven*, ovvero una comunicazione guidata dal Consumatore. Al momento distinguiamo due tipologie di nodi:

- *Consumer*: è il nodo che richiede i contenuti;
- *Producer*: è il nodo che potenzialmente possiede i contenuti e risponde ai Consumers;

Oltre a distinguere due tipi di nodi, si distinguono due tipologie di pacchetto:

- *Interest*:inviato dal Consumer al Producer, esprime la richiesta del contenuto;
- *Data* o (*Content*): inviato dal Producer al/i Consumer(s), rappresenta il contenuto richiesto;

In figura 1.1 è possibile vedere il formato dei pacchetti. La richiesta di conte-

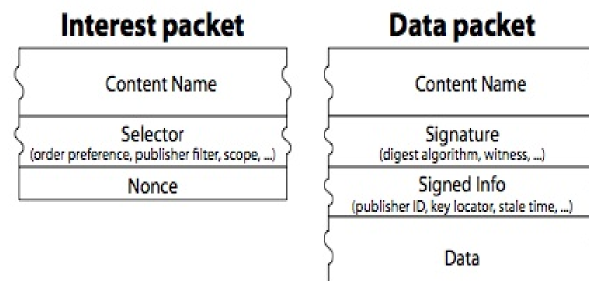


Figura 1.1: Tipologia di pacchetto CCN

nuto viene inviata in broadcast attraverso tutta la connettività disponibile; qualunque nodo che percepisce il pacchetto ed è in possesso del contenuto richiesto, risponde con il Content. Il pacchetto di tipo Data viene trasmesso solo a fronte di una richiesta di Interest da parte del Consumer, e si

dice in tal caso che il Data “consuma” l’Interest. Ad ogni Interest quindi, corrisponde un Data, mantenendo così un bilanciamento del flusso. Dal momento che sia il Data che l’Interest identificano il contenuto scambiato dal nome, più nodi interessati al medesimo contenuto possono condividere la trasmissione su un mezzo broadcast.
Con riferimento a figura 1.1 il Data “soddisfa” un Interest se il ContentName nell’Interest è un prefisso nel ContentName del Data. I nomi hanno una struttura di tipo gerarchico, quindi il matching del prefisso è equivalente a dire che il pacchetto Data è nel sotto albero del nome specificato dal pacchetto Interest. Quello che può accadere è che alcuni Interest possono essere ricevuti per contenuti non ancora disponibili, consentendo però così di poter generare il contenuto “al volo” in risposta alla richiesta. In questo modo CCN consente di avere sia contenuti memorizzati in cache staticamente, che contenuti generati dinamicamente. Il “look-up” del contenuto è quindi simile al look-up effettuato da IP. Quando un pacchetto arriva su una *faccia*, viene effettuato un longest-match look-up sul nome e poi viene eseguita un’azione sulla base del risultato del look-up. Si utilizza il termine faccia al posto di interfaccia perché i pacchetti non sono inoltrati solo sulle interfacce dell’hardware di rete, ma anche scambiate direttamente tra processi applicativi all’interno della medesima macchina. In figura 1.2 è visibile

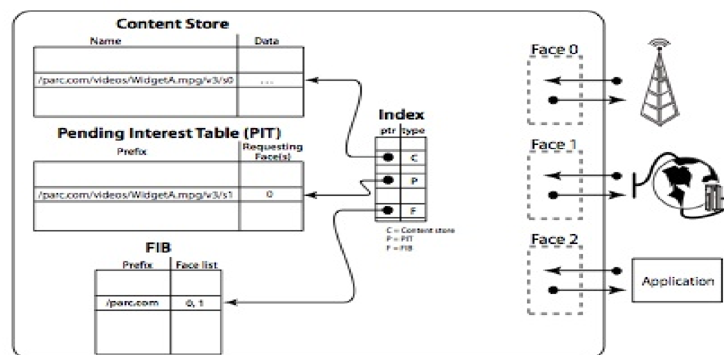


Figura 1.2: Modello di forwarding in CCN

il modello di funzionamento del forwarding. Vi sono tre strutture principali: la FIB (Forwarding Information Base), Content Store (o buffer di memoria, CS) e la Pending Interest Table (PIT).

La **FIB** è utilizzata per fare il forwarding dei pacchetti di tipo Interest verso potenziali sorgenti di Data in grado di fare matching. E' molto simile alla FIB dell'IP, anche se a differenza sua permette l'inoltro presso più interfacce, e non verso una solo interfaccia. Più sorgenti possono quindi essere interrogate parallelamente.

Il **Content Store** è un buffer di memoria simile a quello che si trova nei router IP, ma possiede una politica di gestione diversa. In CCN, ogni pacchetto è idempotente, auto-identificato e auto-autenticato, per cui lo stesso pacchetto potenzialmente è utile a più Consumers.Per massimizzare lo sharing quindi, e minimizzare la richiesta della banda in upstream e il delay in downstream, la politica adottata da CCN è ricordarsi i pacchetti di tipo Data il più a lungo possibile (politica LRU o LFU). In IP invece, siccome ogni pacchetto appartiene ad un solo collegamento punto-punto, quello che accade è che, una volta fatto il forwarding, il posto nel buffer viene immediatamente riciclato (politica di tipo MRU).

La **PIT** tiene traccia delle richieste di Interest inoltrate upstream verso le sorgenti di contenuti, in modo che i pacchetti Data possano essere inoltrati all'indietro verso chi li ha richiesti. In CCN viene fatto il routing solo dei pacchetti di tipo Interest e, siccome vengono propagati upstream, lasciano una sorta di traccia per consentire ai pacchetti Data di poter essere mandati indietro all'origine della richiesta. Ogni entry nella PIT costituisce una traccia. Non appena il Data giunge al nodo presso cui è presente la entry, questa viene cancellata non appena il forwarding è avvenuto (il Data consuma l'Interest). Entry della PIT tali per cui un matching per un certo Interest non viene mai trovato, vengono eventualmente mandate in time out. E' compito del Consumer esprimere nuovamente l'Interest se vuole ottenere il Data.

Di seguito viene spiegato come funziona il processamento dell'Interest: quando un pacchetto Interest arriva su una faccia, viene effettuato un longest-match look up all'interno del proprio Content Store. L'ordine di preferenza per il look - up è il seguente: un matching sul ContetStore è preferito rispetto ad un matching sulla PIT, il quale sarà preferito rispetto ad un matching sulla FIB. In questo modo, se nel Content Store è già presente il Data richiesto, questo viene direttamente inoltrato verso la faccia presso la quale

è arrivata e l'Interest sarà scartato in quanto considerato soddisfatto. In caso contrario, viene interrogata la PIT. Se all'interno della PIT è presente una entry che corrisponde al quel contenuto, la entry viene aggiornata aggiungendo la nuova faccia che ha richiesto l'Interest, e l'Interest viene scartato (questo è il caso in cui una richiesta di Interest è già stata inoltrata upstream, quindi tutto quello che è necessario fare è che quando il pacchetto Data arriva, uno sua copia venga mandata anche presso questa nuova faccia). Se non vi e alcun matching nella PIT, allora viene consultata la FIB, e ciò significa che l'Interest deve essere inoltrato upstream. Se esiste una entry per il contenuto richiesto, l'Interest viene inoltrato presso tutte le facce che potenzialmente soddisfano la richiesta, e una entry nella PIT viene aggiunta. Se nella FIB non viene trovato nessun matching, l'Interest viene scartato (il nodo non possiede il contenuto e non sa neanche come trovarlo).
Per quanto riguarda il processamento dei pacchetti di tipo Data, non sono soggetti a routing, semplicemente seguono il percorso all'indietro creatosi con le entry della PIT per giungere fino all'origine. All'arrivo del pacchetto, un longest match look-up è fatto sul Content Store; se il matching viene trovato, questo significa che il Data è duplicato, quindi il pacchetto viene scartato. Se non viene trovato alcun matching nella PIT, il Data è definito *unsolicited*, e viene scartato. Un Unsolicited Data può essere generato da comportamenti malevoli, Data in arrivo da più sorgenti o da più percorsi da una singola sorgente. Nel primo caso, quando viene ricevuta la prima copia del Data, l'Interest viene consumato, perciò la entry nella PIT viene rimossa (quindi pacchetti duplicati non trovano una entry che fa match nella PIT). Un matching nella PIT invece, significa che il Data è stato sollecitato da uno (o più) Interest. In questo caso, il Data viene messo nel Content Store e vengono poi soddisfatte tutte le richieste mandando copia del Data presso le opportune facce.
La natura multi-point per il reperimento dei contenuti consente di avere una certa flessibilità in ambienti altamente dinamici. Ogni nodo con accesso a più reti può servire da Content router tra di esse.Grazie alla cache, un nodo può fare da mediatore di comunicazione tra aree disconnesse. Il modello Interest/Data funziona anche quando la connettività è locale, quindi in collegamenti wireless ad-hoc.

1.2.1 Identificazione dei dati: sequencing

Dal momento che i Consumer richiedono porzioni individuali da una vasta collezione di dati, e molti destinatari potrebbero condividere lo stesso Data, è necessario avere un metodo sofisticato per identificare ciascun dato. La localizzazione e la condivisione sono agevolati dall'utilizzo di nomi gerarchici, minimamente significativi per l'essere umano. Un nome è composto da un numero di *componenti*. Ogni componente è composto da un numero arbitrario di ottetti (valori binari di lunghezza variabile che non hanno alcun significato dal punto di vista del trasporto CCN). I nomi devono essere significativi per un qualche livello superiore nello stack protocollare al fine di essere utili, ma il trasporto non impone nessun tipo di restrizione, ad eccezione della struttura del componente. Codifica binaria, interi e altri valori complessi potrebbero essere usati direttamente senza conversione a testo per poi essere trasmessi. I nomi dei componenti potrebbero anche essere criptati per la privacy. Per convenienza di notazione i nomi sono presentati come URIs, e il carattere è utilizzato come separatore dei componenti, come indicato in figura 1.3 . In particolare, la figura mostra la convenzione utilizzata attualmente a livello applicativo, utilizzato per catturare l'evoluzione temporale del contenuto (un marker per la versione, *_v* codificato come *FD*, seguito da un valore intero che rappresenta il numero di versione) e la sua segmentazione (un marker di segmentazione (*_s* codificato come *00*, seguito da un valore intero che potrebbe essere un blocco o un numero di byte o il numero di frame del primo frame di video contenuto nel pacchetto). Il componente finale del nome di ogni pacchetto Data implicitamente include un digest SHA256 del pacchetto. Il digest del pacchetto non è trasmesso dal momento che è derivabile. Esiste così un Interest, o un link, che può essere interpretato in maniera non ambigua e indica esattamente un pezzo del contenuto. Un Interest può specificare esattamente quale contenuto è richiesto, ma nella maggiore parte dei casi, il nome del prossimo dato che viene richiesto non è noto a priori, così il Consumer lo specifica *relativo* a qualcosa il cui nome è noto. Questo è possibile perché l'albero del nome CCN può essere totalmente ordinato, così, relazioni come *successivo* e *precedente* possono essere interpretati in maniera non ambigua dal trasporto CCN senza alcuna conoscieza relativa alla semantica. Un esempio è visibile in figura 1.4 . La figura mostra una porzione del nome associato a ciò che si vede in figura 1.3 . Una applicazione che vuole visualizzare la versio-

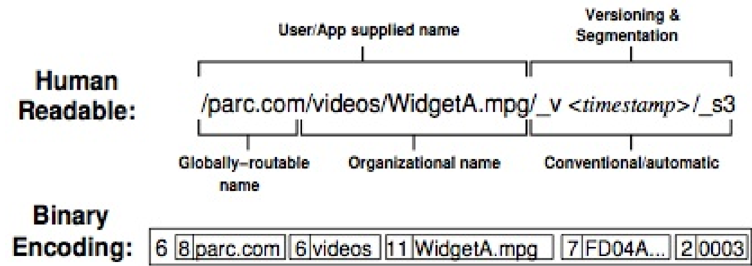


Figura 1.3: Notazione dei nomi

ne più recente del video esprimerebbe l'interest nella seguente notazione: "/parc.com/videos/WidegetA.mpg *RightmostChiled*" che risulta il percorso evidenziato in figura 1.4 . Una volta che è stato ottenuto il dato richiesto, il segmento successivo potrebbe essere ottenuto inviando un Interest contenente il suo nome e specificando *LeftmostRightSibling* come annotazione, oppure semplicemente computando la porzione di nome contenente *_s1*, dal momento che le regole di segmentazione sono note (e determinate) dall'applicazione. La convenzione dei nomi per le porzioni di dato contenuti in una collezione possono essere progettati per trarre vantaggio dal modo in cui questi vengono recuperati dal pacchetto di tipo Interest e le applicazioni possono venire a conoscenza di dati disponibili attraverso la navigazione dell'albero. CCN parla di contenuti, non di nodi, quindi non ha necessità di fare un binding tra una identità di livello tre (l'indirizzo IP) e una di livello due (l'indirizzo MAC). Anche quando la connettività cambia rapidamente, CCN può sempre scambiare dati appena è fisicamente possibile farlo.
CCN modella connettività multiple attraverso la lista delle facce che costituiscono le entries della FIB. L'intento era quello di avere per ogni entry della FIB una sorta macchina astratta specializzata nelle scelte di forwarding, cioè come inoltrare gli Interests. Questo viene chiamato anche *Strategy Layer* e il programma contenuto nella FIB è chiamato *strategia* per ottenere il Content associato ad un certo prefisso contenuto nella FIB. La strategia di default attuale è quello di mandare l'Interest su tutte le facce attive; se non si ha risposta, si provano tutte le altre interfacce in sequenza. In questo modo, i Content disponibili nell'ambiente locale (come ad esempio sul telefono, sul laptop, sul computer di un docente ad una conferenza) saranno

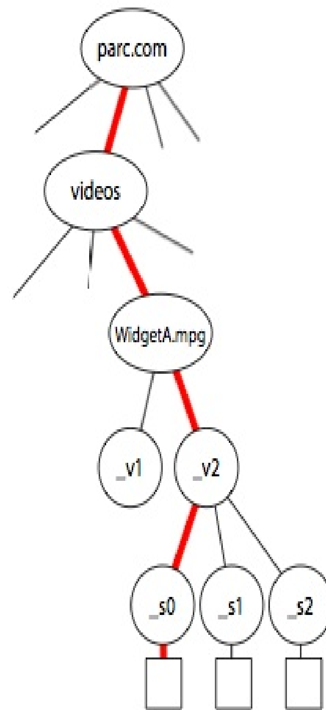


Figura 1.4: Rappresentazione ad albero dei nomi

ottenuti direttamente , e solo i Content che non sono disponibili a livello locale utilizzeranno il routing IP classico. Come avviene il routing classico per reperire Content in ambiente non locale sarà discusso in un successivo paragrafo.
Le altre facce contenute in un prefisso della FIB sono imparate in diversi modi. Le sorgenti di informazioni, come i repository visibili in figura 1.5, si predispongono per ricevere gli Interests per i prefissi che servono (cioè che sono in grado di soddisfare) effettuando una operazione di *Register* presso il core CCN locale. Questo fa sì che si vengano a creare delle entries locali all'interno della FIB per i prefissi registrati. I prefissi registrati hanno dei flag che indicano se il prefisso deve essere annunciato all'esterno dell'ambiente (macchina) locale. In questo modo l'agente CCN legge i prefissi registrati sul nodo locale e annuncia all'esterno solo quelli il cui flag rispecchia quel tipo di politica. L'annuncio può avvenire via CCN (l'agente serve l'Iterest in */local/CCN/registrations*, via standard IP Service Location protocols, oppure via CCN o IP routing (come sarà illustrato nel paragrafo successivo).

1.2.2 Routing

Ogni schema di routing che lavora bene per IP dovrebbe lavorare bene anche per CCN, perché il modello di forwarding utilizzato da CCN è un rigoroso superset del modello adottato da IP ma con con meno restrizioni (nessuna restrizione sul multi-sorgente, multi-destinazione per evitare il loop). Inoltre, possiedono anche la stessa semantica attinente al routing (aggregazione gerarchica dei nomi e longest-match lookup). Per illustrare come CCN è mappato sullo schema di routing , viene illustrato come CCN può essere instradato attraverso l'immutato Internet Link-State IGPs (o OSPF). Si faccia riferimento a figura 1.5 . I protocolli di routing Intra-dominio forniscono ai nodi un insieme di strumenti per la scoperta a la descrizione della loro connettività ("adiacenze"), e per descrivere le risorse connesse direttamente ("annuncio dei prefissi"). Queste due funzioni sono ortogonali: una descrive i collegamenti nel grafo, mentre l'altro descrive cosa è disponibile in un particolare nodo del grafo.
I prefissi CCN sono molto differenti dai prefissi IP, quindi la domanda principale è: è possibile esprimere i prefissi CCN in un protocollo di routing? Sia IS-IS che OSPF consentono di descrivere risorse direttamente connesse tramite il TLV (*Type Label Value*, che si rivela essere adatto a distribuire i

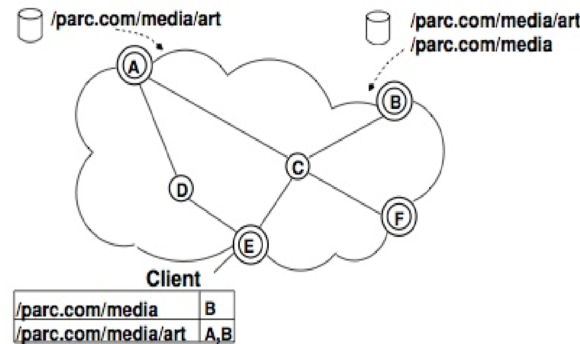


Figura 1.5: Repository e routing verso sorgenti di contenuti

prefissi CCN. La specifica dice che tipi non riconosciuti dovrebbero essere ignorati, che significa che i Content router, i quali implementano il modello completo del forwarding CCN , possono essere attaccati ad un IS-IS o OSPF esistenti, senza modificare la rete o i suoi routers. I content router apprendono la topologia della rete fisica e annunciano la loro posizione nella rete attraverso le adiacenze che possono essere stabilite tramite il protocollo, e poi distribuisce i suoi prefissi CCN in flooding tramite un TLV CCN.
Ad esempio, la figura 1.5 mostra un dominio IGP con alcuni routers IP (quelli a singolo cerchio) e alcuni routers IP con stack CCN. Il repository di contenuti multimediali vicino ad A annuncia (tramite un broadcast CCN in un namespace della rete locale) che lui può servire Interest i cui prefissi fanno matching con */parc.com/media/art* . Un'applicazione routing in A sente questo annuncio (dal momento che ha espresso Interest nel namespace dove l'annuncio è stato fatto), installa una CCN FIB entry per il prefisso, il quale punta alla faccia dalla quale ha sentito l'annuncio, e impacchetta l'annuncio in un IGP LSA che è mandato in flooding a tutti i nodi. Quando l'applicazione di routing su E , ad esempio, processa il LSA, crea una faccia CCN verso A, poi aggiunge l'entry del prefisso per "/parc.com/media/art" attraverso la faccia locale verso quella contenuta nella FIB. Quando un altro repository adiacente a B annuncia */parc.com/media* e */parc.com/media/art*, B effettua il flooding di un LSA per questi due prefissi il cui risultato è che la CCN FIB di E è quella presente in figura 1.5. Un Interest per */parc.com/media/art/impressionist-history.mp4* espresso da un client adia-

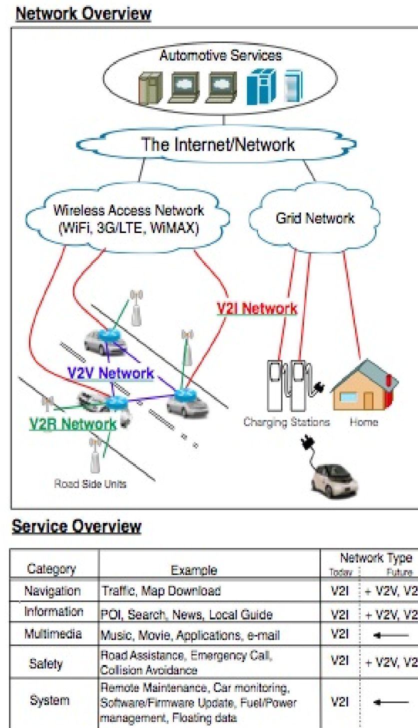
cente ad E sarà inoltrato sia verso A che verso B, i quali lo inoltreranno verso i loro repository adiacenti.
CCN costruisce dinamicamente topologie che sono vicine all'ottimo sia per l'uso della banda che per il ritardo sperimentato (ad esempio, i Data vanno solo dove ci sono gli Interest, lungo il percorso minimo e solo al più una copia di ciascun pezzo di Data viene inviata sopra ogni link). Questa topologia di consegna è chiaramente non ottima dal momento che un client adiacente ad F interessato allo stesso film implicherebbe l'invio di una seconda copia del contenuto sul link A-C oppure sul link B-C . Questo accade perché C non è un content router, per cui non effettua il caching. Quando il router C avrà un upgrade al software CCN, E ed F inoltreranno gli Interest attraverso C e la distribuzione sarà ottima.
C'è un comportamento differente tra IP e CCN quando ci sono annunci multipli dello stesso prefisso. In IP ogni nodo inoltrerà tutto il traffico che fa matching con un certo prefisso solo ad uno di coloro che li annuncia. In CCN ogni nodo invierà tutte le richieste che fanno matching a tutti coloro che annunciano quel prefisso. Questo arriva da una differenza semantica: un prefisso IP annunciato da un router IGP dice: tutti gli host con questo prefisso possono essere raggiunti attraverso me". Un annuncio equivalente fatto da un router CCN dice: "alcuni dei contenuti con questo prefisso possono essere ricevuti attraverso me" .
Per quanto riguarda il Routing Inter-dominio, avviene in maniera analoga a quanto visto per l'Intra-domain routing, solo sfruttando il protocollo BGP. Il BGP contiene l'equivalente dell'IGP TLV, meccanismo che consente ai domini di annunciare i loro prefissi.
CCN è quindi una architettura di rete costruita sui principi dell'IP, ma che utilizza il nome dei contenuti in luogo degli identificatori degli host. CCN è progettato per rimpiazzare IP, ma può essere incrementalmente sviluppato come un overlay, rendendo i suoi vantaggi funzionali disponibili alle applicazioni senza richiederne un'adozione universale. E' stato implementato un prototipo di stack CCN, dimostrandone la sua utilità sia per la distribuzione dei contenuti che nei protocolli point-to-point.

1.3 Named Data Networking su reti veicolari

Oggi giorno l'avanzamento delle tecnologie ha coinvolto sempre di più il mondo delle autovetture, migliorando e semplificando la vita di tanti automobilisti. L'*Intelligent Transportation System* (*ITS*), e i servizi telematici rappresentano la nuova idea di futuro dei servizi per le autovetture. Con questi servizi la guida diventa più sicura, più confortevole e una valida alleata nel contrastare i problemi dovuti alla cresciuta del traffico. I servizi correnti e quelli futuri impongono alle autovetture di essere connessi ad Internet in maniera permanente. Mentre la tradizionale comunicazione veicolo-infrastruttura continuerà ad essere la medesima in futuro per alcuni dei servizi, una connettività ad-hoc (come veicolo-veicolo (V2V) e veicolo-road side unit (V2R)) dovranno essere utilizzate totalmente per raggiungere una più sicura e confortevole esperienza di guida. Dopo diversi tentativi, queste tecnologie sono state sviluppate e tutt'ora lo studio e lo sviluppo sono ancora in corso. Fornitori e rivenditori hanno standardizzato diverse tecnologie, tra cui *Dedicated Short Range Communications* (DSRC, IEEE 802.11p) e IEEE 1609 (WAVE: *Wirelwss Access in Vehicular Environment*). Tuttavia, l'adozione di nuovi schemi di comunicazione fanno sorgere nuove sfide quando si considera l'integrazione dei servizi e la convergenza dei protocolli di rete su queste nuove tecnologie. Applicare il modello point-to-point a reti veicolari porta a molteplici problemi. Oggi non si ha una singola architettura di rete in grado di supportare tutti i servizi che "viaggiano" sopra tutte le diverse tecnologie di comunicazione.
Cercando un nuovo schema di comunicazione in grado di aiutare ad avere questa convergenza di più reti per il veicolare, si pensa che il Named Data Networking (anche detto CCN), come nuova proposta di architettura per il futuro Internet, abbia le potenzialità giuste per risolvere le sfide. Di seguito verrà illustrato come questo modello, che si propone di ottenere dati tramite nomi, si adatta bene al supporto dei servizi e alle reti veicolari, e si evidenziano le principali aree di ricerca che devono essere studiate per costruire con successo una rete veicolare basata su NDN.

1.3.1 Reti e servizi per il veicolare

Facendo riferimento a [2], in questo paragrafo verranno introdotti i servizi correnti e futuri per il veicolare, insieme ai rispettivi requisiti di rete, poi si discuteranno le ragioni per le quali l'architettura corrente non può essere estesa per gli scenari futuri immaginati. In figura 1.6 sono mostrate mol-



Category	Example	Network Type Today	Network Type Future
Navigation	Traffic, Map Download	V2I	+ V2V, V2R
Information	POI, Search, News, Local Guide	V2I	+ V2V, V2R
Multimedia	Music, Movie, Applications, e-mail	V2I	←
Safety	Road Assistance, Emergency Call, Collision Avoidance	V2I	+ V2V, V2R
System	Remote Maintenance, Car monitoring, Software/Firmware Update, Fuel/Power management, Floating data	V2I	←

Figura 1.6: Panoramica di servizi e reti veicolari

teplici reti e servizi veicolari; alcuni di essi sono già stati sviluppati. La navigazione di base, i servizi di multimedia e informazioni sono già integrati nell'unità veicolare. Si affidano alla comunicazione veicolo-infrastruttura (V2I) sopra la connettività della telefonia. Alcuni servizi sono stati anche implementati per gli smartphone, che il conducente può portare con sé. Entrambe le implementazioni utilizzano la tecnologia IP esistente. Il recente sviluppo di veicoli elettrici ha introdotto una nuova connettività di tipo V2I attraverso prese di rete intelligenti per la gestione dell'energia. Ci si aspetta

che i servizi di sicurezza come i freni di emergenza, l'anti-collisione (due dei più importanti servizi) faranno uso dell'emergente standard 802.11p . In più, i servizi di informazione e navigazione possono essere potenziati consentendo ai veicoli di comunicare direttamente tra di loro, o attraverso le road side unit (RSU: piccole stazioni statiche distribuite lungo le strade). Generalmente, le reti V2I tendono ad essere unicast per natura, mentre le V2V e le V2R sono per natura broadcast.
Dal trend corrente ci si aspetta che nel non lontano futuro ci saranno milioni e milioni di macchine online o connesse tra di loro. In figura 1.6 sono stati individuati alcuni servizi. Dal momento che il V2V e il V2R devono ancora essere messi sul mercato, tutti i servizi disponibili al giorno d'oggi sono utilizzati usando IP e viaggiano su V2I. Tuttavia, 3G ed LTE consentono ai veicoli di comunicare attraverso l'infrastruttura, ponendo problemi di elevata quantità di traffico su di esse e costi molto considerevoli. Oltretutto, gli autoveicoli richiedono copertura totale ovunque, dalle città alle aree più rurali, per essere sempre connessi, ed impensabile che la comunicazione V2I sia sempre disponibile. Di conseguenza, l'incorporazione di connettività alternative per i servizi del veicolare non solo sono auspicabili ma anche essenziali.
Tuttavia, questo porta con se importanti sfide in termine di ingegneria di rete e protocolli. Come spiegato precedentemente, gli autoveicoli si connetteranno ad una varietà di reti in futuro. In base al servizio desiderato, i veicoli avranno necessità di selezionare la rete appropriata per scambiare dati. Queste reti condividono alcune caratteristiche degli autoveicoli e i requisiti seguenti:

1. Comunicazione intermittente e breve: l'intermittenza è prevedibile a causa della elevata velocità di spostamento in certe regioni. A prescindere dalla copertura cellulare, la comunicazione V2I non è sempre stabile, ma piuttosto disturbata. Nella comunicazione V2V l'interruzione è anche più severa a causa del limitato range di comunicazione del 802.11p e dai rapidi cambiamenti di topologia che insorgono all'interno di scenari mobili. Nelle reti veicolari implementare lo scambio di dati basato su un modello di comunicazione IP con locazioni fisse è sia costoso che infattibile. Specialmente nella comunicazione V2V, se due veicoli passano vicini l'uno all'altro ad alta velocità la comunicazione deve essere completata in meno di un secondo. E' quindi obbligatorio che ogni servizio si affidi ad un range di comunicazione corto.

2. Routing: nel V2I, i metodi di routing tradizionali con sorgenti e destinazioni fisse non sono adatti per l'elevata mobilità del veicolare. Diventa anche difficile mantenere lo stato delle informazioni di routing ai routers intermedi mentre vengono inoltrati dati su più nodi. Per risolvere questo problema sono state proposte le reti ad-hoc veicolari (anche dette VANET), su diversi protocolli di routing. Nella maggior parte delle applicazioni V2V l'obiettivo è ottenere dati da vicini anonimi, non instaurare una comunicazione con un veicolo in particolare. L'attenzione è spostata sull'informazione in sé, e non su chi ha quella informazione. La cosa più difficile nel V2V è identificare chi possiede l'informazione voluta data la natura anonima della comunicazione.

3. Sicurezza e privacy: i requisiti per la sicurezza nell'ambito del veicolare sono più importanti che in qualsiasi altro tipo di rete, dal momento che vulnerabilità nelle autovetture possono condurre a situazioni di pericolo o incidenti. Dato il breve range di comunicazione proteggere il percorso tra due nodi non sarebbe fattibile. Quindi, gli odierni SSL e IPsec non sono applicabili nel V2V. Tutti i dati sensibili a bordo del veicolo, come l'itinerario del veicolo, la musica selezionata il comportamento del conducente, etc. , devono essere protetti di conseguenza.

1.3.2 Applicabilità di NDN alle reti veicolari

Nei paragrafi precedenti sono state introdotte le principali caratteristiche di NDN nelle reti cablate con connettività Internet. Qui si vedrà quali sono i vantaggi che NDN può portare alle reti veicolari, oltre che le sfide che rimangono aperte nella sua realizzazione.

- **NDN.** NDN trasforma la comunicazione di rete in favore del *cosa* invece del *dove*. Tutta la comunicazione in NDN avviene tramite due distinti tipi di pacchetti: *Interest* e *Data*, ciascuno dei quali trasporta un *nome* che identifica univocamente una parte del dato che può essere trasportato nel pacchetto. Il Consumer mette il nome dentro un pacchetto di Interest e lo invia alla rete. I router utilizzano il nome per fare il forwarding dell'Interest verso il Producer del dato, e il Data il cui nome possiede il miglior match viene ritornato al Consumer.

Tutti i pacchetti di dato portano con se una signature che lega il nome al dato. Il modello sembra ben adattarsi alle applicazioni nelle reti veicolari. Tutta via, le applicazioni V2V hanno maggiori requisiti di progettazione delle reti cablate.

- **Adattabilità di NDN.** La sfida maggiore per le industrie degli autoveicoli è far sì che tutti i loro servizi che attualmente eseguono su più tipologie di rete convergano su una unica rete. La figura 1.7 mostra l'architettura di sistema utilizzando due differenti astrazioni, IP ed NDN. I principali vantaggi che NDN introduce nelle reti veicolari derivano dagli scenari V2V descritti di seguito.

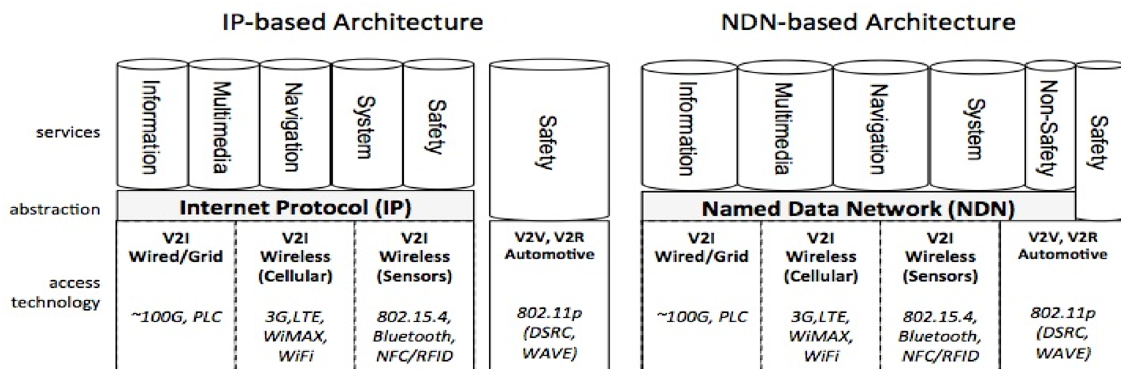


Figura 1.7: IP vs NDN

1. *Recupero dei dati su connettività intermittente*: in una rete NDN la comunicazione avviene in un workspace ben costituito e noto a tutte le parti partecipanti. Tutte le informazioni possono essere mappate a un ben definito, forse multidimensionale, spazio dei nomi (namspace). Se viene stabilito una convenzione per i nomi da dare alle informazioni, poi un veicolo può facilmente esprimere Interest per queste informazioni. Con convenzioni stabilite sui nomi, il modello richiesta/risposta di NDN sembra adattarsi bene alle necessità dello scambio di dati nel V2V ogni qualvolta che due veicoli sono l'uno nel range dell'altro.

2. *Separazione dei dati dai dispositivi di comunicazione*: un largo numero di applicazioni V2V coinvolgono dati che non appartengono ad un

singolo veicolo. Ad esempio, se viene creato un evento per notificare il fatto che si è verificato un incidente, l'informazione non è legata ad un veicolo in particolare. Qualunque veicolo che passa potrebbe potenzialmente generare informazioni relative all'accaduto e disseminarle attraverso la rete. Il modello prevede semplicemente la pubblicazione delle informazioni su un certo namespace precedentemente definito. Quando un veicolo è interessato ad avere certe informazioni, semplicemente inoltra la sua richiesta al namespace seguendo la convenzione di nomi prevista, inserendo il nome all'interno dell'Interest, e senza bisogno di sapere l'indirizzo IP di destinazione.

3. *Il modello orientato ai dati fa un miglior uso del canale broadcast*: per lo standard attuale del 802.11p, V2V e V2R sono per natura comunicazioni broadcast, piuttosto che unicast. Quando si recuperano informazioni da un ben definito namespace, uno può trarre vantaggio dal layer fisico. Quando un Interest viene inviato, questo è mandato in broadcast sul canale, e tutti i veicoli nei paraggi possono ricevere il pacchetto e capire quale dato si sta cercando. Se capita che una qualunque di questi macchine ha il dato cercato, questa può rispondere. Per ridurre il numero di collisioni è richiesto un protocollo diverso rispetto a quello del 802.11, soprattutto quando il numero di veicoli nei paraggi è elevato. Analogamente, quando un Content viene inviato in broadcast, tutti i veicoli nei paraggi possono riceverlo. Alcuni veicoli potrebbero essere interessati alle informazioni e memorizzarle nel Content Store; anche i veicoli che non sono interessati all'informazione potrebbero decidere di tenere l'informazione in cache, in questo potenzialmente potrà essere utile in futuro per la ricezione di qualche Interest per l'informazione messa in cache.

4. *Supporto per la tolleranza all'interruzione di comunicazione*: l'elevato grado di dinamicità nelle reti veicolari porta a due sfide. In primo luogo, dato che la connettività è intermittente, la comunicazione dei dati deve avvenire con un certo grado di tolleranza ai ritardi e alle interruzioni. Una parte di questa tolleranza viene fornita implicitamente tramite la possibilità di nominare i dati direttamente e di utilizzare il sistema di caching delle informazioni. La convenzione dei nomi condivisa da tutti i veicoli introduce il contesto di comunicazione a priori, in modo che ogni veicoli possa scambiare Interest e Content tra di

loro a qualunque istanza essi siano connessi. La disseminazione delle informazioni avviene quindi sia attivamente (tramite gli Interest) che passivamente (tramite l'ascolto del canale broadcast), e le informazioni vengono propagate sia grazie al range di comunicazione che al movimento degli autoveicoli. In secondo luogo, la natura discontinua della comunicazione V2V non si presta a stabilire e mantenere percorsi, i quali possono essere molto costosi o ancora di più non realistici. Quindi, l'idea di utilizzare i protocolli convenzionali per distribuire la connettività non è applicabile nelle reti ad hoc veicolari. Invece del tradizionale approccio di ottenere la raggiungibilità dai protocolli di routing, una nuova soluzione deve essere sviluppata al fine di poter effettuare il forwarding degli Interest verso la sorgente delle informazioni desiderate (questo verrà illustrato successivamente).

5. *Sicurezza facilmente implementabile*: la sicurezza dei dati è direttamente costruita all'interno del progetto NDN, dal momento che ogni pacchetto di tipo Data deve portare una signature valida. E' ragionevole pensare la fattibilità di poter mettere, da parte delle case di produzione di autoveicoli, chiavi criptografiche all'interno del processore dei veicoli, che può essere poi usata per firmare i pacchetti che vengono generati. Con ogni macchina equipaggiata con una chiave per firmare i pacchetti, i veicoli possono pubblicare le informazioni, collezionarle e verificarle tramite la chiave di root delle case di produzione.

1.3.3 Sfide nell'applicare NDN a reti veicolari

Anche se NDN sembra promettente come una effettiva soluzione per il veicolare, fino ad ora le prove effettuate dalla ricerca NDN si son largamente focalizzate su reti Internet cablate. Solo di recente si sta lavorando ad estensioni per reti non cablate. Come applicare NDN a reti ad-hoc veicolari è ancora totalmente da esplorare. Sono stati individuati i seguenti problemi tecnici, che devono essere presi in considerazione per sviluppare un framework completo di NDN per i servizi di autovetture:

- *Forwarding dei dati*: potrebbe essere raggiunto con una buona progettazione dei nomi e l'adozione del georouting. Per la consegna delle risposte abbiamo visto in precedenza che nelle reti wired quello che si fa è tener traccia delle interfacce da cui arrivano le richieste, per poi

essere utilizzate una volta che vengono consegnate le risposte. Questa soluzione nel V2V non è applicabile, perché un veicolo potrebbe avere solo la sua interfaccia wireless broadcast. Non solo, l'elevata dinamicità rende infattibile non solo il routing, ma anche l'uso delle tracce, quindi è necessario trovare una soluzione per il problema della consegna.

- *Naming dei dati*: la convenzione dei nomi per i messaggi di informazioni è estremamente importante. Ci dovrebbe essere un buon compromesso tra specificità e generalità. Per cui, parte del naming è strettamente collegato alle applicazioni utilizzate sulla rete NDN. La sfida più grande per gli scenari di autoveicoli è quindi individuare le applicazioni potenziali e determinarne le similitudini.

- *Ambito di applicazione geografico*: mentre milioni di veicoli potrebbero essere potenzialmente coloro che pubblicano informazioni, le applicazioni potrebbero essere solamente interessati alla locazione dei dati all'interno di una specifica area geografica. La nozione di geolocazione (non locazione topologioca) è così molto importante per gli scenari di autoveicoli per recuperare i dati desiderati.

- *Ambito di applicazione temporale*: definisce quanto un contenuto è valido dopo essere stato pubblicato. In uno scenario mobile, le informazioni invecchiano velocemente. É necessario accertarsi che una informazione ricevuta da un veicolo abbia ancora senso al momento della ricezione. Le informazioni dovrebbero quindi contenere un timestamp.

- *Autenticità dei dati*: il sistema deve fornire meccanismi per accertare l'autenticità dei dati. La chiave per firmare i messaggi è una delle proposte.

- *Interruzione della connettività*: quando non è presente alcuna infrastruttura, non si può assicurare la connettività tra i veicoli. La ritrasmissione degli Interest va presa in considerazione, fintanto che il dato non viene ricevuto. Questo Interest ritrasmesso dovrebbe essere aggiornato di conseguenza se l'ambito temporale o geografico è cambiato dall'ultima trasmissione.

1.3.4 Un esempio di applicazione

L'applicazione può essere usata dai veicoli per richiedere diversi tipi di informazione che avvengono in tempo reale lungo la strada. Un esempio può essere informazioni riguardanti la congestione del traffico per via dell'ora di picco, chiusura di corsie, eventi sportivi di una certa rilevanza. Alla ricezione di informazioni di congestione il veicolo potrebbe computare un percorso ottimale alternativo verso la destinazione. Questo tipo di disseminazione di informazione è attualmente disponibile attraverso i servizi di telefonia e satellitare. Nella figura 1.8 è visibile una panoramica dell'architettura telematica proposta per i veicoli. La head unit fornisce, ad esempio, informazioni sul traffico all'automobilista. Le informazioni possono essere reperite dagli altri veicoli attraverso la comunicazione V2V tramite quello che viene definito NDN forwarding engine che è presente all'interno di ogni veicolo. L'obiettivo primario del sistema è fornire all'automobilista informazioni preziose attraverso la head unit del veicolo (oppure attraverso altri dispositivi che l'automobilista potrebbe portare con se, come ad esempio smartphone o tablet). A questo fine, si fa l'assunzione che ogni veicolo sia equipaggiato con il NDN forwarding engine (detto anche modulo NDN) che gestisce una o più interfacce di rete, come DSRC/WAVE e WiFi. Nell'applicazione di esempio, quando l'applicazione della navigazione che risiede nell'head unit interagisce con il modulo NDN per recuperare le informazioni desiderate, il modulo NDN può usare la WiFi in modalità ad-hoc e/o DSR/WAVE per inviare gli Interest agli altri veicoli attraverso un singolo hop in modalità broadcast (qui si farà riferimento a invii a singoli hop, ma in seguito verrà esplorato l'invio attraverso più hop), senza il supporto di alcuna infrastruttura. Analogamente, il veicolo potrebbe recuperare le informazioni anche dalle road side unite, che sono anch'esse equipaggiate con modulo NDN. Oltre a richiedere i dati, ogni veicolo ne genera anche, sia periodicamente che tramite sensori di camera e sistema di sicurezza (come ad esempio l'anticollisione). Questi dati generati localmente saranno tutti memorizzati. Ogni veicolo può avere sufficiente spazio di memoria per memorizzare i dati. Ogni veicolo gioca tre ruoli:

- data consumer: richiede i dati dalla rete;
- data producer: sorgente delle informazioni e generatore di pacchetti derivanti da richieste o sensori intelligenti;

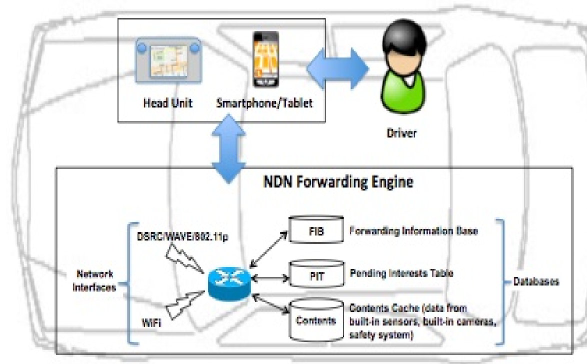


Figura 1.8: Architettura telematica del veicolo

- data mule: colleziona e trasporta informazioni per altri veicoli;

Dal punto di vista del terzo ruolo è auspicabile che esso collezioni più dati possibili. La capacità di memoria sui veicoli non è un grosso problema. La collezione dei dati è un requisito importante mentre si progetta la convenzione relativa ai nomi.
Un modulo NDN contiene tre grandi database: FIB, PIT e Content Store. Applicando NDN al V2V e supponendo di avere una comunicazione a singolo hop, la FIB non viene usata. Si esamina il caso in cui quando un Interest viene inviato in broadcast da un veicolo, può essere ricevuto da un singolo hop tra i veicoli vicini e RSUs. In uno scenario in cui i veicoli si muovono a velocità elevate la probabilità di non ricevere risposta agli Interest è più alta rispetto al caso in cui si hanno reti cablate, quindi si devono considerare differenti schemi nella gestione della PIT. Quando è stato illustrato il funzionamento di NDN è stato detto che i router accettano pacchetti dati se e solo se esiste una corrispondenza nella PIT. Nel V2V, data la natura broadcast della comunicazione, un veicolo può sempre percepire pacchetti di dati in risposta a richieste inviate da altri veicoli. Al fine di servire il ruolo di buon data mule, ogni veicolo dovrebbe cogliere l'opportunità e mettere in cache il pacchetto origliato, invece che ignorarlo. Inoltre, anche le RSUs possono agire come data mule e immagazzinare i dati, con l'unica differenza che non si possono muovere fisicamente, diventando così data mule statici, passando informazioni da veicolo a veicolo. Di seguito verrà illustrato come vengono risolte alcune problematiche poste nei paragrafi precedenti.

Convenzione dei nomi: si propone di usare per l'esempio in oggetto un formato del tipo *Location/DataType*. Per il componente *Location* si potrebbe usare l'identificatore della strada con assegnati i numeri di sezione (in alternativa si potrebbero usare latitudine e longitudine). Il componente *DataType* potrebbe essere uno dei predefiniti set di valori, come ad esempio *Traffic*, *Accident*, *RoadSide*, *Trip Time* etc . Ognuno di questi esprime una informazione desiderata.. Per la seguente spiegazione, supponiamo di usare *Traffic* come nome rappresentativo. In questo caso, il formato completo dell'Interest sarebbe */RoadId/SelectionNumber/Traffic* . Un Interest corrisponderebbe ad una sezione di una strada che un veicolo necessita seguire prima della sua destinazione. Il *RoadId* rappresenta un nome univoco della strada. In casi di omonimia di nomi di strade per stati diversi si potrebbe pensare di inserire un identificativo dello stato e della città per ottenere l'univocità completa. Sulle highways, i numeri di sezione potrebbero essere i numeri delle uscite. La convenzione è quindi molto importante per esprimere Interest riguardanti l'area geografica di interesse a diverse granularità.

Ambito temporale: vengono definiti tre tipi di dato necessari che dovrebbero essere inclusi nel pacchetto: tipo di evento, geolocalizzazione dell'evento e timestamp. Il tipo di evento potrebbe essere un valore predefinito; la geolocalizzazione potrebbe essere sia le coordinate che i numeri di sezione; il timestamp registra quando l'evento è avvenuto. Alla ricezione di un pacchetto, sta all'applicazione decidere se memorizzare il dato o considerarlo obsoleto.

Autenticità dei dati: consiste nel verificare, da parte del richiedente, la validità del dato e allo stesso tempo mantenere la privacy di coloro che pubblicano dati. Vengono utilizzati i certificati, che vengono poi eliminati dopo essere stati usati per diverso tempo, ad esempio cinque minuti. Oppure possono essere usate firme di gruppo. Lo schema della signature di gruppo consente ai richiedenti di verificare se il dato viene da un gruppo, ma non può dire quale individuo del gruppo lo ha firmato.

Interruzione della connettività: in questi casi il richiedente deve esprimere nuovamente l'interest.

> Propagazione dei pacchetti: si comincia con un modello di propagazione semplice degli Interest. Quando un veicolo riceve un Intrest, lo invia in broadcast a tutti i veicoli nelle vicinanze. Lo stesso avviene per i pacchetti di Content. É una forma di flooding tra veicoli. Deve esserci un meccanismo di soppressione dei pacchetti tale per cui si possa evitare la ripetuta lettura del medesimo pacchetto. La PIT può essere utilizzata per individuare Interest duplicati, e il Content Store per individuare dati duplicati.

La progettazione dei nomi è fondamentale; qui è stato preso in considerazione un caso specifico, ma i nomi utilizzati in applicazioni per il veicolare hanno necessità di avere, in generale, informazioni relative sia alla locazione che al tempo, con granularità flessibile rispetto ad entrambe le dimensioni. Il nome di un dato consiste quindi in una sequenza lineare di componenti e può facilmente rappresentare dati di differenti granularità lungo le dimensioni. Come progettare la struttura dei nomi con una certa flessibilità e granularità rimane comunque un problema aperto. Il modello utilizzato da NDN consente la disseminazione delle informazioni usando i nomi dei dati definiti a priori al momento dello sviluppo dell'applicazione e sono comprensibili a tutti i veicoli. Questo modello request-response meglio si adatta allo scambio di dati nelle reti veicolari, in cui i cambiamenti sono molto rapidi e la durata della connettività tra i veicoli è molto breve. L'efficacia e flessibilità di questa soluzione dipende molto da una opportuna selezione della struttura dei nomi, e può essere utilizzata dai veicoli per esprimere quali dati desiderano sotto diverse situazioni e quando giocano ruoli diversi.

Capitolo 2

Strumenti per la mobilità

Le reti veicolari, anche note come *VANET* (Vehicular Ad-Hoc Network), hanno attratto un sempre più crescente interesse sia nell'ambito della ricerca che nell'ambito delle industrie automobilistiche. Una delle sfide più importanti e difficili che si pongono le VANET è quello della definizione di un modello di mobilità veicolare accurato e il più possibile realistico, sia a livello microscopico che a livello macroscopico. L'altra importante sfida è la capacità di modificare questo modello di mobilità veicolare in funzione del protocollo di comunicazione veicolare utilizzato. Come spiegato in [3], è possibile avere e seguire una linea guida per la generazione di modelli di mobilità veicolare. Qui si riportano le principali tematiche che devono essere prese in considerazione quando si cerca di creare un modello di mobilità il più realistico possibile.

2.1 Linee guida per la costruzione di una mobilità veicolare

La comunicazione tra veicoli è vista come tecnologia chiave per migliorare la sicurezza e le comodità al volante. Questo crescente interesse ha spinto a sviluppare tecnologie e protocolli per la comunicazione tra veicoli, e per la comunicazione tra veicoli e infrastrutture. Le VANET rappresentano una classe di MANETs (Mobile Ad Hoc Networks). Le VANET sono reti distribuite, auto-organizzanti costruite dal movimento dei veicoli e sono caratterizzate da velocità elevate e limitati gradi di libertà nei pattern di

movimenti dei nodi stessi. Queste caratteristiche rendono inefficiente o impossibile l'utilizzo degli standard protocolli di rete. Il principale obiettivo che si pongono i protocolli sviluppati per le VANET è fare sì che lo scambio di messaggi modifichi il traffico stradale, sia per ragioni di sicurezza che per evitare ingorghi.
Un aspetto critico quindi nelle simulazioni delle VANET è la necessità di avere un modello di mobilità che riflette il comportamento reale del traffico veicolare.

2.1.1 Un framework per modelli di mobilità veicolare

I modelli di mobilità in letteratura sono classificati come *macroscopici* e *microscopici*. Il modello di descrizione *macroscopica* riguarda diversi aspetti tra cui la densità veicolare, velocità media e tratta il traffico veicolare come fluido dinamica. Il modello *microscopico* considera invece ogni veicolo come entità distinta, modellando il suo comportamento in maniera più precisa ma computazionalmente più costosa. Si consiglia di identificare blocchi funzionali: vincoli di movimento, generatori di traffico, tempo e influenze esterne. Da un lato, i *vincoli di movimento* definiscono i gradi di libertà relativi ad ogni veicolo. A livello macroscopico, i vincoli di movimento sono le strade o gli edifici, ma a livello microscopico sono costituiti dai veicoli vicini, dai pedoni o dai diversi modelli di macchine o dalle abitudini degli automobilisti. Dall'altro lato, il generatore di traffico definisce diversi tipi di macchina e gestisce le interazioni in base all'ambiente sotto studio. Macroscopicamente modella le densità di traffico, velocità e flussi, mentre microscopicamente ha a che vedere con proprietà come la distanza tra veicoli, accelerazioni, frenate e sorpassi. Un altro importantissimo aspetto è il *tempo*, che è visto come il terzo blocco funzionale, il quale descrive configurazioni di mobilità diverse per uno specifico momento della giornato o giorno della settimana. L'ultimo blocco è rappresentato dalle *influenze esterne*, che modella l'impatto del protocollo di comunicazione o ogni altra sorgente di informazione sul pattern di movimento.
Un modello di mobilità che intende generare pattern di mobilità realistici dovrebbe possedere le seguenti caratteristiche (cioè i seguenti blocchi funzionali):

- una accurata mappa topologica: la differente topologia delle strade dovrebbe gestire diverse densità di interazione, contenere corsie multiple,

diverse categorie di strade e relativi limiti di velocità;

- ostacoli: nel senso ampio del termine, sia come vincoli per il movimento che come ostacoli alla comunicazione wireless;
- punti di attrazione/repulsione: sorgente e destinazione del viaggio non sono random. La maggior parte del tempo gli automobilisti si muovono verso destinazioni simili, chiamati punti di attrazione (ad esempio l'ufficio), o si muovo da locazioni simili, chiamati punti di repulsione (ad esempio la casa), caratteristiche che creano dei colli di bottiglia.
- caratteristiche dei veicoli: ogni veicolo ha le sue caratteristiche, che hanno un impatto sui set dei parametri di traffico. Ad esempio, a livello macroscopico alcune strade sono proibite a certi veicoli; a livello microscopico accelerazioni, decelerazioni e velocità di camion e macchine saranno diverse. Questo influisce sul generatore di traffico;
- movimento nel viaggio: il viaggio, a livello macroscopico è visto come un insieme di sorgenti e punti destinazione in un area urbana. Automobilisti diversi potrebbero avere interessi diversi che influiscono la scelta del viaggio;
- Percorsi di movimento: un percorso è visto a livello macroscopico come un insieme di segmenti intrapreso da una macchina all'interno del suo viaggio tra un punto iniziale e uno finale. La scelta della rotta non è casuale, ma fatta tenendo conto dei limiti di velocità, del periodo della giornata, del livello di traffico, della distanza e anche delle abitudini personali;
- accelerazioni e decelerazioni: sono considerati anche questi modelli;
- pattern di guida: non vi sono solo i vincoli dovuti a all'ambiente, ma anche quelli dovuti agli altri veicoli e pedoni;
- gestione degli incroci: gestione della segnaletica quale gli STOP, i semafori o i vigili;
- pattern temporali: la densità di traffico non è la medesima durante la giornata; una densità eterogenea si osserva durante le ore di

punta o eventi speciali, influenzando i vincoli di movimento e il generatore di traffico, che potrebbe alterare il percorso e i punti di attrazione/repulsione;

- influenze esterne: impatto dovuto a incidenti, chiusura di strade o eventi non modellabili a priori;

Maggiore è il numero di elementi sopra citati presi in considerazione, tanto più realistico sarà il modello. Esistono quindi diversi approcci alla modellazione della mobilità che hanno come base questi elementi. Blocchi come le *mappe topologiche*, *generatore di veicoli* o *generatore del comportamento degli automobilisti* non possono essere random ma devono riflettere configurazioni reali. Data la grande complessità di ottenere un certo tipo di informazioni, la varie comunità di ricerca hanno effettuato assunzioni più semplicistiche eliminando totalmente alcuni blocchi funzionali o fornendo versioni molto esemplificative degli stessi.

2.1.2 Classi di modelli di mobilità

I modelli di mobilità possono essere classificati in quattro categorie:

- *Synthetic Models*: tutti i modelli basati su modelli matematici;
- *Survey Models*: estraggono pattern di mobilità da sondaggi e ricerche ;
- *Trace Models*: genera pattern di mobilità da tracce di mobilità reale;
- *Traffic Simulator Models*: le tracce di mobilità veicolare sono estratte da dettagliati simulatori di traffico;

2.1.3 Synthetic Models

Questo modello può essere ulteriormente scomposto in altre cinque sottoclassi:

- *Stochastic Models*: contengo modelli puramente random;
- *Stream Models*: trattano la mobilità veicolare come se fosse idrodinamica;

- *Car Following Models*: il comportamento di ogni veicolo è modellato secondo quello dei veicoli che lo precedono;
- *Queue Models*: le strade sono modellate come code FIFO è le macchine come client;
- *Behavioral Models*: ogni movimento è determinato da regole comportamentali come le influenze sociali;

La validazione dei modelli matematici è un passo importante al fine di garantire il suo realismo comparato a mobilità reali. Una soluzione è prendere tante tracce di mobilità derivanti da grandi campagne di misure e poi comparare i pattern con quelli sviluppati dai modelli matematici. Un forte limite di questo tipo di modelli deriva dalla complessità di modellare in maniera dettagliata i comportamenti umani.

2.1.4 Survey-based Models

I sondaggi sono una importantissima sorgente di informazioni sulla mobilità macroscopica. La maggiore parte di questi sono forniti dallo US Department Labor and Statistics. Sulla base di questi dati, il simulatore di mobilità modella il tempo di arrivo al lavoro, il tempo del pranzo, della pausa, degli impegni, le dinamiche di pedoni e veicoli e anche il tempo di lavoro, come la durata degli incontri e la loro frequenza. Il traffico veicolare è derivato dalle statistiche del traffico veicolare collezionate nei vari stati dai governi locali, in modo da poter modellare le dinamiche dei veicoli e l'uso giornaliero delle strade. E' bene citare anche l'Agenda-based mobility model, che combina sia le attività sociali che i movimenti geografici. Il movimento di ogni nodo è basato su una agenda individuale che include tutti i tipi di attività in uno specifico giorno. I dati derivanti dallo US Nation Household Travel Survey sono stati utilizzati per ottenere la distribuzione delle attività, l'occupazione della distribuzione.

2.1.5 Trace-based Models

Un altro tipo di approccio è quello che prevede l'estrazione di generici pattern di mobilità dai movimenti contenuti nelle tracce. Questo metodo ha ottenuto una sempre più crescente popolarità da quando le tracce sono state ottenute attraverso diverse campagne di misurazione lanciate da diversi

progetti. La parte più difficile di questo approccio è quella di estrarre pattern non osservati direttamente dalle tracce. Usando modelli matematici complessi è possibile prevedere pattern di mobilità non riportati nelle tracce. La limitazione è spesso legata alla campagna di misura; ad esempio, se le tracce sono state ottenute per i sistemi degli autobus, un modello che estrapola i dati non può essere applicato per il traffico dei veicoli privati. Un'altra limitazione deriva dalla limitata disponibilità di tracce veicolari.

2.1.6 Traffic Simulator-based Models

Diversi gruppi di ricerca, raffinando il Synthetic Model e passando attraverso un forte processo di validazione su tracce reali o sondaggi ha sviluppato veri e propri simulatori di traffico realistici, tra cui si cita PARAMICS, CORSIM, VISSIM, TRANSIMS e SUMO. Questi sono capaci di simulare modelli di traffico urbani microscopici, consumo energetico, o anche l'inquinamento e il monitoraggio di livello dei rumori. Tuttavia, non possono essere usati immediatamente per i simulatori di rete, poiché spesso l'interfaccia con questi ultimi è assente e le tracce sono incompatibili. SUMO, ad esempio, ha consentito l'esportazione delle tracce in modo che siano usabili dal simulatore di rete. In più, molti di questi simulatori di traffico sono commerciali, e richiedono quindi l'acquisto di una licenza. Lo svantaggio principale di questo approccio è l'elevata complessità della configurazione di questi simulatori.

2.1.7 Validazione

Uno strumento fondamentale senza il quale avrebbe senso parlare di mobilità realistiche è la validazione. L'unico modo di validare il realismo è comparare i pattern di movimento con una topologia reale. La validazione può essere eseguita in due modi:

- computare l'errore tra le tracce generate dal Synthetic model e le tracce di mobilità reale;
- in assenza delle tracce di mobilità reale, si usano tracce di un modello già validato.

Sia alcuni simulatori di traffico commerciali (CORSIM e VISSIM) che alcuni free (SUMO, SHIFT) sono stati validati con tracce reali.

2.2 Modelli di mobilità e simulatori di rete

I modelli di mobilità devono poter essere resi accessibili ai simulatori di rete. Modelli di mobilità e simulatori di rete non sono mai stati pensati per comunicare tra loro e sono stati progettati per essere controllati separatamente. Quando si pensa alle applicazioni promettenti che possono essere ottenute dalle reti veicolari, dove la comunicazione può modificare la mobilità, e dove la mobilità potrebbe migliorare la capacità della rete, questa assenza di comunicazione potrebbe determina una ulteriore battuta di arresto per lo sviluppo di reti veicolari. E' importante che vi sia un livello di interazione tra il modello di mobilità e il simulatore di rete. Anche qui esistono diversi modelli che si illustrano brevemente:

- *Isoleted Mobility Model*: il simulatore di rete è in grado di caricare scenari di mobilità. I diversi modelli necessitano di essere generati a priori, prima della simulazione, e devono essere parsati dal simulatore di rete secondo un formato predefinito della traccia. Gli scenari non possono subire modifiche, e di fatto non esiste nessuna interazione. La maggior parte dei modelli disponibili ricade in questa categoria. Il vantaggio è che lo sviluppo della mobilità e del simulatore sono totalmente indipendenti.

- *Embedded Mobility Model*: la mobilità e il simulatore di rete sono sostituiti da simulatori a eventi discreti. Spesso però in questo modello il simulatore di rete è molto semplicistico, e porta con se molto pochi protocolli. In questo modello il simulatore genera la mobilità e al tempo stesso simula la rete con i suoi protocolli, quindi l'interazione tra i due è nativa.

- *Federated Mobility Model*: mobilità e simulatore utilizzano una interfaccia per comunicare. Vi è una infrastruttura di simulazione composta da due parti di simulatori, che eseguano in maniera distribuita. Da una parte si ha un simulatore di traffico validato, dall'altra un simulatore di rete e poi un software che si interpone tra i due fornendo servizi per lo scambio di dati e la sincronizzazione tra i due. Deve essere quindi definito un formato comune per i messaggi. L'interazione è dinamica, quindi ad esempio i flussi di mobilità possono essere modificati dai flussi di rete e viceversa. La più grande limitazione deriva,

come già anticipato, dalla complessa configurazione del simulatore di traffico. Un modello di questo tipo è ad esempio adottato da SUMO (che è il generatore di traffico) ed NS-2 (che è il simulatore di rete). Utilizzando una interfaccia chiamata *Interpreter*, le tracce estratte da SUMO possono essere trasmessa a NS-2, e viceversa, istruzioni possono essere inviate da NS-2 a SUMO per la messa a punto del traffico (questo è l'approccio *TraNS* [4]) .

2.3 Criteri di scelta derivanti dai requisiti dell'applicazione

Sono forniti quattro modelli principali:

1. Motion Constraints;
2. Traffic Generator;
3. Simulator Related;
4. Recommendation;

L'obiettivo è fornire tutti gli elementi in modo che il lettore possa comparare le varie caratteristiche e scegliere quello che maggiormente si adatta alle sue esigenze. Più blocchi funzionali include il modello, più realistico sarà. Di seguito vengono riportati i blocchi funzionali che dovrebbero essere inclusi per rendere il modello il più realistico possibile.

Motion Constraints
Si occupa di tutto quello che riguarda gli impedimenti che alterano lo spazio libero della mobilità. Non si considera solo la topologia delle strade, ma si includono anche l'influenza derivante dai pattern di movimento dei veicoli derivanti, ad esempio, dalla politica degli incroci, dai limiti di velocità, dai sensi unici, dalle caratteristiche delle multicorsie o anche dagli effetti dei punti di interesse, quindi se definiscono i seguenti criteri:

- *Grafo*: la mobilità è ristretta ad un grafo. Rappresenta un punto chiave; i grafi possono essere *user-defined*, cioè quando la topologia è specificata come lista di vertici e segmenti; *random*, quando

generato da metodi come Voronoi, Spider o Manhattan Grid; *maps*, quando viene estratta da mappe reali, come ad esempio TIGER, GIS o GDF.

- *Punti Sorgente e Destinazione*: potrebbero essere casuali, casuali ristretti al grafo, o basati su punti di attrazione/repulsione. Il modello di sorgente e destinazione random è piuttosto irreale, ma anche il più diffuso. I punti di attrazione/repulsione hanno caratteristiche ad esempio dipendenti dal periodo, dall'istante temporale del periodo della giornata, che li rendono preferibili rispetto ad altri punti di attrazione/repulsione presenti sulla topologia.
- *Politiche degli incroci*: descrivono il tipo di incrocio e le politiche, come ad esempio segnali di STOP e semafori (se presenti). Gli incroci sono visti come ostacoli statici.
- *Multi-corisa*: la topologia include più corsie.
- *Limiti di velocità*: la topologia include limiti di velocità, dipendenti dal segmento di strada.

Traffic Generator
Genera i veicoli e modella la loro mobilità rispettando tutti i vincoli imposti dal blocco funzionale precedente. Questo blocco potrebbe essere a sua volta suddiviso in vari blocchi:

- *Trip Generation*: genera un viaggio random tra sorgente e destinazione, oppure secondo una sequenza di attività.
- *Path Computation*: fornisce l'algoritmo per generare un percorso completo tra sorgente e destinazione, sulla base del viaggio creato dal blocco precedente.
- *Human Mobility Pattern*: i movimenti e le interazioni con le macchine potrebbero essere ispirati da movimenti umani descritti da modelli matematici.
- *Lane Changing*: descrivo i modelli di sorpasso implementati dal traffic generator, se presente.
- *Velocity*: potrebbe essere uniforme o dipendente dalla strada.

- *Intersection managment*: descrive il comportamento rispettando le politiche degli incroci contenuti nel Motion Constraints, se presente.

Simulator Related
L'interazione con un simulatore di rete è importante, o anche con l'utente finale per fornirgli un output grafico di quanto ottenuto. Si individuano quindi gli ulteriori blocchi:

- *Visualization*: il modello include un tool di visualizzazione, oppure no.
- *Output*: output di rete generato dall'isoleted mobility model. Non è adatto per gli embedded model.
- *Platform*: fornisce il linguaggio di programmazione sul quale il simulatore è costruito.
- *Class*: indica il livello di interazione tra il modello di mobilità e il simulatore.

Recommendation: fornisce una valutazione sull'appropriatezza di ciascun modello di mobilità e aiuta il lettore a scegliere quello che meglio si adatta alle proprio esigenze. Scegliere un modello piuttosto che un altro dipende molto dai requisiti di applicazione della comunicazione del veicolare. Si utilizzano i seguenti criteri:

- *Minimum Requirments for Realism*: si valutano i criteri minimi in base allo scenario scelto (urbano o autostrada), o nessuno dei due in base alle caratteristiche di ogni modello.
- *Radio Obstacles*: considera gli ostacoli radio, sia nella forma della topologia per il simulatore di rete, o semplicemente una traccia di propagazione. La disponibilità di ostacoli radio è un requisito fondamentale negli scenari urbani, mentre è meno critico nelle autostrade.
- *Validation*: è il processo di validazione descritto nei paragrafi precedenti.

In un test effettuato da Fiore [5] è stato provato che negli scenari urbani solo modelli che implementano simultaneamente l'interazione car-to-car e car-to-infrastructure sono in grado di passare i test per la teoria del traffico

in ambienti urbani. Negli ambienti urbani quindi, il modello dovrebbe includere, oltre ai pattern di guida degli automobilisti, anche la gestione degli incroci, implementando di conseguenza una interazione tra il Motion Constraints e il Traffic Generator. Questi sarebbero requisiti minimi, in quanto non vi sono studi che fino ad ora abbiano fornito quali siano i requisiti sufficienti per modellare la mobilità. Avere requisiti *sufficienti* per un modello di mobilità significa che ogni requisito extra non avrebbe alcun impatto sui pattern di movimento. Per definizione, questo insieme di requisiti dipende dall'applicazione.
I simulatori di traffico sono stati sviluppati per analizzare la mobilità veicolare sia a livello macroscopico che microscopico con un livello di dettaglio molto alto. In SUMO (*Simulation of Urban Mobility Scenario*), la parte di Traffic Generator e assegnamento delle rotte può essere importata da varie sorgenti. Il modello dei pattern di guida è quello del Car Following e il path motion utilizza un modello stocastico per l'assegnamento del traffico tramite una scelta probabilistica delle rotte. Per il Motion Constraints, contiene dei parser per vari formati, tra cui si ha TIGER e GIS Arcview. Inoltre, è in grado di produrre in output tracce usabili dai simulatori di rete. Il Car Followig Model (CFM) è la classe più specializzata per i modelli microscopici che implementano i pattern di mobilità umana. L'adattamento per il car following avviene secondo un insieme di regole il cui obiettivo è quello di evitare qualunque tipo di contatto con il veicolo che precede lungo il senso di marcia. Esistono diversi modelli che implementano il CFM, di seguito si cita:

- Krauss Model (KM)
- Nagel and Schreckenberg Model (N-SCHR)
- Wiedeman Psycho-Physical Model (Psycho)
- General Motors Model (GM)
- Gipps Model (GP)
- Intelligent Driver Model (IDM)

2.3.1 Simulatori di rete

Esistono diversi tipi di simulatori di rete, commerciali e non. Tra quelli commerciali uno dei più utilizzati è *Opnet Modeler*, un simulatore a eventi discreti molto efficiente che contiene centinaia di protocolli di rete, inclusa una suite wireless completa. Un altro simulatore che ricade in questa categoria è *Qualnet* che ha la peculiarità di gestire la computazione multiprocessore. Come Opnet, contiene diverse librerie per i protocolli di rete wireless, tra cui la suite MANET. Tra questi troviamo anche *OmNet++*. Tra i simulatori di rete free troviamo NS-2, ora disponibile anche NS-3 [6]. NS-2 diventò uno degli standard de facto come simulatore di riferimento per le MANETs. Contiene il modello dello stack OSI. NS-2 sembra il simulatore più adatto per le reti veicolari.

2.4 Modelli di mobilità

In questo paragrafo verranno illustrati diversi modelli di mobilità che sono in grado di generare tracce per i simulatori di rete pur non interagendo con loro. Esistono classi di modelli che hanno subito una evoluzione nel corso degli anni, che hanno portato ad un loro perfezionamento.

2.4.1 Isoleted Vehicular Model

Questo modello ricade nella seguente evoluzione: *Legacy Models*, *Improved Motion Constraints*, *Improved Traffic Generator*, *Improved Motion Constraints and Traffic Generator*.

1. Legacy Mobility Model: questo modello è stato l'origine di tutte le valutazioni dei protocolli di rete utilizzati nelle MANETs. Sono stati proposti diversi modelli per generare pattern di mobilità dei nodi quando per la prima volta fu considerata la mobilità in simulazioni di reti wireless. I primi modelli utilizzati includono la generazione di spostamenti a velocità costante all'interno di una topologia delimitata. Lavori sviluppati in seguito introdussero pause temporali, accelerazioni e decelerazioni. Tuttavia, l'intrinseca natura di questi modelli di mobilità porta a pattern di movimento non realistici, comparati con comportamenti reali. Per le MANET, utilizzare uno qualunque di

questi modelli produce risultati non significativi. Di conseguenza, la comunità di ricerca ha cominciato a cercare modelli più realistici. Il Freeway model e il Manhattan (o Grid) Model furono i primi passi.

2. Improved Motion Constraints: questo modello prevede l'estrazione dei dati, in particolare della topologia stradale, da database come ad esempio TIGER. Quindi, la topologia è costruita partendo da mappe reali. Tuttavia, continua ad essere un modello random.

3. Improved Traffic Generator: un miglioramento si è ottenuto a partire dall'introduzione del concetto di mobilità umana dinamica, come ad esempio i *punti di attrazione/repulsione*, *attività* e *ruoli*. I punti di attrazione rappresentano destinazioni di interesse per più persone. Le attività consistono nel processo di movimento verso un punto di attrazione e il trattenersi in quel punto, mentre il ruolo rappresenta la tendenza della mobilità di diverse classi di persone. Questo rappresenta un primo tentativo di miglioramento della mobilità.

4. Improved Motion Constraints and Traffic Generator: in tutti i casi sopra elencati manca una interazione tra i blocchi di Motion Constraints e Traffic Generator. Fu ideato un modello in cui la parte di Motion Constraints è gestita da un parser TIGER che include caratteristiche per il multi-lane, mentre il Traffic Generator contiene una interazione di livello molto base tra le macchine che si susseguono al fine di evitare la collisione.

In generale, l'Isolated Model è privo di una interazione tra il Motion Constraints e il Traffic Generator, quindi, applicazioni di safety non possono essere studiate con questo tipo di modello.

2.4.2 Embedded Vehicular Mobility Model

Questo modello è più avanzato rispetto all'Isoleted Model. In alcuni di questi modelli, il blocco funzionale Motion Constraints contiene mappe topologiche dal database TIGER e la distribuzione di mobilità è non uniforme. Quindi, i vincoli di mobilità contengono anche limiti di velocità per le strade e un generatore di percorso basato sulle velocità. Modelli sviluppati in seguito includono il car following model, ma le tracce per il simulatore di rete

ancora non sono disponibili. Altri modelli proposti successivamente cominciano ad includere moduli di simulatori di guida, moduli di propagazione radio e moduli di simulatori di rete. Tutti questi moduli sono collegati da un sistema di feedback in modo che ogni alterazione che subisce un modulo, possa influenzare gli altri.

2.4.3 The Federated Mobility Models

Questo modello presenta molte caratteristiche avanzate per modellare il movimento in reti veicolari e a livello di capacità di rete, ma è anche molto complesso da configurare. I modelli derivanti da questo tipo di approccio sono però i più realistici. Nella categoria degli open-source, viene utilizzato SUMO per la generazione delle tracce che sono poi date ad un simulatore di rete, come ad esempio NS-3. Le tracce fornite da SUMO sono al momento quelle più realistiche.

2.4.4 Alcune considerazioni

Le reti veicolari sono anche un ambiente wireless il cui canale radio si presume abbia un impatto importante almeno tanto quanto la mobilità. La qualità del canale radio tra due veicoli dipende significativamente dalle mutue mobilità. Un modello di propagazione realistico e un modello di fading dovrebbero quindi essere implementati, in modo che possano considerare ostacoli radio e possibili blocchi del segnale. Questa parte è di solito presente nel simulatore di rete. Il modulo radio deve avere accesso ai pattern di mobilità e alle informazioni topografiche.

Capitolo 3

Strumenti utilizzati

Data l'analisi svolta ai paragrafi precedenti, in questo paragrafo si riportano quali sono stati gli strumenti utilizzati per poter creare ciò che sarà la base per le simulazioni, ovvero la mobilità veicolare a la mappa su cui la mobilità si appoggia.

3.1 TIGER

TIGER [7][8] è un database di mappe digitali disponibili senza l'acquisito di alcuna licenza. Contiene informazioni relative a tutte le linee geografiche sopra il territorio statunitense. Le linee geografiche includono strade, ferrovie, confini di contea, linee di costa e molte altre. Ogni linea è divisa in segmenti che sono definiti da due end-point e altre caratteristiche. L'attenzione cade sui segmenti che sono marcati come strade. In questo db le strade sono classificate in diverse categorie. La classificazione è usata da noi per attribuire limiti di velocità e un numero di corsie che seguono un certo schema. TIGER non ha informazioni relative ai sensi unici, quindi i segmenti sono tutti considerati a doppio senso. Il reader di TIGER non fa altro che tradurre "tutti i segmenti" che rappresentano i dati in un grafo orientato con nodi e lati, dove i nodi sono le interconnessioni e i lati sono i segmenti. Il *Road Traffic Pattern* (RTP) è un insieme di input di cui un simulatore di mobilità ha bisogno al fine di costruire la traccia di mobilità. Siccome differenti simulatori di mobilità necessitano di diversi insiemi di input, si definisce il RTP in un insieme di senso più ampio tale che, due simulatori diversi nutriti con lo stesso RTP produrranno due tracce di mo-

bilità con le stesse microscopiche caratteristiche. Il RTP è costituito da diverse primitive:

1. Topologia della strada: una descrizione della rete stradale come grafo orientato;

2. Flusso di input: il numero di veicoli per unità di tempo che entrano da ogni segmento di strada in ingresso (ad esempio le strade ai bordi della mappa);

3. Flusso interno: il numero di veicoli per unità di tempo che attraversano le strade interne della mappa;

4. Probabilità di svolta: per ogni intersezione, la frazione di veicoli che transitano da una strada in arrivo ad una di uscita;

5. Matrice di origine-destinazione: numero di veicoli che partiranno da qualunque segmento di ingresso della strada e viaggeranno a un segmento di strada di uscita.

Una descrizione approfondita di come TIGER classifica le strade è contenuta in [9]; vi è tutta la classificazione delle strade.

3.2 Simulation of Urban MObility (SUMO)

SUMO [10] (Simulation of Urban MObility) è il simulatore (generatore) di traffico open source che permette di simulare il traffico di grandi reti stradali, in particolare a livello microscopico. Ogni veicolo è modellato esplicitamente, ha una propria strada e si muove in maniera individuale sulla rete. L'output finale di SUMO è una traccia di mobilità. Una documentazione completa è possibile trovarla a [11]. SUMO definisce un proprio formato della rete stradale. Una rete SUMO è un grafo orientato. I *nodi* rappresentano incroci o giunzioni, mentre i *lati* rappresentano le strade/vie. In aggiunta a queste informazioni base, una rete SUMO contiene ulteriori informazioni:

- ogni strada ha una collezione di *corsie*;

- vi è una posizione, forma e la velocità limite per ogni corsia;

- contiene la regolazione del diritto di precedenza;
- la posizione e la logica dei semafori;

Ogni lato include la definizione delle corsie. Ogni corsia è definita da:

- un identificatore della corsia;
- un'indicazione sulla corsia di partenza del veicolo;
- una lista delle classi di veicoli a cui è consentito stare nella corsia;
- velocità massima della corsia;
- lunghezza della corsia;

Alle corsie è anche possibile dare una forma. Ogni lato è unidirezionale e comincia da un *fromNode* per terminare in un *toNode*. I parametri degli edge possono essere definiti (se non vengono definiti sono usati dei valori di default), con edge a singola lane e una velocità massima intorno ai 50km/h. La lunghezza dell'edge viene computata come distanza tra lo start e l'end point. Per una descrizione completa di cosa e come è possibile configurare SUMO, si rimanda a [12]. SUMO utilizza anche altri tool che saranno introdotti nei successivi paragrafi e che ora vengono brevemente accennati. Il file che contiene la rete SUMO descrive il traffico relativo ad una certe parte della mappa in oggetto. Principalmente contiene la rete di strade/vie, incroci/giunzioni e semafori presenti nella mappa. Il formato di questi file è interno a SUMO e riflette le sue caratteristiche. Sebbene siano human readable in quanto file XML, non sono pensati per essere scritti a mano, ma generati da altri tool, tra cui *NETCONVERT*, *JTRROUTER* e *VERGILIUS*. Quest'ultimo in particolare, rivestirà un ruolo molto importante nella processo di creazione della mobilità.

3.3 CORNER

Nei paragrafi precedenti è stato detto che un requisito fondamentale per gli scenari di mobilità urbana è il modello di propagazione. Il modello di propagazione utilizzato è *CORNER* [13]. CORNER è un modello ideato per studiare il percorso di propagazione del segnale, in particolare per valutare la presenza di ostacoli lungo il percorso del segnale, e valutarne la

perdita. Considerando quindi una coppia di nodi, si vuole identificare un percorso di propagazione (il più breve) e stimare la perdita tra la coppia. Per identificare tale percorso, tra gli insiemi di segmenti di strade associati a ciascuna coppia, CORNER seleziona i due segmenti (uno per ogni nodo), che coinvolgono il minore numero di intersezioni da attraversare. Esistono tre tipi di classificazione, fatte in base al concetto di vista angolare (*angular view*) del nodo. La vista angolare del veicolo è la porzione di piano che esso può vedere dall'apertura offerta dall'incrocio. Questa apertura dipende dall'impronta degli edifici che circondano l'ambiente. Viene considerato "edificio" qualunque cosa non sia una strada. Le strade sono considerate spazi aperti dove il segnale può propagarsi liberamente. Coppie di nodi sono così classificate:

1. *LINE OF SIGHT* (LOS): quando due veicoli sono mappati sullo stesso segmento di strada; sono considerati tali anche veicoli mappati su segmenti di strada diversi, ma connessi da un incrocio e in cui uno dei due veicoli è nella vista angolare di quell'altro. Inoltre, potrebbero essere considerati tali anche due veicoli che sono mappati su segmenti di strada diversi, separati da due incroci. In questo caso, uno dei veicoli considerati ha differenti angoli di vista, uno per incrocio. Siano definite AW_A e AW_B le viste angolari generate rispettivamente dagli incroci più vicini e più lontani. Se un altro veicolo sta viaggiando in AW_B e AW_B è completamente contenuto in AW_A, allora i due veicoli sono considerati in LOS. Si veda figura 3.1 .

2. *NON LINE OF SIGHT 1* (NLOS1): quando due veicoli sono mappati su segmenti di strade adiacenti e non sono nella vista angolare dell'altro. Sono classificati tali anche quando i veicoli sono mappati su segmenti di strade separati da due incroci e uno dei veicoli è in LOS con l'incrocio più lontano (e cioè una parte del segmento di quel veicolo sta nel cono di vista). Un esempio grafico è visibile in figura 3.2 .

3. *NON LINE OF SIGHT 2* (NLOS2): quando due veicoli sono mappati su segmenti di strade separati da due incroci e in cui uno dei due non è né in LOS né in LOS1 (ovvero il segmento è fuori da qualsiasi angolo di vista dell'altro veicolo). L'esempio grafico è visibile in figura 3.3 .

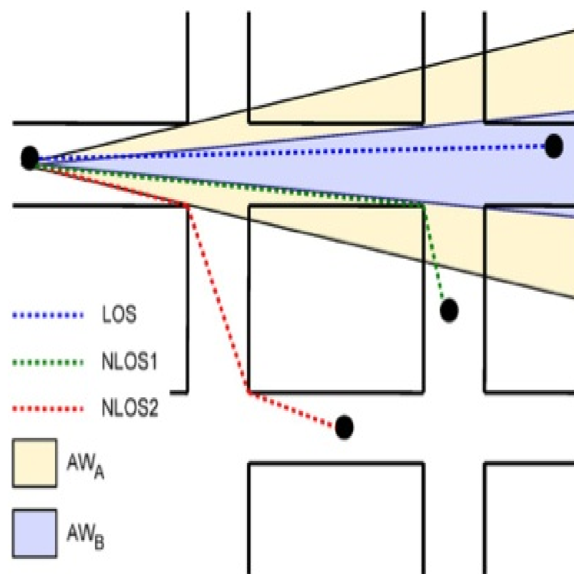


Figura 3.1: Propagazione: esempio grafico per veicoli mappati su segmenti separati tra due incroci

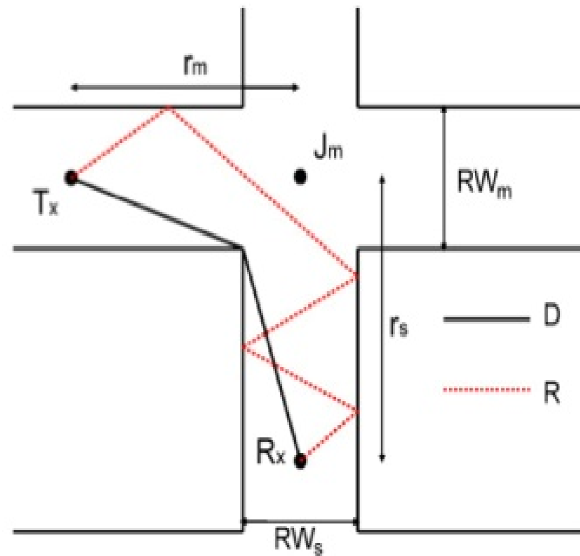


Figura 3.2: Esempio grafico della geometria nel caso NLOS1

Per classificare le coppie queste sono sottoposte a *Reverse Geocoding*; Reverse Geocoding permette di mappare la posizione sopra la topologia della rete. In base alla classificazione, viene poi applicata una formula di Path Loss (PL). Una volta classificata la situazione di propagazione, sono necessarie altre computazioni geometriche per applicare la formula. Si fa notare che per considerare altri ostacoli (quali alberi, altri veicoli. . .) si dovrebbero prendere in considerazione altri componenti. Si rende necessario calcolare una distanza geometrica per computare il PL nelle tre possibili situazioni. Tale distanza è espressa in dB. Nel caso 1 (LOS), si applica l'attenuazione dello spazio aperto, che dipende dalla lunghezza d'onda e dalla distanza tra i due veicoli, formula (2) in [13]; nel caso 2 (NLOS1), si sommano le componenti dovute alla riflessione e diffrazione dei raggi, non che alla lunghezza d'onda, alla distanza dei due veicoli dal punto di riferimento comune (solitamente il centro dell'incrocio, intersezione delle strade) e la larghezza della strada. La componente dominante è quella che è più vicino al centro dell'incrocio, formula (3) in [13]; nel caso 3 (NLOS2), è molto più complicato, perché si dovranno tenere in considerazione più fattori, tra cui

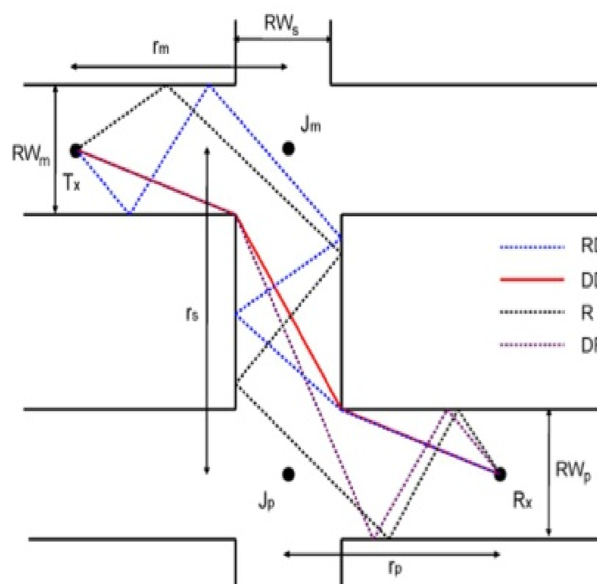


Figura 3.3: Esempio grafico della geometria nel caso NLOS2

due punti di riferimento, larghezze delle strade su cui sono mappati veicoli e punti di riferimento, distanze dei veicoli dai rispettivi punti di riferimento, riflessione e diffrazione (oltre che alla lunghezza d'onda), formula (4) in [13] . In una situazione di LOS, il segnale può propagarsi per centinaia di metri; al contrario, in presenza di angoli, il segnale degrada "nell'attraversarli". Per testare il funzionamento di CORNER sono stati fatti due tipi di esperimenti con autovetture vere: uno per valutare la connettività intorno agli angoli, coinvolgendo nodi mobili e fissi (trasmettitore fisso e ricevitore mobile); l'altro per valutare la qualità del link intorno agli angoli utilizzando posizioni fisse. Gli esperimenti sono stati condotti tutti in una zona residenziale di Los Angeles, per poi essere riprodotti con l'implementazione di CORNER e QualNet. Una cosa che è necessaria far notare è che, l'interferenza dovuta ai molteplici Acces Point (AP) non era facilmente riproducibile in simulazione. La scheda wifi scandisce tutto il canale e cerca di associarsi a quello con meno interferenze (scandisce tutta la banda e sceglie il canale con meno interferenze). Sebbene questo garantisce il minor numero di interferenze, l'alto numero di AP influisce pesantemente

sulla ricezione dei pacchetti. Per valutare la connettività intorno agli angoli sono stati effettuati esperimenti sia da fisso-a-mobile che da mobile-a-fisso. In entrambi i casi, la macchina fissa periodicamente manda dei broadcast, mentre quella mobile si salva le coordinate del punto in cui ha ricevuto il pacchetto. Nella simulazione i pacchetti ricevuti sono molti di più per via degli effetti non riproducibili. Per valutare la qualità del link viene computato il numero di pacchetti ricevuti per il numero di pacchetti inviati. Gli esperimenti sono stati condotti sia sul campo che in simulazione, poi i risultati sono stati comparati. CORNER tiene conto delle distanze e degli edifici, ma non considera l'effetto shadowing dovuti a grandi quantità di veicoli e effetti di attenuazione.

3.4 VERGILIUS

VERGILIUS [14] è un tool progettato per velocizzare e rendere più efficiente la generazione di tracce di mobilità e computazione del path loss nello studio delle reti veicolari [15]. Questo tool consente la configurazione di parametri di mobilità e di metriche di scenario al fine di abilitare un profondo e sistematico studio della relazione causale tra parametri per scenari differenti e le performance di rete. Il primo contributo introdotto da VERGILIUS è la presenza di un livello macroscopico per la generazione di pattern di movimento. Possiede un modulo per il Motion Pattern Generator. L'output del Motion Pattern Generator è adatto ad eseguire simulatori che gestiscono il livello microscopico come SUMO, consentendo così una riproduzione di una vasta gamma di tracce di mobilità per reti veicolari. VERGILIUS automatizza il processo di estrazione dei dati dalle mappe e della creazione delle tracce di mobilità.
Il secondo importante contributo fornito da VERGILIUS è la presenza di un modello che tiene in considerazione la propagazione del segnale in ambiente urbano.
Il primo passo compiuto da VERGILIUS è l'estrazione delle informazioni geografiche dal database TIGER e la sua traduzione in una descrizione della topologia delle strade. Questo output è poi utilizzato per costruire un Road Traffic Pattern che viene poi dato in ingresso a SUMO. Il Simulatore di mobilità fornisce poi una traccia che unita alle informazioni di topologia delle strade viene fornita al simulatore di rete. In questo il simulatore di

rete può fornire metriche relative al protocollo utilizzato e influenzate dalla mobilità datagli in ingresso.

3.5 Lettura delle mappe tramite TIGER

Come anticipato il database TIGER contiene informazioni circa le linee geografiche del territorio Statunitense. Le linee geografiche includono strade, ferrovie, confini di stato, linee costiere e molto altro. Ciascuna linea è divisa in segmenti, ciascuno dei quali è definito da due end-point e diverse caratteristiche. Qui l'attenzione è focalizzata sulle strade che sono classificate in diverse categorie attraverso un campo chiamato *Census Feature Class Code* (CFCC). Questa classificazione è poi utilizzata per assegnare a ciascuna strada il limite di velocità e un numero di corsie. TIGER non fornisce informazioni sulle strade a senso unico, quindi tutte le strade sono considerate a doppio senso.

3.6 Generatore di scenario

Prima di descrivere il generatore di scenario è necessario introdurre il concetto di *Road Traffic Pattern* (RTP). Un Road Traffic Pattern è un insieme di parametri che insieme definiscono un comportamento a livello macroscopico del traffico stradale su una certa topologia stradale. Un RTP è fondamentalmente un set di input di cui il simulatore di mobilità ha bisogno al fine di costruire la traccia di mobilità. Può essere quindi visto come un insieme di primitive che definiscono in maniera univoca il comportamento del traffico a livello macroscopico a prescindere dal simulatore che viene usato per creare la traccia di mobilità reale. Le primitive consistono in:

1. *lettura della topologia*: la rete è descritta come un grafo diretto;

2. *flussi di ingresso*: il numero di veicoli per unità di tempo che entrano da ogni segmento di ingresso (ad esempio le strade ai bordi della mappa);

3. *flussi interni*: il numero di veicoli per unità di tempo che attraversano ogni segmento interno alla mappa;

4. *probabilità di svolta*: per ogni incrocio, la frazione di veicoli che transita da una strada di ingresso ad una qualunque strada di uscita;

5. *matrice di origine destinazione*: numero di veicoli che partono da una qualunque strada di ingresso ad una qualunque strada di uscita.

SUMO potrebbe essere eseguito con solamente i punti (1, 2, 4) oppure i punti (1, 5). Per avere una descrizione unica bisogna comunque includerli tutti quanti.
Il Generatore di Scenario adottato permette di generare pattern di traffico in certo modo, consentendo di ottenere i pattern di output manipolando solo relativamente pochi parametri di input. Il percorso ha sempre origine da un punto di ingresso ad uno di uscita. I punti di ingresso e uscita sono segmenti di strada ai bordi della mappa. Per ogni ingresso sono si definisce il numero di percorsi che partono da lì e terminano nelle loro destinazioni.

Numero di percorsi: l'utente definisce l'aggregato del flusso totale di ingresso. Questo flusso aggregato di ingresso sarà poi distribuito tra tutti gli ingressi basato sull'importanza del segmento di strada su cui si trova l'ingresso. Ad ogni classe di strada viene assegnato un peso incrementale basato sul limite di velocità. Strade con limite di velocità maggiore avranno peso maggiore. Viene poi di conseguenza assegnato un numero di percorsi.

Scelta della destinazione: una destinazione deve essere scelta per ogni ingresso. Tutti i percorsi terminano in un punto di uscita. Per ogni ingresso viene creata una lista di possibili destinazioni che vengono ordinate in maniera decrescente per distanza geometrica dall'ingresso. La scelta può essere effettuata in maniera deterministica (scegliendo sempre la destinazione più lontana dal punto di ingresso) o random.

Costruzione del percorso: una volata che origine e destinazione sono state scelte è necessario costruire un percorso effettivo, cioè la sequenza di segmenti che un veicolo attraverserà effettivamente per andare dall'origine alla destinazione. La costruzione viene fatta computando il percorso tramite l'algoritmo di Dijkstra, il quale può essere eseguito usando come peso o la distanza geometrica o il tempo, che induce ad usare il percorso più breve in un caso o il più veloce nell'altro, rispettivamente.

Per i dettagli si rimanda a [14] .

3.7 Simulatore di rete: NS-3 + ndnSIM

Come simulatore di rete è stato utilizzato NS-3 [6]; in particolare, è stato utilizzato una modulo per NS-3, *ndnSIM*, ovvero un simulatore di NDN per NS-3. NS-3 necessita di strumenti flessibili per valutare aspetti del modello di comunicazione. Facendo riferimento a [16], ndnSIM fornisce una implementazione di NDN consentendo una valutazione di diversi aspetti di comunicazione di NDN.
NS-3 funge da framework di simulazione ed ndnSIM si pone i seguenti obietti:

- essere open source per venire incontro alle esigenze della comunità di ricerca per gli studi sulla simulazione;
- essere in grado di simulare fedelmente le operazioni di base di NDN;
- essere in grado di supportare una larga scala di simulazioni con 1000 e più nodi;
- agevolare gli esperimenti con il caching dei dati, l'inoltro dei pacchetti, il routing e la gestione della congestione;

Seguendo l'architettura di NDN, ndnSIM è implementato come modello di un nuovo protocollo di livello di rete, che può essere eseguito sopra qualunque modello di protocollo di livello due attualmente disponibile (point-to-point, CSMA, wireless, etc.), oltre che sui protocolli di rete attuali IPv4 ed IPv6 e sui protocolli di livello di trasporto quali TCP e UDP. Questa flessibilità consente di simulare vari scenari di reti omogenee ed eterogenee (ad esempio, solo NDN, NDN-over-IP, etc). Il simulatore è implementato in maniera modulare, tramite il linguaggio C++. Ogni classe C++ modella il comportamento di ogni entità di livello di rete di NDN: Pending Interest table (PIT), Forwarding Information Base (FIB), Content Store (CS), interfacce di rete e applicazione, strategia di forwarding per gli Interest, etc. Questo approccio modulare consente lo sviluppo, la modifica, la sostituzione di ogni componente in maniera agevole con impatto nullo o minimo sugli altri componenti. In più, il simulatore fornisce una collezione

di interfacce ed helpers che consentono di eseguire un tracing dettagliato sul comportamento di ogni componente, oltre che sul flusso di traffico di NDN. Documentazione e codice sono disponibili presso [17].
NS-3 è un simulatore relativamente nuovo e non ha ancora tutto quello che hanno i simulatori commerciali (come ad esempio Qualnet) o il suo predecessore NS-2, ma offre comunque una architettura di progetto chiara, una estensiva documentazione, e flessibilità di implementazione.

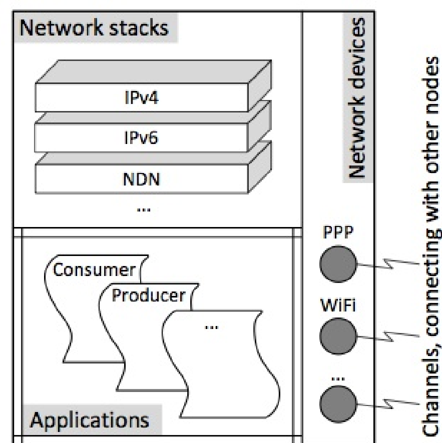


Figura 3.4: Astrazione di ndnSIM in NS-3

Similmente agli stack di protocollo esistenti come IPv4 e IPv6, ndnSIM è stato progettato come una stack di protocollo indipendente che può essere installato su un nodo di rete, si veda figura 3.4. Oltre allo stack protocollare, ndnSIM include anche un certo numero di applicazioni che generano traffico ed helpers che agevolano la creazione di scenari di simulazione (installando lo stack e le applicazioni sui nodi) e tool per ottenere statistiche sulle simulazioni per ragioni di misure. In figura 3.5 è riportato il diagramma a blocchi dei componenti di ndnSIM, il cui dettaglio è disponibile in [16] . L'astrazione della comunicazione tra livelli è invece visibile in figura 3.6 .

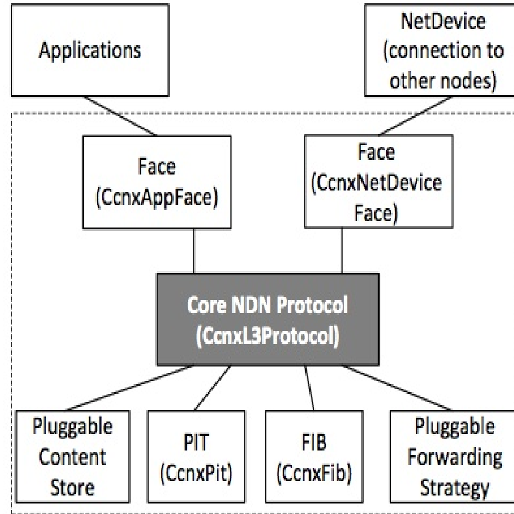


Figura 3.5: Componenti di ndnSIM

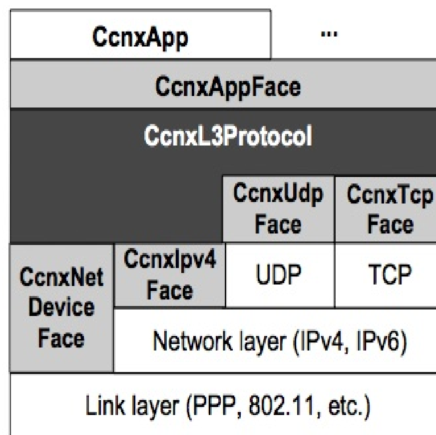


Figura 3.6: Astrazione della comunicazione tra layer in ndnSIM

3.8 Mettere insieme gli strumenti, dalla configurazione alla mobilità

In questo paragrafo saranno illustrati i passi necessari per ottenere la mobilità, creando una catena tra gli strumenti utilizzati.
Il primo passo consiste nel creare il file di configurazione di VERGILIUS. VERGILIUS ha diverse modalità di funzionamento, ma quelle oggetto di interesse saranno la modalità 1 (indispensabile per creare tutti file necessari a creare la mobilità) e la modalità 7 (quella che creerà la mobilità). Si crea quindi un file con estensione *.config*, all'interno del quale vanno specificati i seguenti parametri:

- *MODE*: come anticipato, VERGILIUS ha diverse modalità di funzionamento; al primo step, è necessario specificare 1
- *ORIGIN*: rappresenta dove sarà centrata la mappa e sarà estratta da TIGER; è necessario specificare latitudine e longitudine in notazione decimale e moltiplicato per 1 000 000 (un milione)
- *WIDTH*: larghezza della mappa in metri
- *HEIGHT*: altezza della mappa in metri (unita alla larghezza determina le dimensioni della mappa che sarà estratta)
- *INTERNAL_NODES*: questo parametro non è ben noto in quanto non documentato; è possibile specificare YES oppure NO, ma diversi test preliminari effettuati hanno dimostrato che è meglio specificare YES
- *RT1*: nome del file TIGER con estensione RT1; è una parte del file dal quale saranno estratte le informazioni; se viene specificato il full path del file, è necessario metterlo tra doppi apici
- *RT2*: nome del file TIGER con estensione RT2; è l'altra parte del file dal quale saranno estratte le informazioni; anche qui, se viene specificato il full path del file, è necessario metterlo tra apici doppi
- *BUILD_SUMO_OUTPUT*: impostato a YES, genera i file che saranno dati in ingresso a SUMO
- *SUMO_SIMULATION_TIME*: tempo di simulazione in secondi

- *SUMO_NODES_FILE*: qui è necessario specificare il nome del file in cui verrà scritta la configurazione dei nodi; è il file XML generato in output in formato leggibile da SUMO; se viene specificato il full path, questi va inserito tra doppi apici
- *SUMO_EDGES_FILE*: qui è necessario specificare il nome del file in cui verrà scritta la configurazione dei segmenti; è il file XML generato in output in formato leggibile da SUMO; se viene specificato il full path, questi va inserito tra doppi apici
- *SUMO_TURNS_FILE*: qui è necessario specificare il nome del file in cui verrà scritta la configurazione delle svolte; è il file XML generato in output in formato leggibile da SUMO; se viene specificato il full path, questi va inserito tra doppi apici
- *SUMO_FLOWS_FILE*: qui è necessario specificare il nome del file in cui verrà scritta la configurazione dei flussi; è il file XML generato in output in formato leggibile da SUMO; se viene specificato il full path, questi va inserito tra doppi apici
- *SCENARIO*: il tipo di scenario da costruire
- *AVERAGE_ARRIVALS*: numero medio di macchine all'ora che entrano nella mappa

I parametri sopra elencati sono obbligatori. Altri parametri dipendono dallo scenario scelto. Ad esempio, scegliere *RANDOM* come SCENARIO (che significa che tutte le vie avranno destinazioni scelte in maniera casuale), sarà necessario specificare anche i seguenti parametri:

- *SCENARIO_DIJKSTRA_WEIGHT*: il peso può essere *TIME* oppure *DISTANCE*
- *RANDOM_SCENARIO_DESTINATION_TYPE*: questo valore specifica dove si vogliono mandare le vetture (più il valore è alto, più i veicoli tenderanno andare nella stesa strada)
- *ENTRY_FLOW_MODE*: rappresenta come vengono distribuiti i flussi lungo la mappa

- *WEIGHTED_FLOWSEXPONENT*: rappresenta l'impatto della differenza di pesi (più è alto il valore, più il traffico sarà lungo le strade grandi)

Un esempio di file di configurazione è visibile di seguito

```
MODE            1
ORIGIN   34024725        -118491458
WIDTH    2100
HEIGHT   2100
INTERNAL_NODES  No
RT1      "/your/fulll/path/TGR06037.RT1"
RT2      "/your/fulll/path/TGR06037.RT2"
BUILD_SUMO_OUTPUT       YES
SUMO_SIMULATION_TIME    1800
SUMO_NODES_FILE SUMOnodes.xml
SUMO_EDGES_FILE SUMOedges.xml
SUMO_TURNS_FILE SUMOturns.xml
SUMO_FLOWS_FILE         SUMOflows.xml
SCENARIO        RANDOM
SCENARIO_DIJKSTRA_WEIGHT                TIME
RANDOM_SCENARIO_DESTINATION_TYPE         RANDOM 4
AVERAGE_ARRIVALS        95
ENTRY_FLOW_MODE WEIGHTED
WEIGHTED_FLOWS_EXPONENT 1
```

Una volta creato il file, il secondo passo consiste nell'esecuzione di VERGILIUS tramite il seguente comando:

mono VMF.exe /path/to/file_name.config

Il file con estensione .config è quello creato in precedenza. Il risultato di questo comando è la creazione dei seguenti file nella cartella corrente:
Data.txt links.data Links.txt mode1.config nodes.data Nodes.txt output.bmp SUMO_Config.cfg SUMOedges.xml SUMOflows.xml SUMOnodes.xml SUMOturns.xml
Il passo successivo consiste nell'usare due nuovi tool: *netconvert* [18] e *jtrrouter* [19]. Il tool netconvert consente di importare reti stradali digitali da diverse origini, e generare reti stradali che possono essere usati da altri tool, nel caso in oggetto, SUMO. Il tool jtrrouter si occupa invece di computare le rotte che possono essere utilizzate da SUMO, usando una diverse definizioni e probabilità di svolta agli incroci. I comandi sono dati nel seguente ordine:

netconvert -n SUMOnodes.xml **-e** SUMOedges.xml **-o** SUMO.net.xml

jtrrouter -n SUMO.net.xml **-f** SUMOflows.xml **-t** SUMOturns.xml -o SUMO.rou.xml

Tramite l'opzione *-n* di netconvert si specifica da quale file leggere la definizione dei nodi, mentre l'opzione *-e* specifica da quale file leggere la definizione dei segmenti. L'opzione *-o* specifica invece in quale file verrà scritto l'output del comando. L'opzione *-n* di jtrrouter specifica quale file SUMO usare come rete stradale sul quale eseguire i percorsi, l'opzione *-f* specifica da quale file leggere la definizione dei flussi, mentre l'opzione *-t* specifica da quale file leggere le percentuali relative alle svolte. L'opzione *-o* specifica invece in quale file verrà scritto l'output del comando.
Il prossimo passo consiste nell'editare il file SUMO_Config.cfg; tale file va editato specificando il path dei file SUMOnet.xml e SUMOroutes.xml, nel modo seguente:

```
<configuration>
        <input
                net-file="/your/full/path/SUMOnet.xml"
                route-files="/your/full/path/SUMOroutes.xml"
                begin="0"
                end="endTime"
        />
</configuration>
```

L'attributo *end* sarà automaticamente settato ai valore specificato nel file di configurazione di VERGILIUS.
Ora viene utilizzato SUMO tramite il seguente comando:

sumo -c SUMO_Config.cfg **-n** SUMOnet.xml **-r** SUMOroutes.xml **–netstate-dump** sumoOutput.xml

L'opzione *-c* specifica che deve essere letto il file di configurazione, l'opzione *-n* specifica da quale file leggere la definizione della rete stradale, l'opzione *-r* da quale file leggere la definizione delle rotte, mentre l'opzione *–netstate-dump* in quale file andare a scrivere l'output generato.
A questo punto è necessario creare un altro file di configurazione, sempre con estensione *.config* . Si devono specificare i seguenti parametri:

- *MODE*: modalità di utilizzo di VERGILIUS, in questo step è 7

- *SUMO_INPUT_NODES_FILE*: è il full path del SUMO file che contiene la configurazione dei nodi
- *SUMO_INPUT_EDGES_FILE*: è il full path del SUMO file che contiene la configurazione dei segmenti
- *SUMO_INPUT_NETWORK_FILE*: è il full path del SUMO file che contiene la configurazione della rete stradale
- *SUMO_INPUT_TRACE_FILE*: è il full path del SUMO file ottenuto al punto precedente, cioè sumoOutput.xml

Un esempio di file di configurazione è il seguente:

```
MODE                    7
SUMO_INPUT_NODES_FILE   "/your/full/path/SUMOnodes.xml"
SUMO_INPUT_EDGES_FILE   "/your/full/path/SUMOedges.xml"
SUMO_INPUT_NETWORK_FILE "/your/full/path/SUMOnet.xml"
SUMO_INPUT_TRACE_FILE   "/your/full/path/sumoOutput.xml"
```

Una volta creato il file è necessario eseguire il comando:

- **mono** VMF.exe /path/file_name2.config

Il risultato di questo comando è la creazione del file di mobilità, con estensione *.mobility* .

Capitolo 4

Scenari di Simulazione

In questo parte viene illustrato quali scenari di comunicazione sono stati implementati e con quali caratteristiche. I file di configurazioni creati per le simulazioni sono stati scritti facendo riferimento alle linee guida dei capitoli precedenti. Tutti i file ai quali si farà riferimento sono visibili in appendice.

4.1 Casi d'uso

L'idea è quella di preparare quattro scenari di mobilità con caratteristiche diverse, in particolare:

1. mobilità con 8000 veicoli all'ora entranti nella mappa, con un peso pari a 0
2. mobilità con 16000 veicoli all'ora entranti nella mappa, con un peso pari a 0
3. mobilità con 8000 veicoli all'ora entranti nella mappa, con un peso pari a 3
4. mobilità con 16000 veicoli all'ora entranti nella mappa, con un peso pari a 3

Sono state create mobilità diverse con l'idea di eseguire in futuro i medesimi test sui diversi casi. Le simulazioni effettuate negli scenari individuati sono invece state condotte sulla medesima mobilità, ovvero la numero uno.
I primi due scenari individuano un caso in cui i pesi sono uniformemente

distribuiti, mentre il terzo e il quarto danno peso maggiore alle strade più ampie. Vengono quindi preparati quattro file di configurazione diversi, uno per ogni mobilità. Per realizzare la mappa sono state prese le TIGER maps relative alla zona di Santa Monica (California); le mappe sono reperibili al sito dello U.S. Department of Commerce, United States Census Bureau [20] . Le mappe vengono centrate in un punto della Santa Monica Blvd, in una mappa di dimensione (in metri) 2100 x 2100. Il nome del primo file di configurazione è *mode1.config*, quello del secondo è *mode7.config* . I numeri all'interno del nome sono relativi alla modalità che è necessario attivare. Una volta generata la mobilità, è necessario anche generare la mappa che dovrà essere letta da CORNER.
Una volta ottenute le mappe come mostrato nei passi elencati precedentemente, è necessario portare la traccia in un formato leggibile dal simulatore. Per fare ciò, si ricorre ad una funzione di utilità, la quale non fa altro che portare la mobilità generata da SUMO dal formato

id time (x, y, z)

al formato

id time x y z

Il primo elemento, *id*, è un identificatore univoco del veicolo; *time* è l'istante temporale; x, y e z rappresentano le coordinate Universal Transvers Mercator (UTM) relative alla mappa di riferimento ad ogni istante temporale. Una volta ottenuta la mappa in formato leggibile dal simulatore, è necessario sapere quante macchine sono presenti, cioè quante macchine sono state generate. Per sapere quante macchine vi sono, è possibile usare un'altra funzione di utilità. Per il primo scenario sono presenti un totale di 7884 macchine; per il secondo 15676; per il terzo 7926, per il quarto 15546 . Una volta generata la mobilità è possibile utilizzare il tool grafico *sumo-gui* [21], per poter vedere l'evoluzione della mobilità. Per poter utilizzare sumo-gui è necessario creare un apposito file di configurazione che deve avere estensione *.sumo.cfg* . Questo file è simile al file *.cfg* generatosi nel processo di creazione della mobilità, con la differenza che vi sono i file con la estensione richiesta da sumo-gui per poter visualizzare la mobilità generata, e il contenuto del file non presenta l'informazione sulla durata della simulazione. Una volta creato il file, è necessario eseguire sumo-gui, impostare la modalità *real world* dal menu a tendina, dove di default vi è impostato

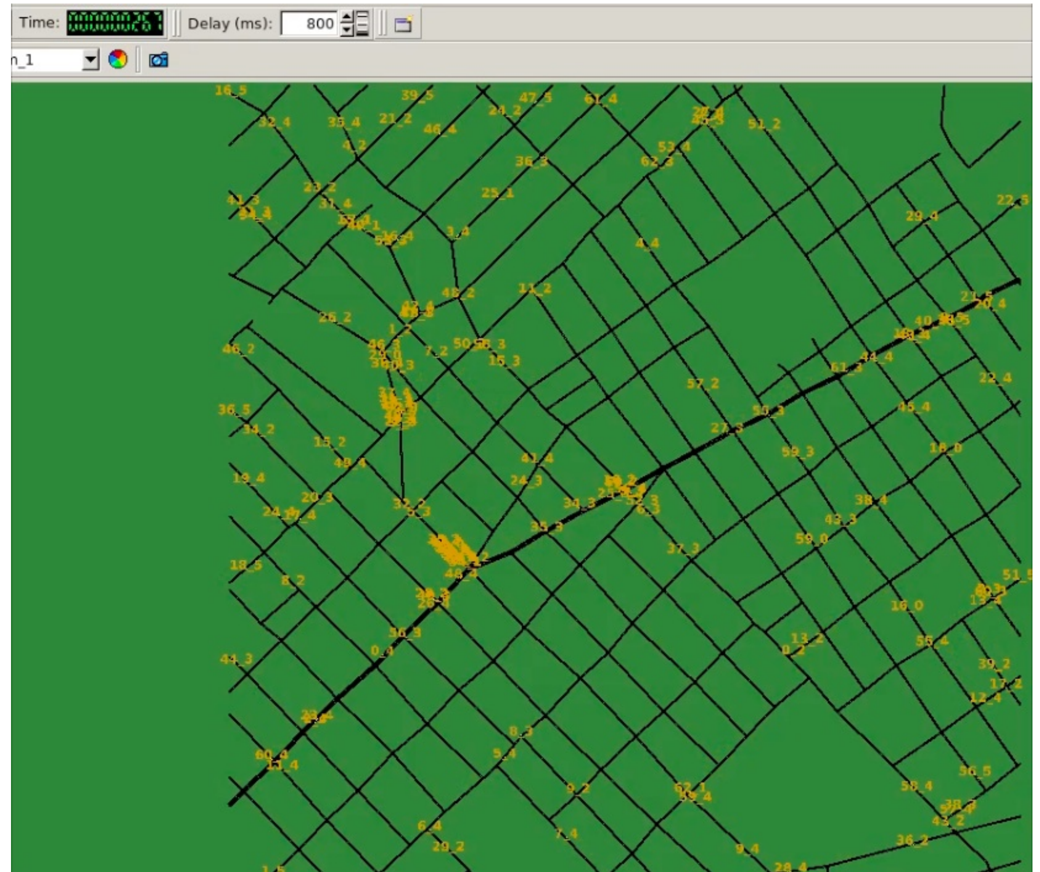


Figura 4.1: SUMO-GUI

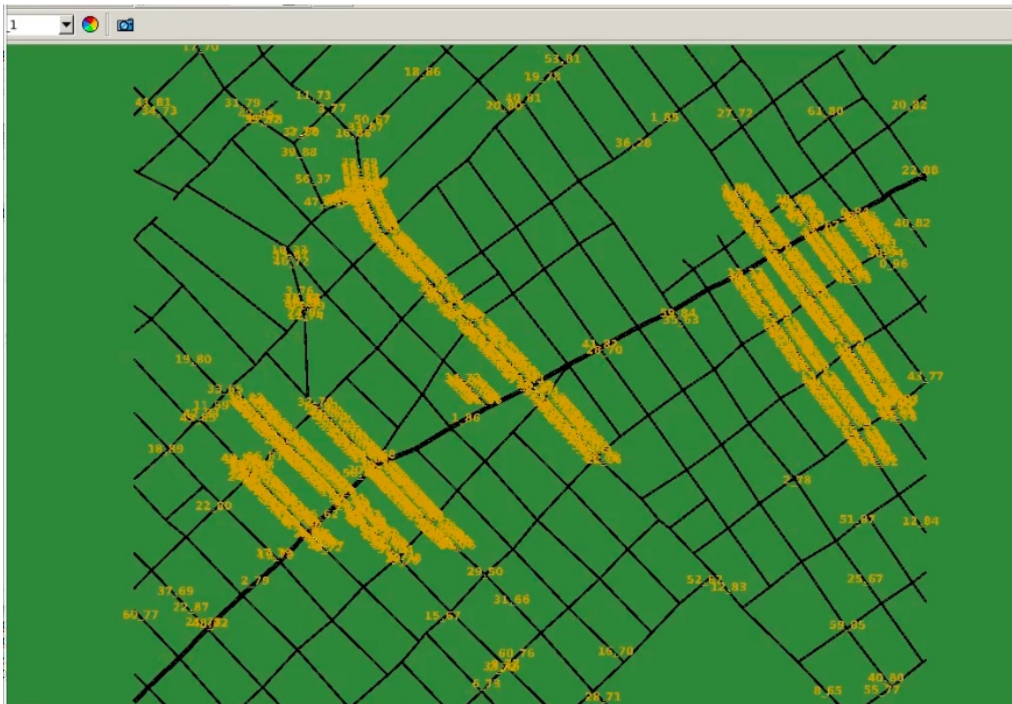


Figura 4.2: SUMO-GUI: evoluzione simulazione

standard; è necessario impostare un delay di un certo numero di secondi, altrimenti sarà impossibile visualizzare la simulazione. Per una migliore visualizzazione, si consiglia di fare click sul pulsante in cui compare un cerchio colorato (tasto a fianco il menu a tendina), che apre il *View Settings*. Da qui, *Vehicles* dal menu a sinistra, selezionare l'opzione *Show vehicle name*, in premere *Use* per rendere effettive le modifiche. In questo modo i veicoli saranno più visibili. Si fa notare che è presente anche una funzionalità che permette di estrarre una porzione di mobilità, specificando un il numero di secondi che si vogliono considerare. Utilizzando questa funzionalità, ovvero prendendo solo una porzione di mobilità, cambia il numero di veicoli presenti in simulazione. Estraendo una porzione di mobilità pari a 600 secondi, si ottiene il primo scenario un numero di veicoli pari a 690; per il secondo scenario1341 , per il terzo scenario 696; per il quarto scenario 1316. Nelle figure 4.1 e 4.2 sono mostrate due immagini del tool SUMO-gui.
Il consiglio sull'uso delle funzionalità disponibili è il seguente: utilizzare innanzi tutto la funzionalità che porta la mobilità dal formato SUMO a quello per il simulatore di rete; poi, scegliere se si vuole estrarre una porzione di mobilità a partire dall'istante temporale 0, oppure estrarre una porzione di mobilità specificando da quale istante temporale partire e la grandezza dell'intervallo temporale. Una volta scelta la porzione di mobilità, per i test a Producer e Consumer fermi è necessario aggiungere alla mobilità ottenuta in precedenza i veicoli statici e per farlo si può utilizzare l'apposita funzionalità. La posizione sarà poi mantenuta la medesima per tutta la durata della simulazione; i veicoli sono stati disposti più o meno casualmente sulla mappa, e in maniera tale per cui Producer e Consumer non possano comunicare direttamente.
Gli scenari pianificati sono i seguenti:

- Singolo Produttore Singolo Consumatore;
- Singolo Produttore Multi Consumatore;
- Multi Produttore Singolo Consumatore;
- Multi Produttore Multi Consumatore;

Per ciascun scenario sono state eseguite simulazioni sia con Producer e Consumer fermi, sia con Producer e Consumer in movimento. A causa degli elevati tempi simulazione e del carico computazione generato, si è scelto di

implementare i casi Multi Consumatore e Multi Produttore con tre Consumer e quattro Producer in un caso, e quattro Consumer e tre Producer nell'altro. Nei casi in cui Producer e Consumer sono fermi, è necessario usare la funzione di utilità che consente di aggiungere alla mappa creata in precedenza identificatore e posizioni dei veicoli, che dovranno rimanere invariati per tutta la durata della simulazione. Tutti gli altri veicoli saranno invece in movimento. Nei due casi in cui tutti i veicoli sono mobili, è sufficiente scegliere quali veicoli saranno Producer e quali Consumer. Per i parametri di configurazione della simulazione si rimanda all'Appendice.

4.2 Analisi dei dati e risultati

All'esecuzione di una simulazione devono essere attivati i *LOG* relativi ai moduli sui quali si vogliono raccogliere informazioni. Questi log non sono altro che stampe re-dirette su file. I log attivati sono relativi a Consumer, Producer, facce dei device, forwarding, FIB, PIT, CS, wifi a livello fisico e di singola faccia.
Per studiare gli indicatori di performance si è reso necessario implementare uno script per l'analisi del log generato dalla simulazione; questa fase si articola in due parti:

i. si estrae dal log solamente le informazioni che sono oggetto di interesse e si scrivono su un nuovo file di log che sarà l'input della fase successiva;

ii. si analizza e si elaborano le informazioni estrapolate al fine di ottenere informazioni utili sugli indici di performance.

Come indici di performance sono stati considerati:

- numero di hop che attraversa un Interest (funzione di distribuzione cumulativa);
- tempo che intercorre da quando il contenuto viene richiesto a quando arriva (valori e funzione di distribuzione cumulativa);
- numero di hop che attraversa un Content (valori e funzione di distribuzione cumulativa);
- numero di ritrasmissioni degli Interest (valori);

Il calcolo della distribuzione cumulativa sul numero di hop fatto dagli Interest è basato sul percorso minimo, calcolato con l'algoritmo di Dijkstra. Altri aspetti che sono stati presi in considerazione sono la percentuale di richieste che sono state soddisfatte, e la percentuale di chi ha soddisfatto queste richieste. Infatti, si distinguono tre casi:

1. richiesta soddisfatta del Producer: il Producer emette il contento a fronte della ricezione di una richiesta da parte di un Consumer, ed esiste un percorso diretto che, attraverso uno o più salti, porta il contenuto a chi aveva emesso la richiesta;

2. richiesta soddisfatta dal Mule: generata a partire da una ritrasmissione (il contenuto non era arrivato a chi lo aveva richiesto e viene espresso nuovamente la richiesta), il Content viene emesso da un nodo intermedio che aveva precedentemente ricevuto il contenuto e lo avevo memorizzato nella cache;

3. richiesta auto-soddisfatta: il Consumer esprime un Interest per un contenuto che ha già nella propria cache.

Il terzo caso si verifica a fronte delle seguente situazione: l'applicazione dopo un certo timer cancella la entry della richiesta dalla PIT del Consumer che ha espresso la richiesta. Una volta che l'ha cancellata, il Consumer potrebbe ricevere il contenuto, quindi, quando decide di esprimere nuovamente la richiesta, la entry nella PIT non c'è più, ma il contenuto in cache si. Questo genera un conteggio degli hop fatti dall'Interest e dal Content pari a zero, ma il tempo impiegato a soddisfare la richiesta relativa a quell'Interest è diverso da zero.
Le successive sezioni sono dedicate ai risultati ottenuti dai singoli scenari.

4.2.1 Singolo Produttore Singolo Consumatore - Scenario 1

I grafici relativi agli Interest sono riportati in figura 4.3 e 4.4, e riportano la distribuzione cumulativa degli hop fatti dagli Interest e del numero di ritrasmissioni di ciascun prefisso, rispettivamente. Nelle figure 4.5 e 4.6 si riportano il numero di hop attraversati dai Content e la relativa distribuzione cumulativa, rispettivamente. Infine, nelle figure 4.7 e 4.8 sono riportati i

tempi impiegati ad ottenere i prefissi e la relativa distribuzione cumulativa, rispettivamente.

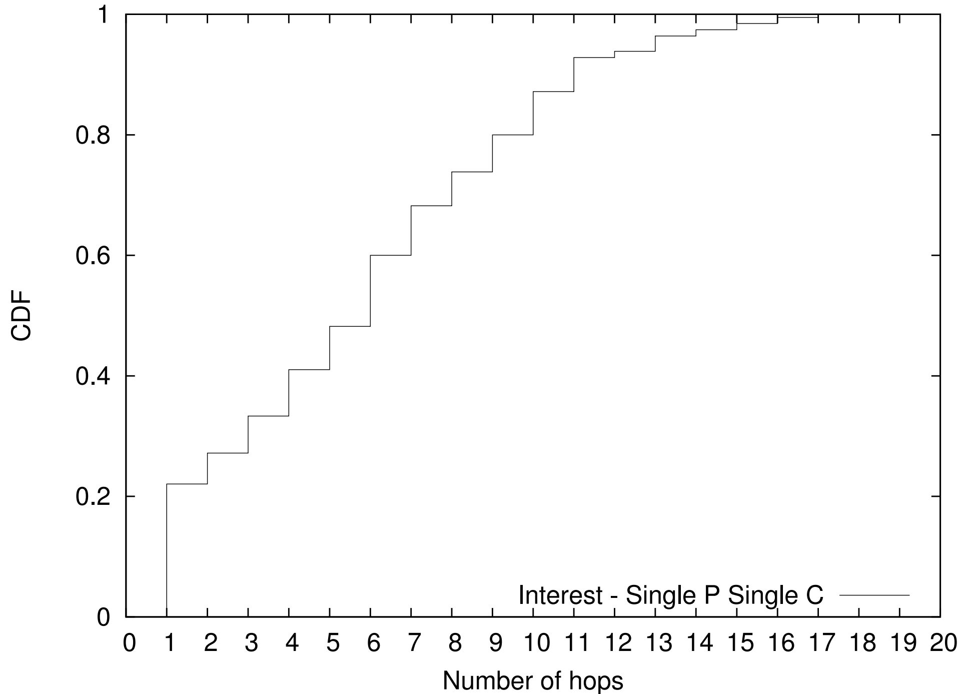


Figura 4.3: Single P Single C - Scenario 1: hop attraversati (cumulativa)

In questo scenario sono stati richiesti un numero di prefissi distinti pari a 132; di questi 132 ne sono stati soddisfatti 107, cioè sono stati emessi 107 contenuti che prima o poi sono giunti al Consumer che li aveva richiesti, determinando una percentuale del 81,06% di richieste soddisfatte. Di queste, il 93,46% è stato soddisfatto dalla cache, il restante 6,54% dal Producer.

4.2.2 Singolo Produttore Singolo Consumatore - Scenario 2

I grafici relativi agli Interest sono riportati in figura 4.9 e 4.10, e riportano la distribuzione cumulativa degli hop fatti dagli Interest e del numero di ritrasmissioni di ciascun prefisso, rispettivamente. Nelle figure 4.11 e 4.12

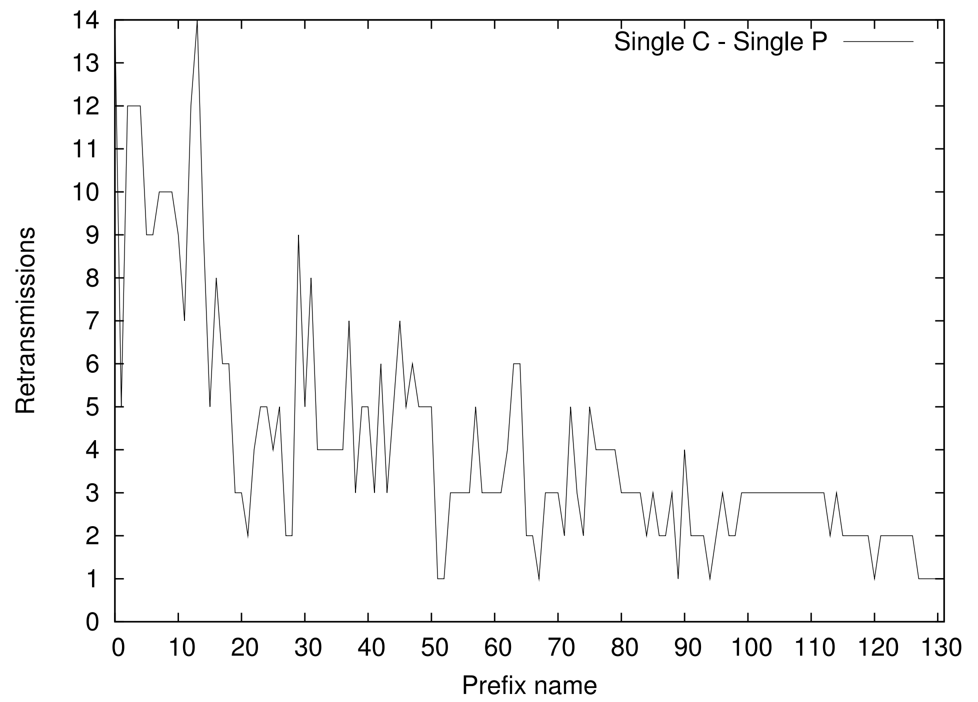


Figura 4.4: Single P Single C - Scenario 1: numero ritrasmissioni

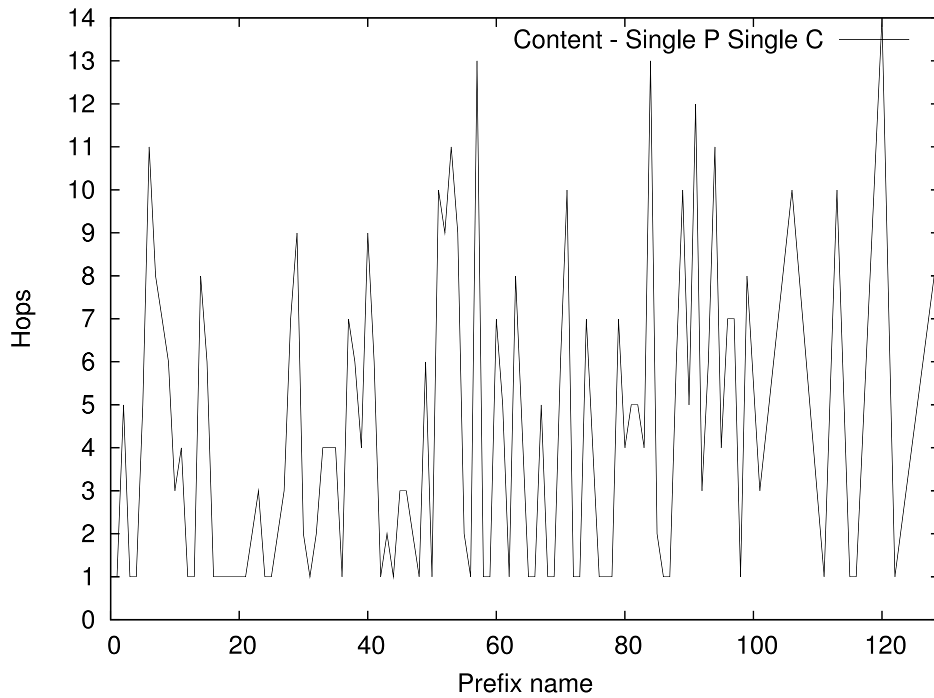


Figura 4.5: Single P Single C - Scenario 1: hop attraversati

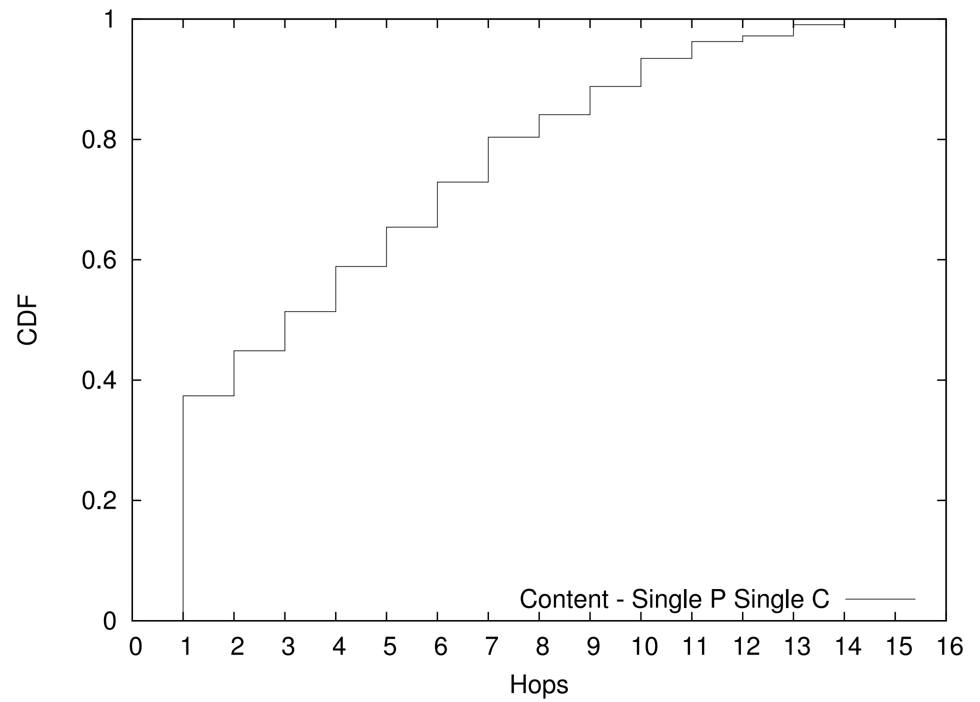


Figura 4.6: Single P Single C - Scenario 1: hop attraversati (cumulativa)

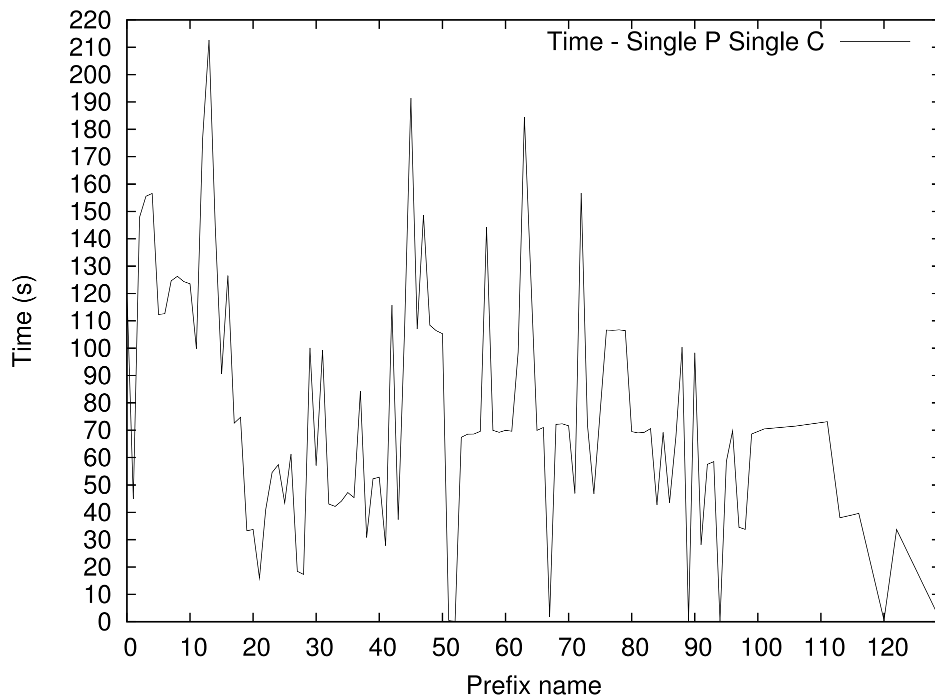


Figura 4.7: Single P Single C - Scenario 1: tempo andata e ritorno

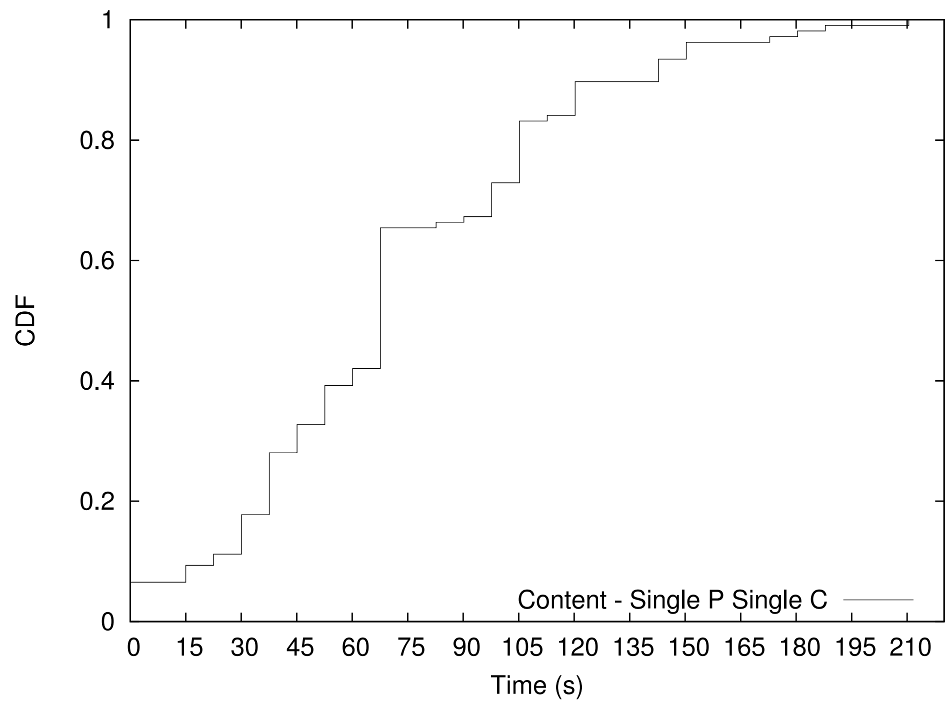


Figura 4.8: Single P Single C - Scenario 1: tempo andata e ritorno (cumulativa)

si riportano il numero di hop attraversati dai Content e la relativa distribuzione cumulativa, rispettivamente. Infine, nelle figure 4.13 e 4.14 sono riportati i tempi impiegati ad ottenere i prefissi e la relativa distribuzione cumulativa, rispettivamente.

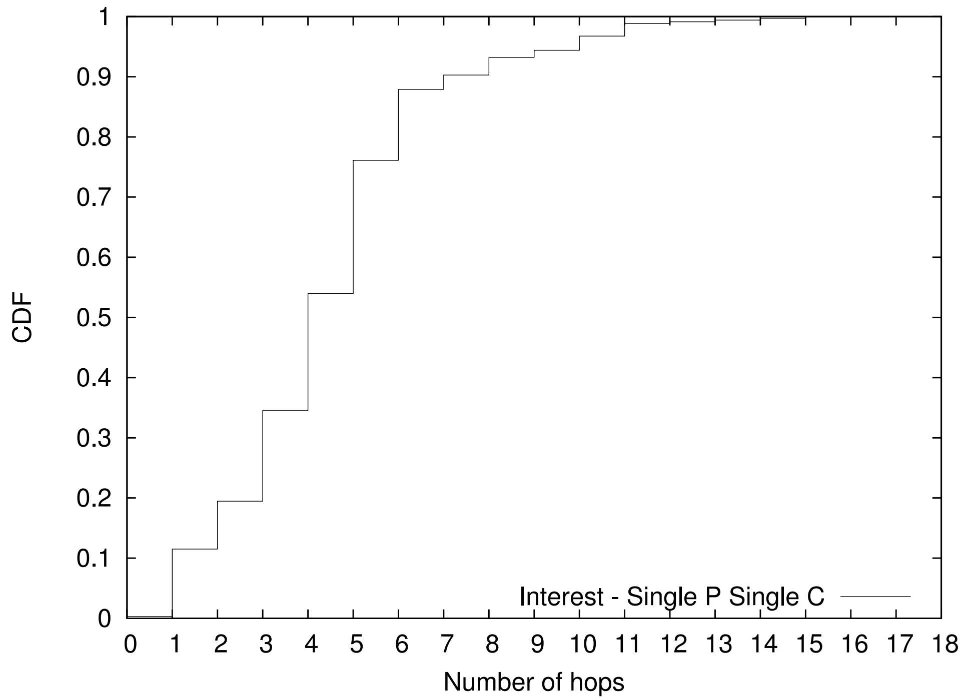


Figura 4.9: Single P Single C - Scenario 2: hop attraversati (cumulativa)

In questo scenario sono stati richiesti un numero di prefissi distinti pari a 241; di questi 241 ne sono stati soddisfatti 236, cioè sono stati emessi 236 contenuti che prima o poi sono giunti al Consumer che li aveva richiesti, determinando una percentuale del 97,93% di richieste soddisfatte. Di queste, il 70,34% è stato soddisfatto dalla cache, il restante 29,66% dal Producer.

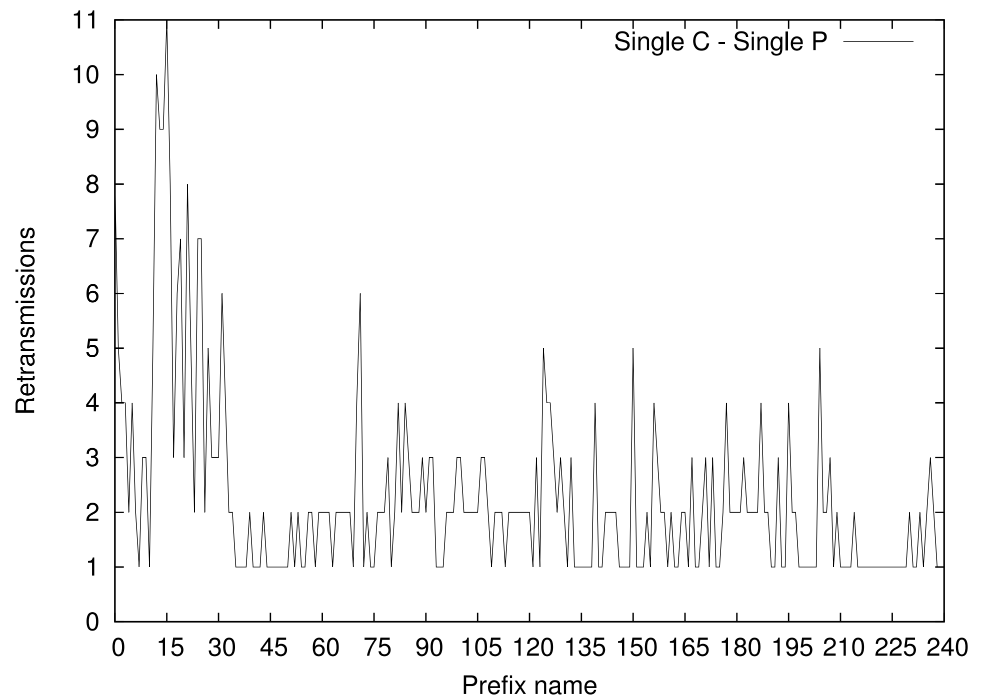


Figura 4.10: Single P Single C - Scenario 2: numero ritrasmissioni

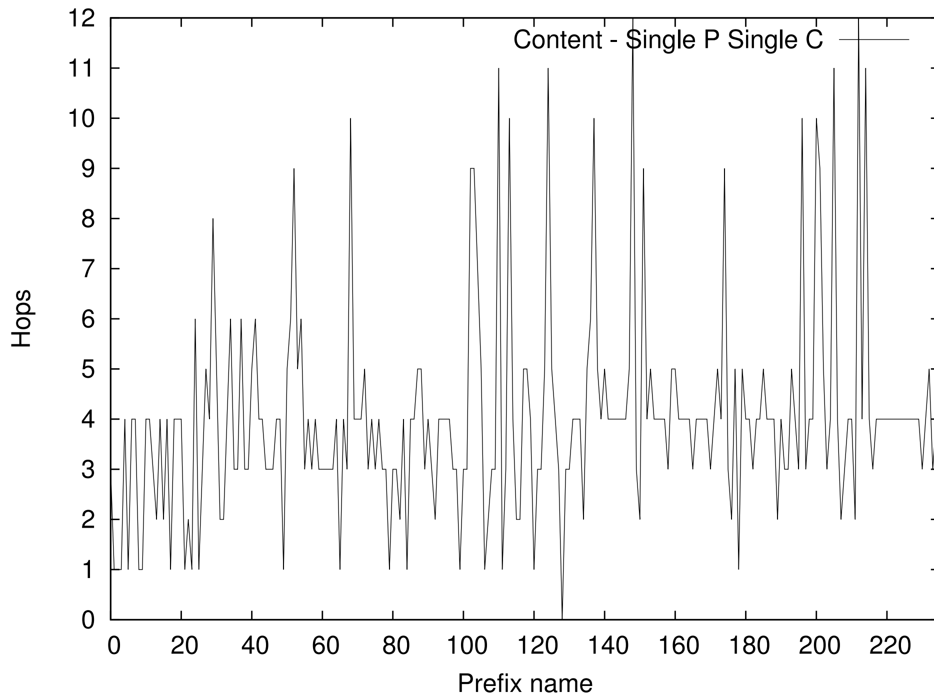


Figura 4.11: Single P Single C - Scenario 2: hop attraversati

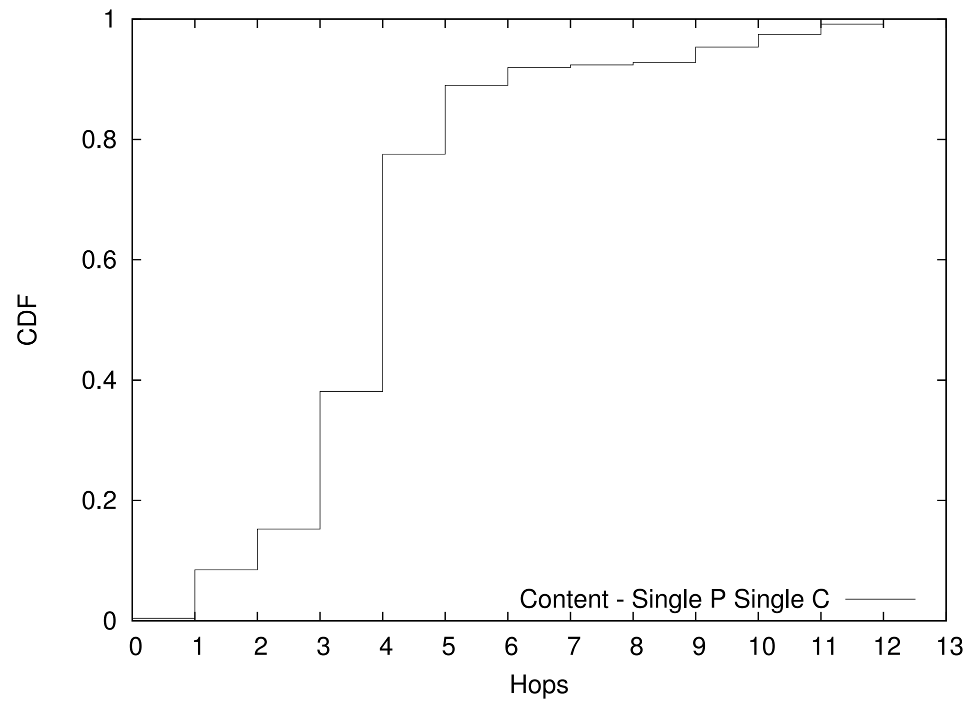


Figura 4.12: Single P Single C - Scenario 2: hop attraversati (cumulativa)

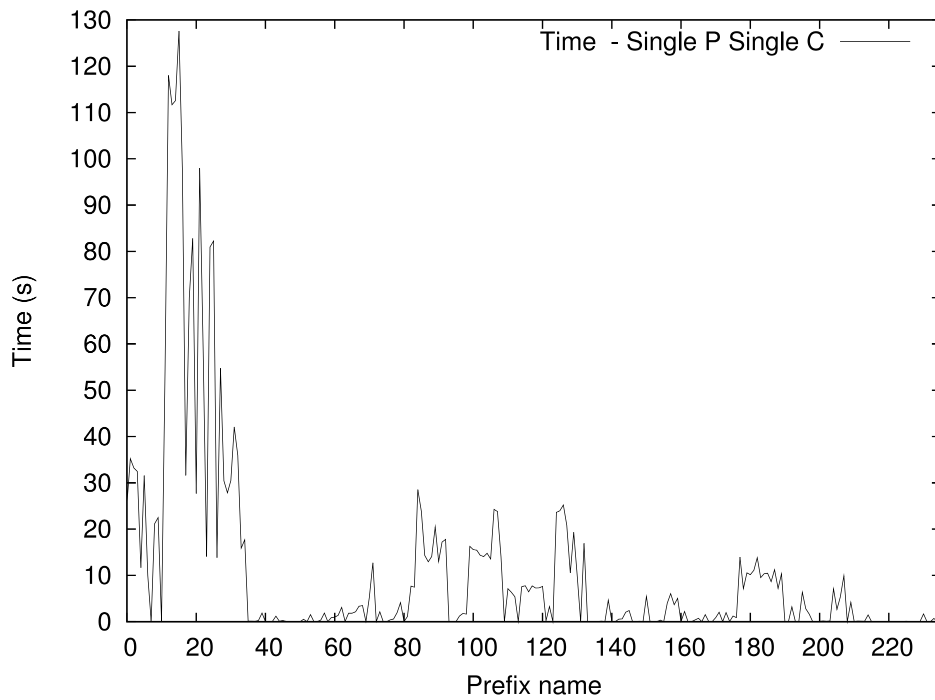


Figura 4.13: Single P Single C - Scenario 2: tempo andata e ritorno

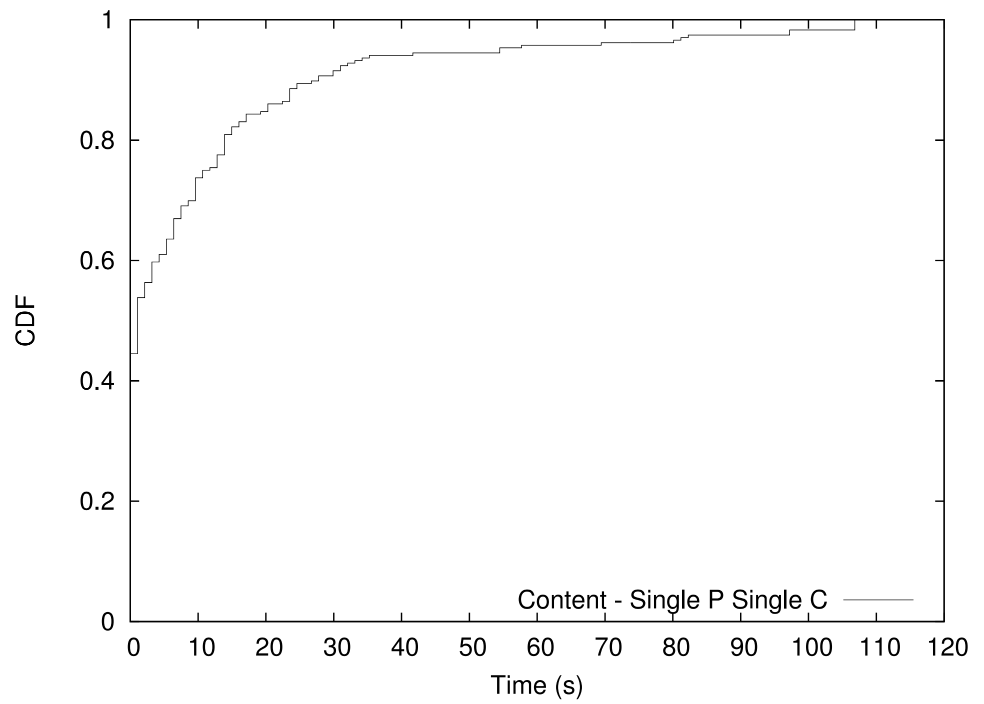


Figura 4.14: Single P Single C - Scenario 2: tempo andata e ritorno (cumulativa)

4.2.3 Singolo Produttore Singolo Consumatore - Scenario 3

I grafici relativi agli Interest sono riportati in figura 4.15 e 4.16, e riportano la distribuzione cumulativa degli hop fatti dagli Interest e del numero di ritrasmissioni di ciascun prefisso, rispettivamente. Nelle figure 4.17 e 4.18 si riportano il numero di hop attraversati dai Content e la relativa distribuzione cumulativa, rispettivamente. Infine, nelle figure 4.19 e 4.20 sono riportati i tempi impiegati ad ottenere i prefissi e la relativa distribuzione cumulativa, rispettivamente.

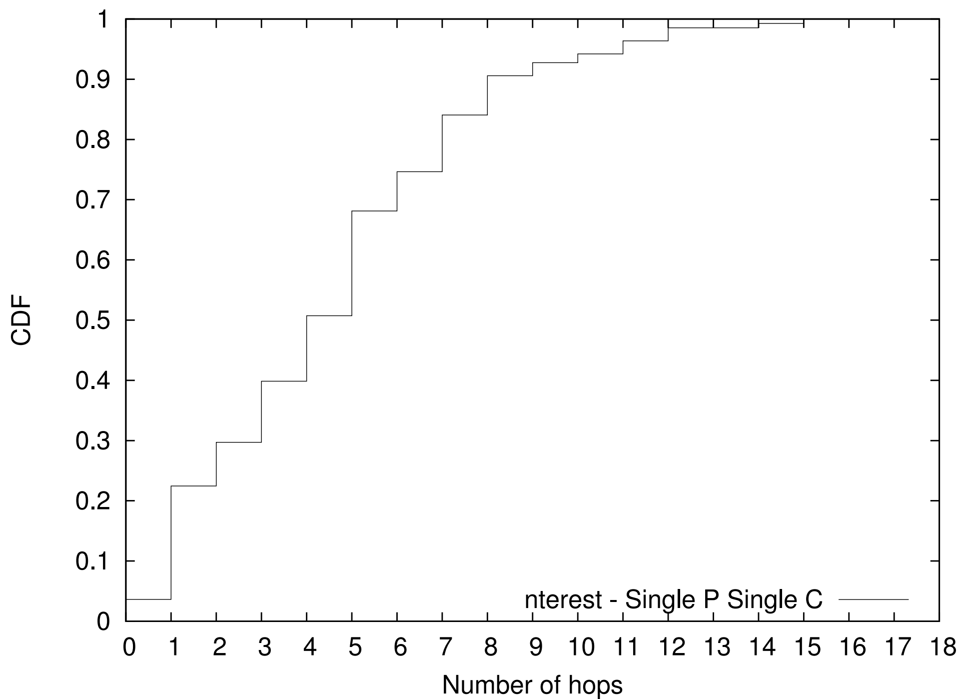


Figura 4.15: Single P Single C - Scenario 3: hop attraversati (cumulativa)

In questo scenario sono stati richiesti un numero di prefissi distinti pari a 88; di questi 88 ne sono stati soddisfatti 75, cioè sono stati emessi 75 contenuti che prima o poi sono giunti al Consumer che li aveva richiesti, de-

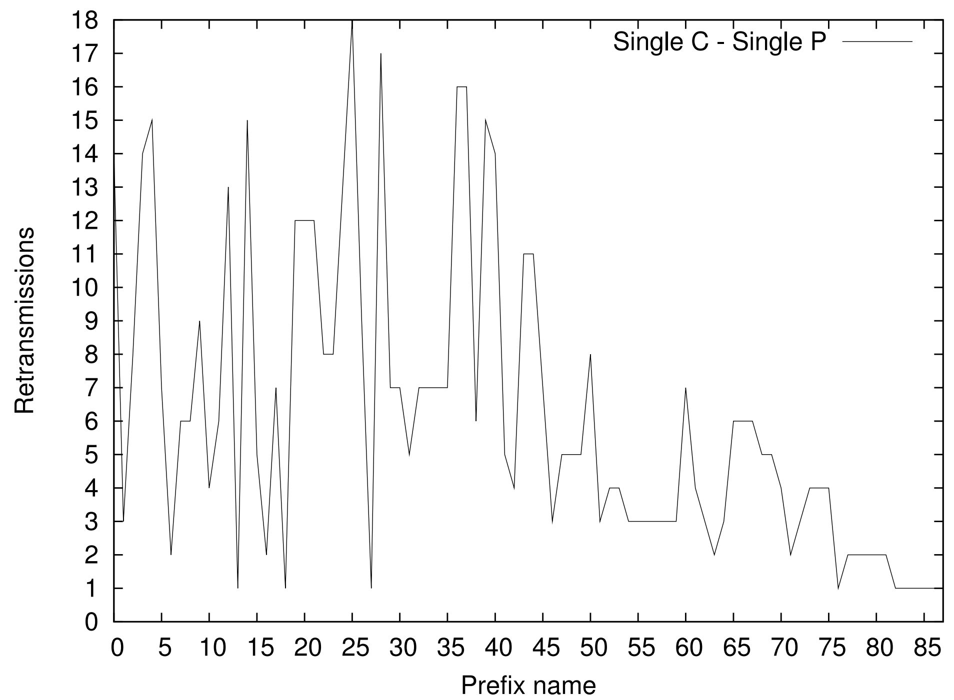


Figura 4.16: Single P Single C - Scenario 3: numero ritrasmissioni

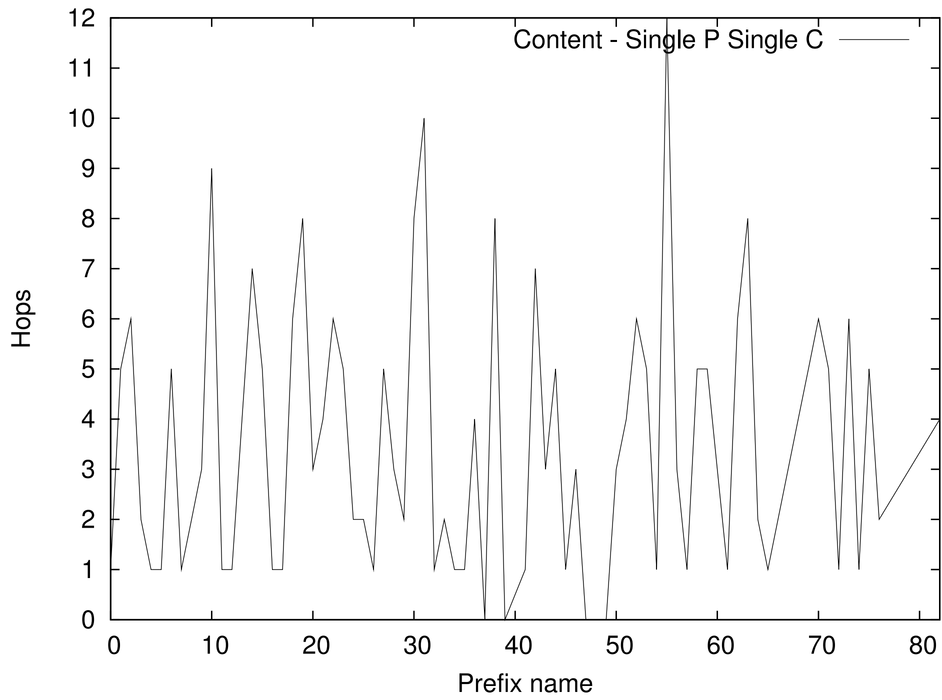


Figura 4.17: Single P Single C - Scenario 3: hop attraversati

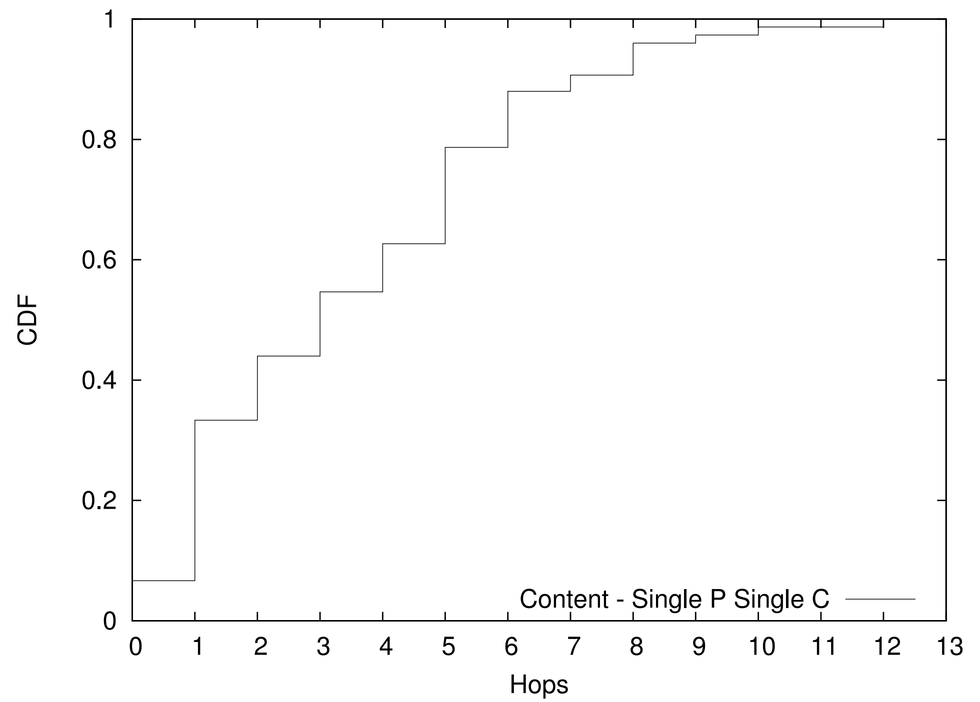


Figura 4.18: Single P Single C - Scenario 3: hop attraversati (cumulativa)

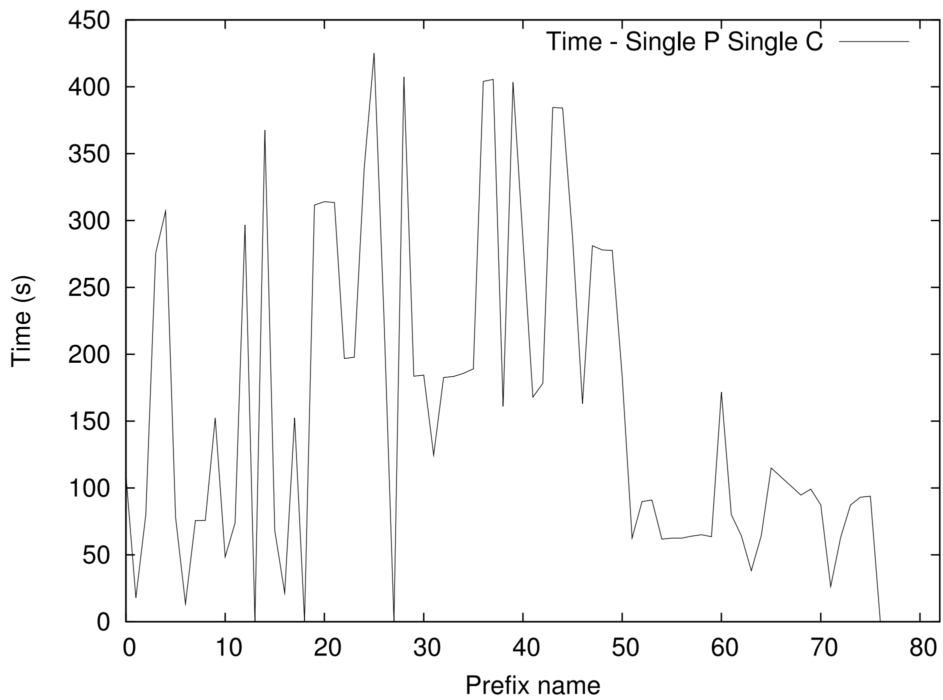


Figura 4.19: Single P Single C - Scenario 3: tempo andata e ritorno

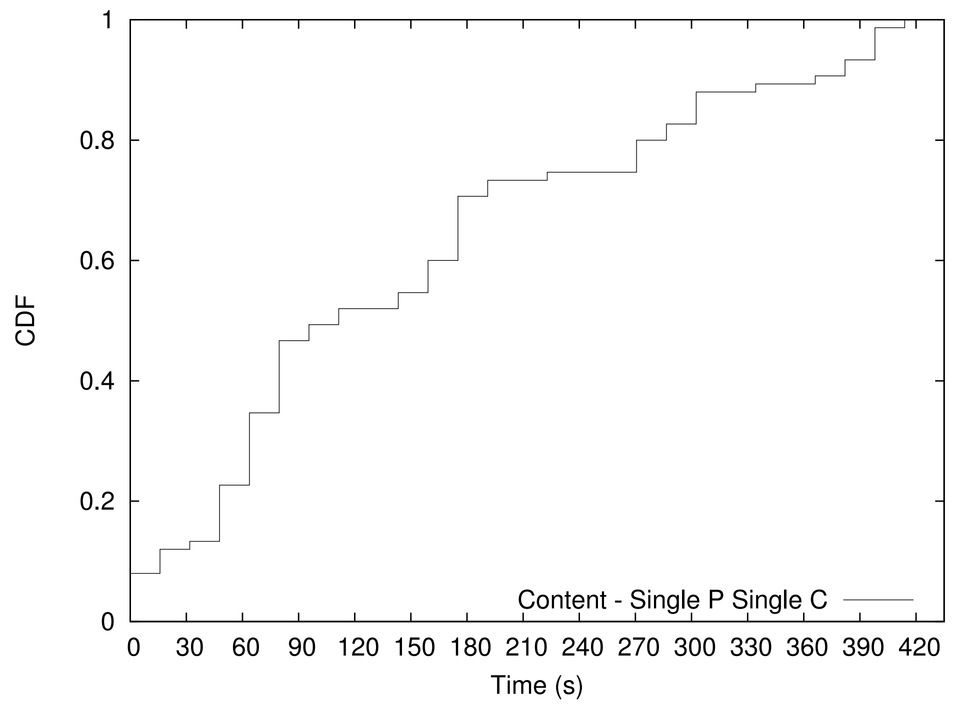


Figura 4.20: Single P Single C - Scenario 3: tempo andata e ritorno (cumulativa)

terminando una percentuale del 85,23% di richieste soddisfatte. Di queste, il 74,67% è stato soddisfatto dalla cache, il restante 25,33% dal Producer.

4.2.4 Singolo Produttore Multi Consumatore - Scenario 1

I grafici relativi agli Interest sono riportati in figura 4.21 e 4.22, e riportano la distribuzione cumulativa degli hop fatti dagli Interest e del numero di ritrasmissioni di ciascun prefisso, rispettivamente. Nelle figure 4.23 e 4.24 si riportano il numero di hop attraversati dai Content e la relativa distribuzione cumulativa, rispettivamente. Infine, nelle figure 4.25 e 4.26 sono riportati i tempi impiegati ad ottenere i prefissi e la relativa distribuzione cumulativa, rispettivamente.

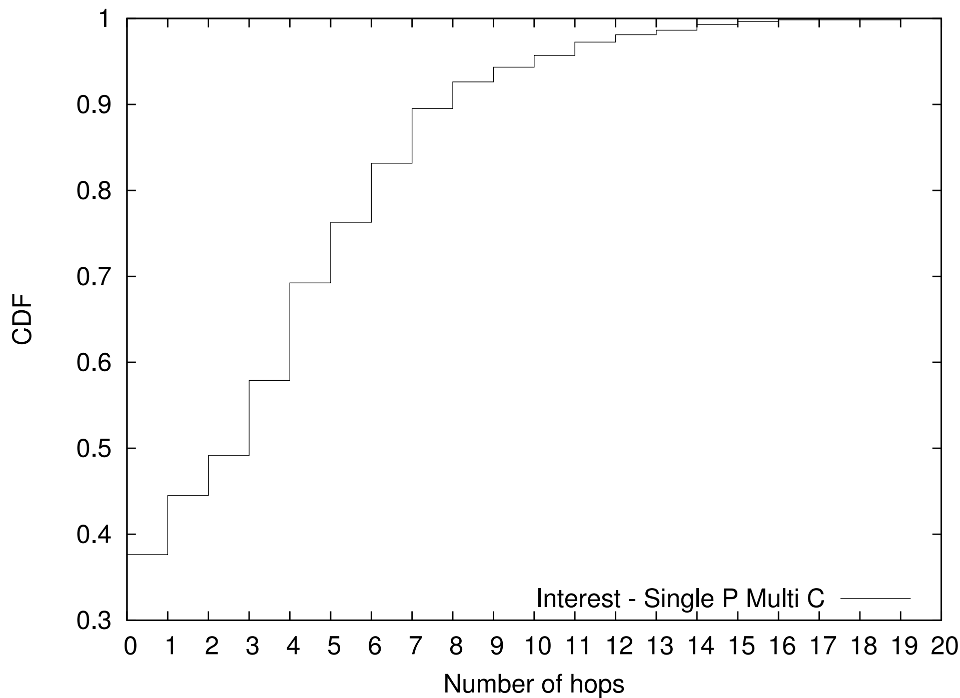


Figura 4.21: Single P Multi C - Scenario 1: hop attraversati (cumulativa)

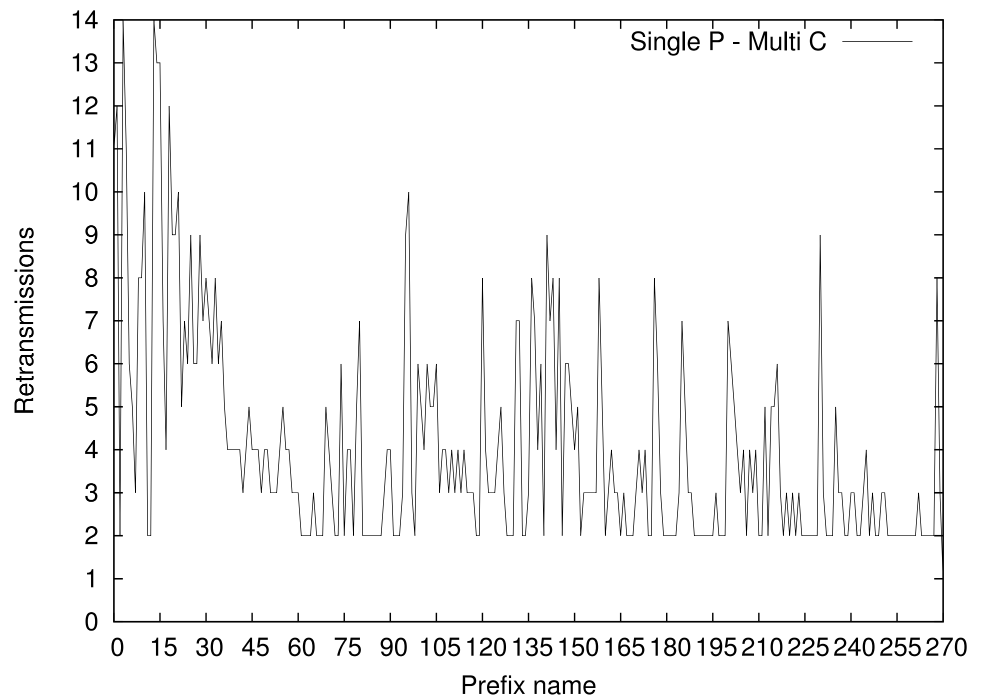


Figura 4.22: Single P Multi C - Scenario 1: numero ritrasmissioni

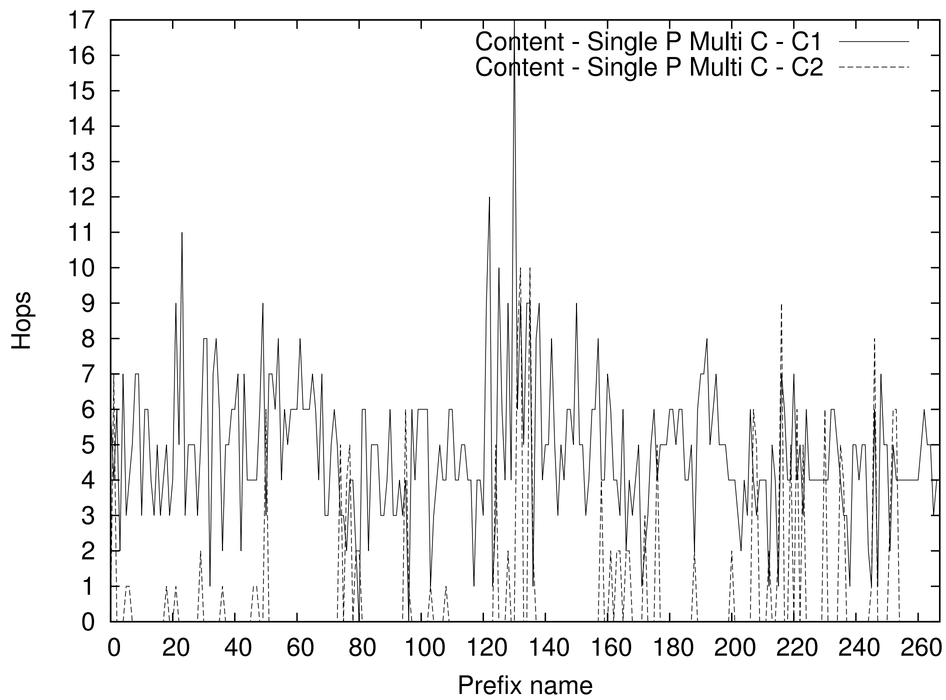


Figura 4.23: Single P Multi C - Scenario 1: hop attraversati

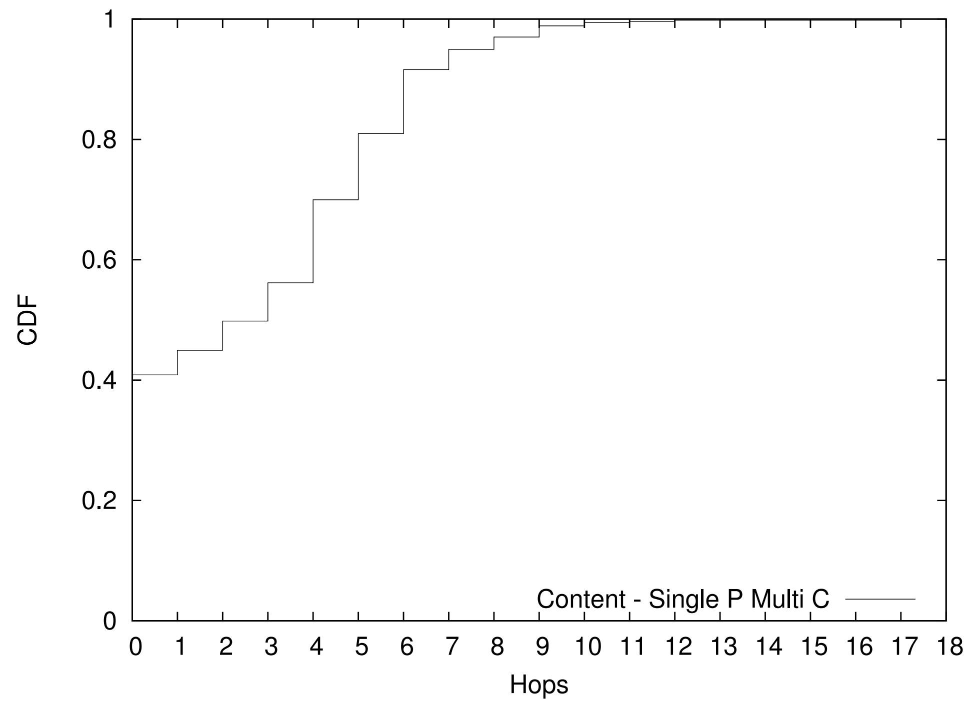


Figura 4.24: Single P Multi C - Scenario 1: hop attraversati (cumulativa)

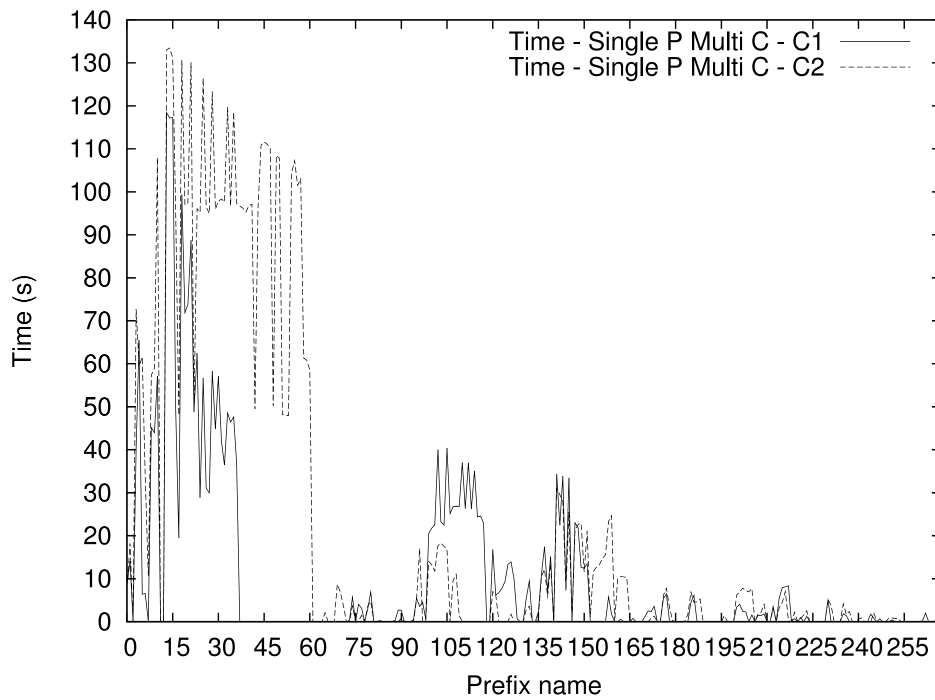


Figura 4.25: Single P Multi C - Scenario 1: tempo andata e ritorno

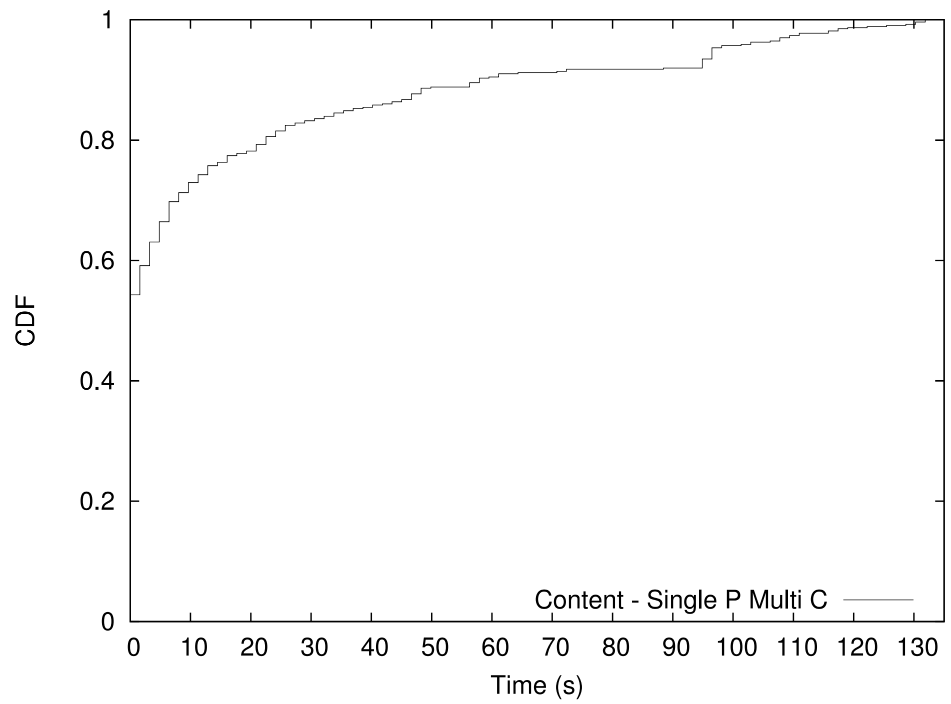


Figura 4.26: Single P Multi C - Scenario 1: tempo andata e ritorno (cumulativa)

In questo scenario sono stati richiesti un numero di prefissi distinti pari a 271; qui però sono presenti due Consumer, quindi si considerano un numero superiori di richieste, se si considera che i due Consumer possono avere richiesto i medesimi prefissi. Perciò in totale le richieste distinte tra i due Consumer sono 541. Di questi 541 ne sono stati soddisfatti 536, cioè sono stati emessi 536 contenuti che prima o poi sono giunti ciascuno al Consumer che li aveva richiesti, determinando una percentuale del 99,08% di richieste soddisfatte. Di queste, il 73,51% è stato soddisfatto dalla cache, il restante 26,49% dal Producer.

4.2.5 Singolo Produttore Multi Consumatore - Scenario 2

I grafici relativi agli Interest sono riportati in figura 4.27 e 4.28, e riportano la distribuzione cumulativa degli hop fatti dagli Interest e del numero di ritrasmissioni di ciascun prefisso, rispettivamente. Nelle figure 4.29 e 4.30 si riportano il numero di hop attraversati dai Content e la relativa distribuzione cumulativa, rispettivamente. Infine, nelle figure 4.31 e 4.32 sono riportati i tempi impiegati ad ottenere i prefissi e la relativa distribuzione cumulativa, rispettivamente.

In questo scenario sono stati richiesti un numero di prefissi distinti pari a 245; qui però sono presenti due Consumer, quindi si considerano un numero superiori di richieste, se si considera che i due Consumer possono avere richiesto i medesimi prefissi. Perciò in totale le richieste distinte tra i due Consumer sono 478. Di questi 478 ne sono stati soddisfatti 474, cioè sono stati emessi 474 contenuti che prima o poi sono giunti ciascuno al Consumer che li aveva richiesti, determinando una percentuale del 99,16% di richieste soddisfatte. Di queste, il 89,24% è stato soddisfatto dalla cache, il restante 10,76% dal Producer.

4.2.6 Multi Produttore Singolo Consumatore - Scenario 1

I grafici relativi agli Interest sono riportati in figura 4.33 e 4.34, e riportano la distribuzione cumulativa degli hop fatti dagli Interest e del numero di ritrasmissioni di ciascun prefisso, rispettivamente. Nelle figure 4.35 e 4.36

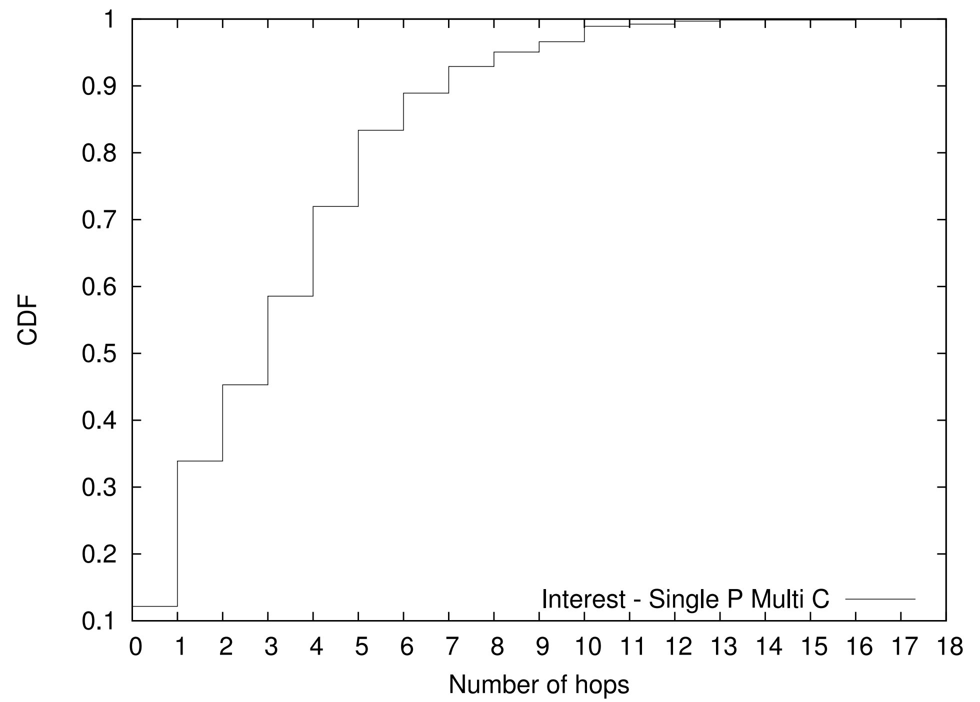


Figura 4.27: Single P Multi C - Scenario 2: hop attraversati (cumulativa)

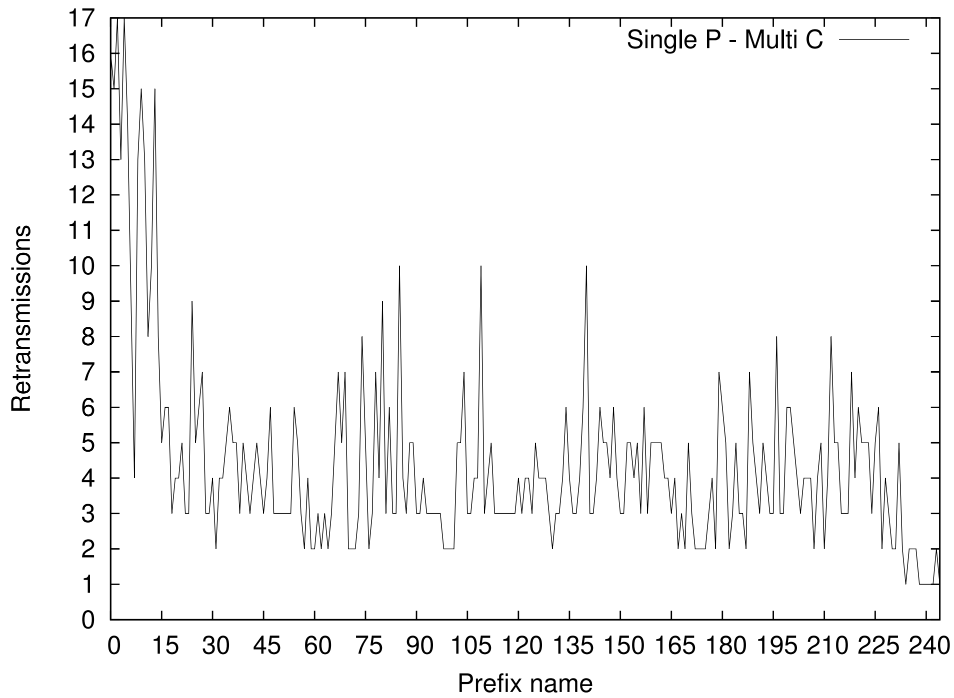


Figura 4.28: Single P Multi C - Scenario 2: numero ritrasmissioni

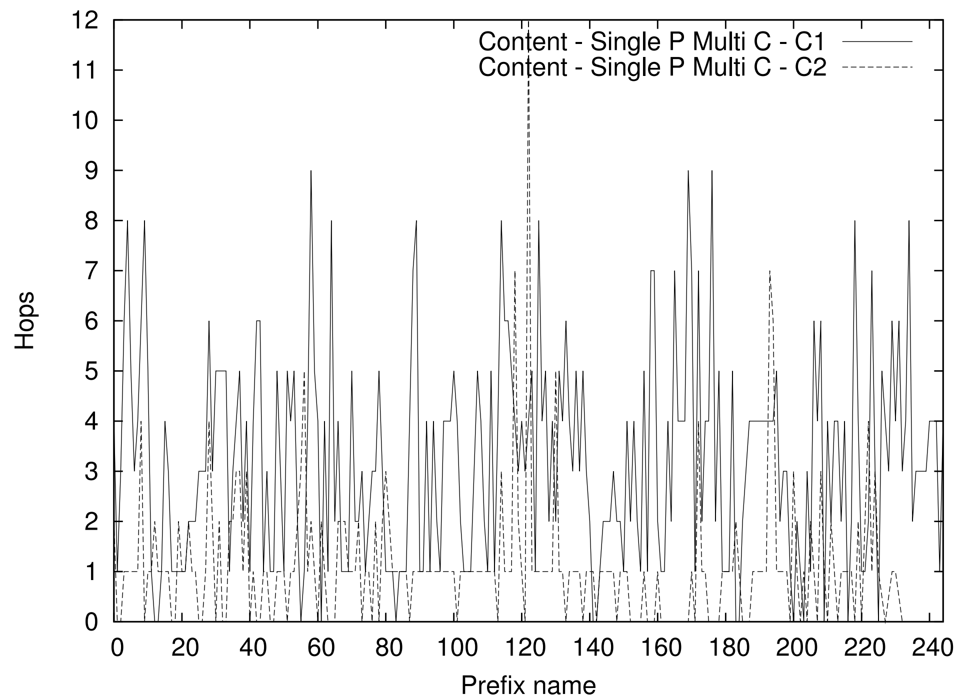


Figura 4.29: Single P Multi C - Scenario 2: hop attraversati

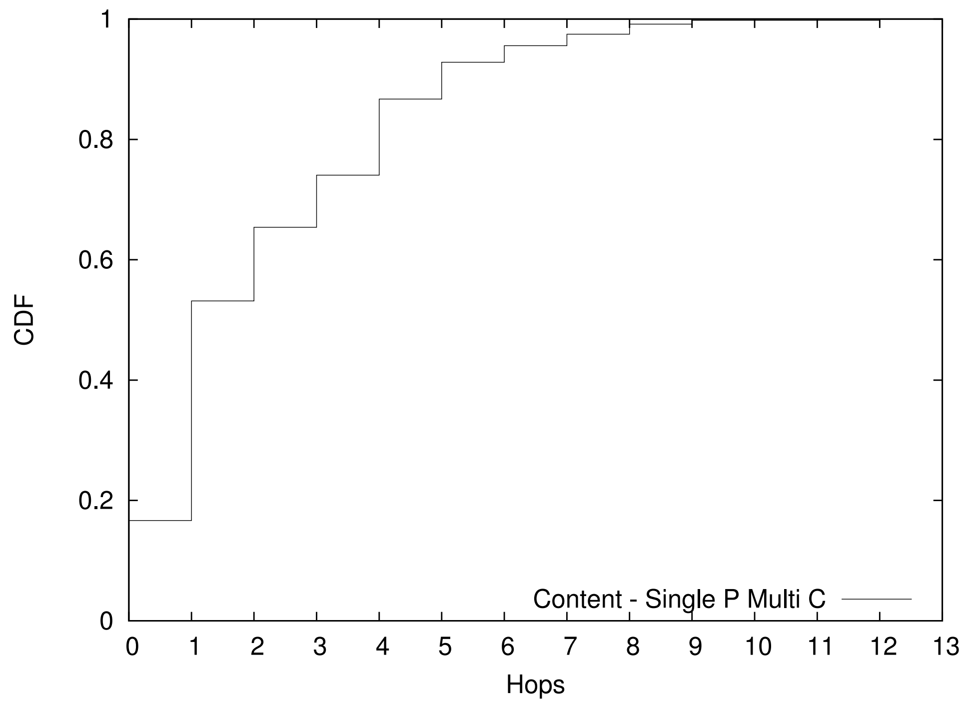


Figura 4.30: Single P Multi C - Scenario 2: hop attraversati (cumulativa)

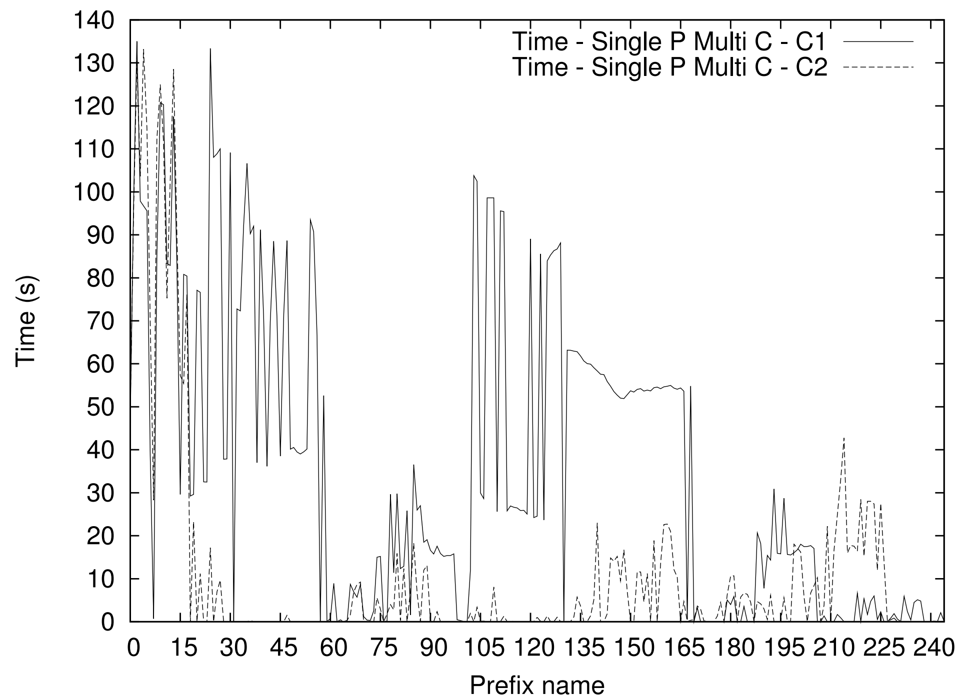


Figura 4.31: Single P Multi C - Scenario 2: tempo andata e ritorno

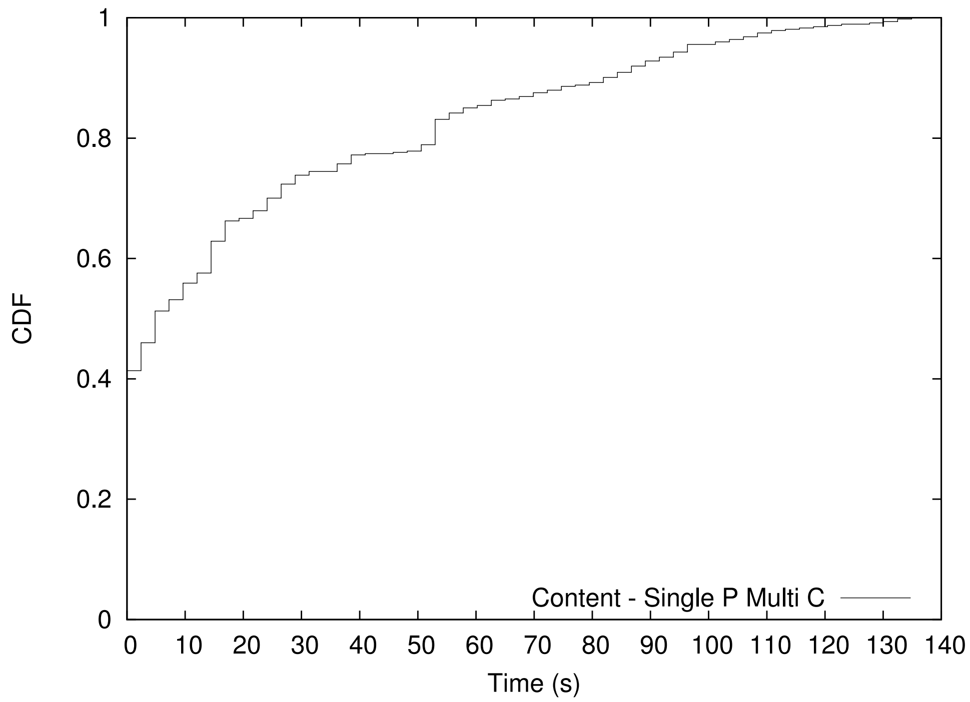


Figura 4.32: Single P Multi C - Scenario 2: tempo andata e ritorno (cumulativa)

si riportano il numero di hop attraversati dai Content e la relativa distribuzione cumulativa, rispettivamente. Infine, nelle figure 4.37 e 4.38 sono riportati i tempi impiegati ad ottenere i prefissi e la relativa distribuzione cumulativa, rispettivamente.

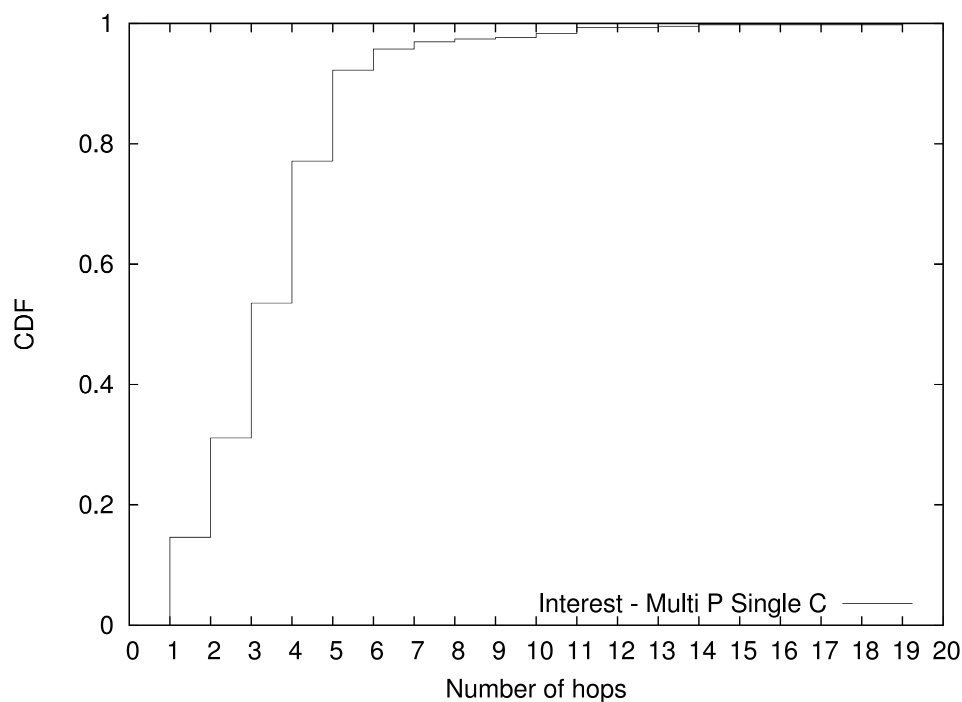


Figura 4.33: Multi P Single C - Scenario 1: hop attraversati (cumulativa)

In questo scenario sono stati richiesti un numero di prefissi distinti pari a 320; di questi 320 ne sono stati soddisfatti 317, cioè sono stati emessi 317 contenuti che prima o poi sono giunti al Consumer che li aveva richiesti, determinando una percentuale del 99,06% di richieste soddisfatte. Di queste, il 78,23% è stato soddisfatto dalla cache, il restante 21,77% dal Producer (in questo caso quattro).

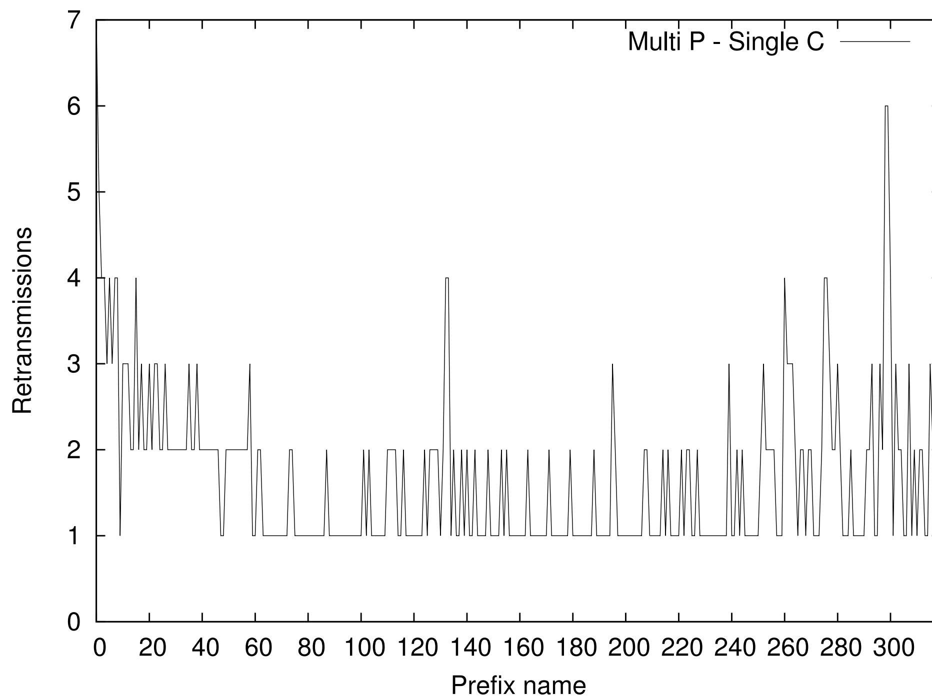


Figura 4.34: Multi P Single C - Scenario 1: numero ritrasmissioni

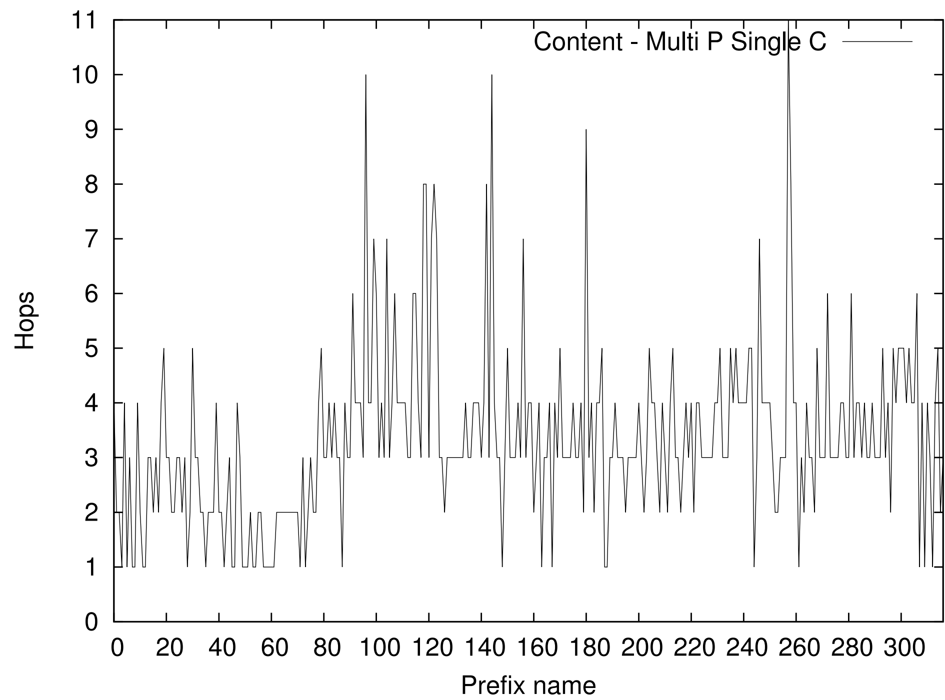


Figura 4.35: Multi P Single C - Scenario 1: hop attraversati

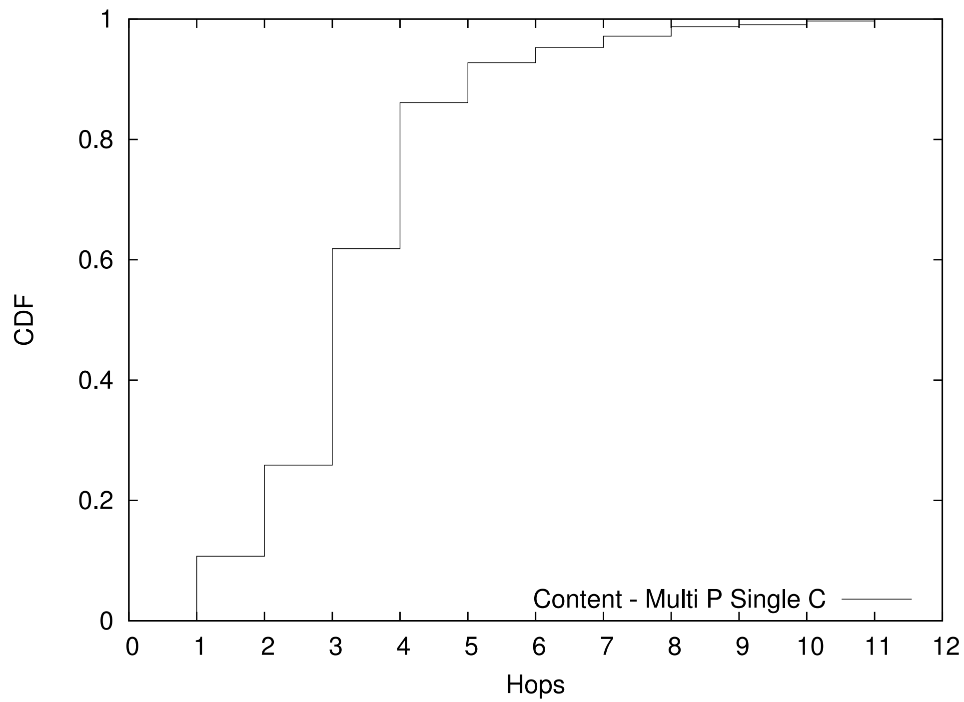


Figura 4.36: Multi P Single C - Scenario 1: hop attraversati (cumulativa)

Figura 4.37: Multi P Single C - Scenario 1: tempo andata e ritorno

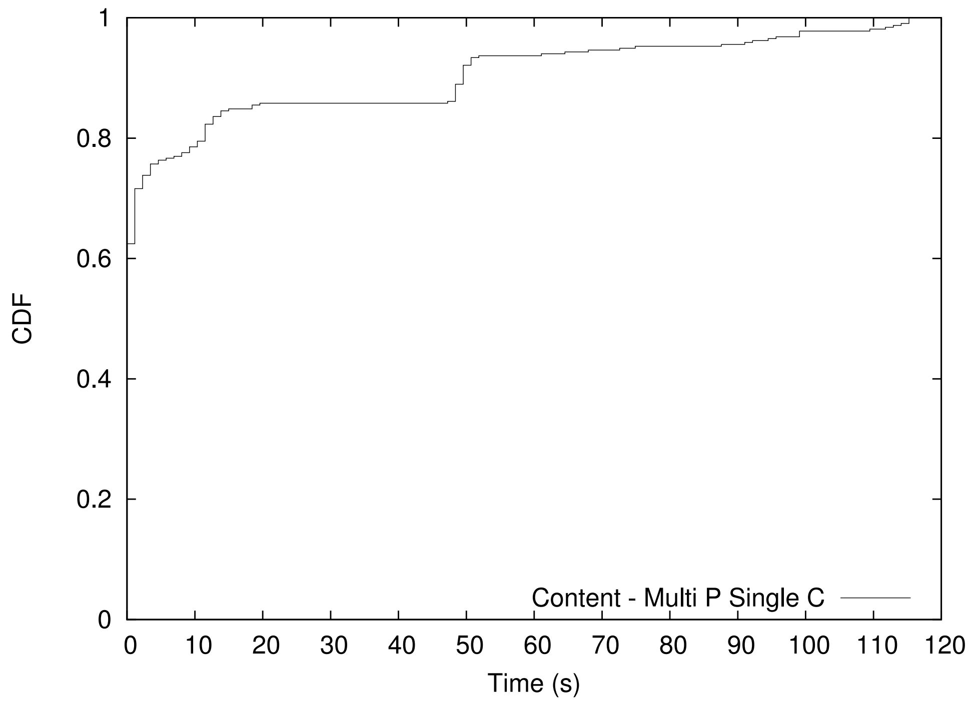


Figura 4.38: Multi P Single C - Scenario 1: tempo andata e ritorno (cumulativa)

4.2.7 Multi Produttore Singolo Consumatore - Scenario 2

I grafici relativi agli Interest sono riportati in figura 4.39 e 4.40, e riportano la distribuzione cumulativa degli hop fatti dagli Interest e del numero di ritrasmissioni di ciascun prefisso, rispettivamente. Nelle figure 4.41 e 4.42 si riportano il numero di hop attraversati dai Content e la relativa distribuzione cumulativa, rispettivamente. Infine, nelle figure 4.43 e 4.44 sono riportati i tempi impiegati ad ottenere i prefissi e la relativa distribuzione cumulativa, rispettivamente.

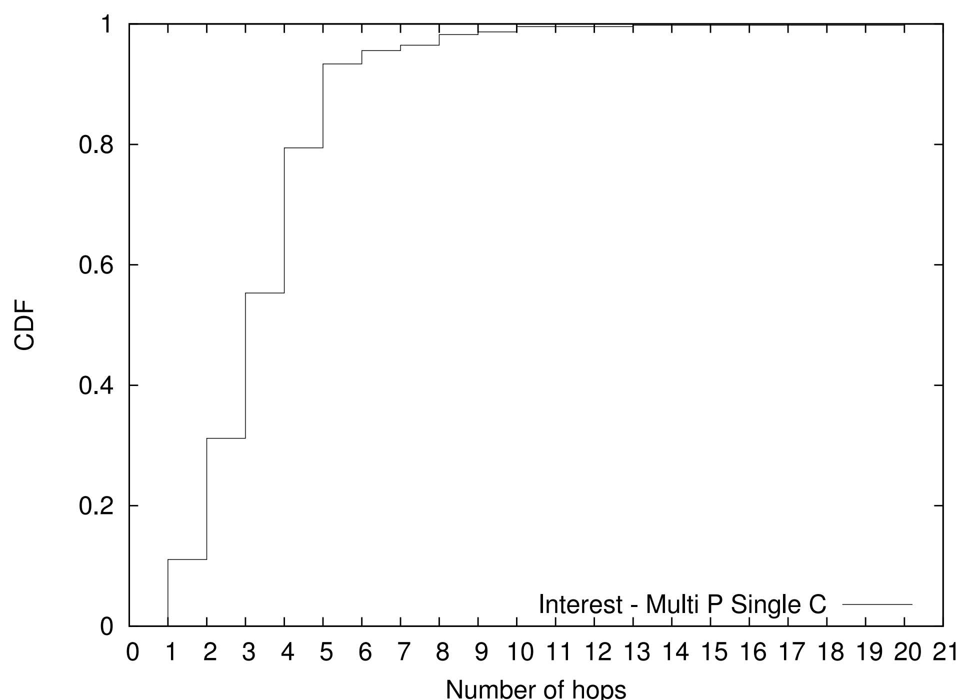


Figura 4.39: Multi P Single C - Scenario 2: hop attraversati (cumulativa)

In questo scenario sono stati richiesti un numero di prefissi distinti pari a 346; di questi 346 ne sono stati soddisfatti 345, cioè sono stati emessi 345 contenuti che prima o poi sono giunti al Consumer che li aveva richiesti, determinando una percentuale del 99,71% di richieste soddisfatte. Di queste,

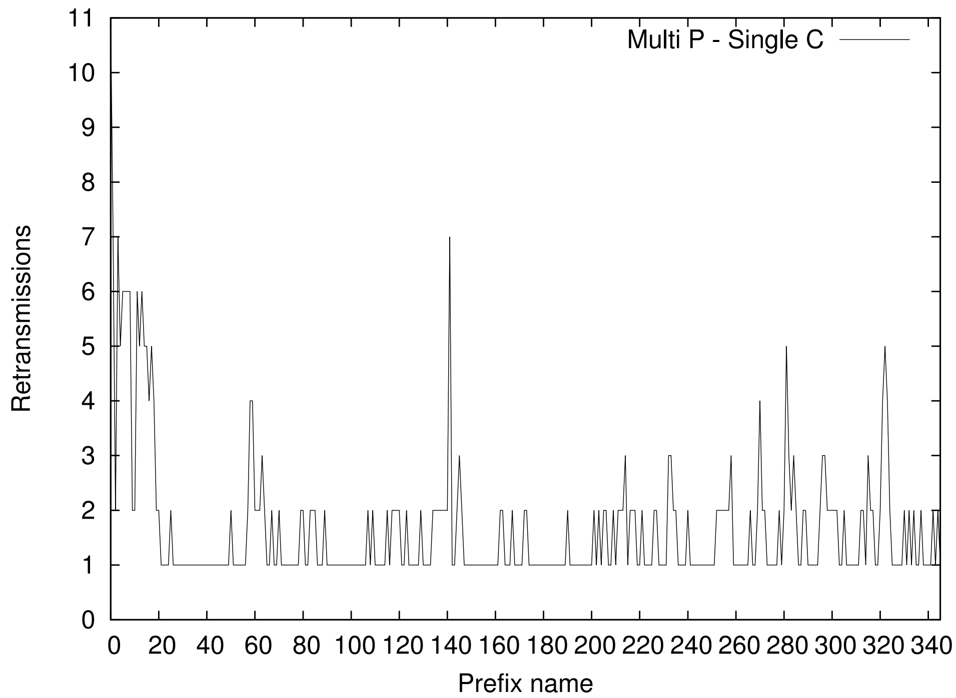


Figura 4.40: Multi P Single C - Scenario 2: numero ritrasmissioni

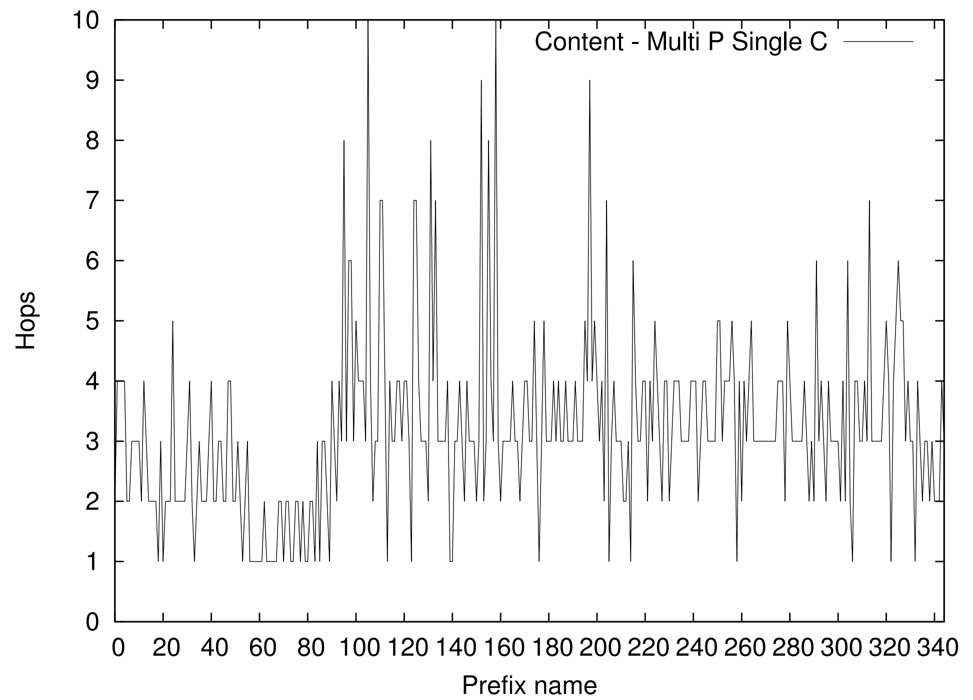


Figura 4.41: Multi P Single C - Scenario 2: hop attraversati

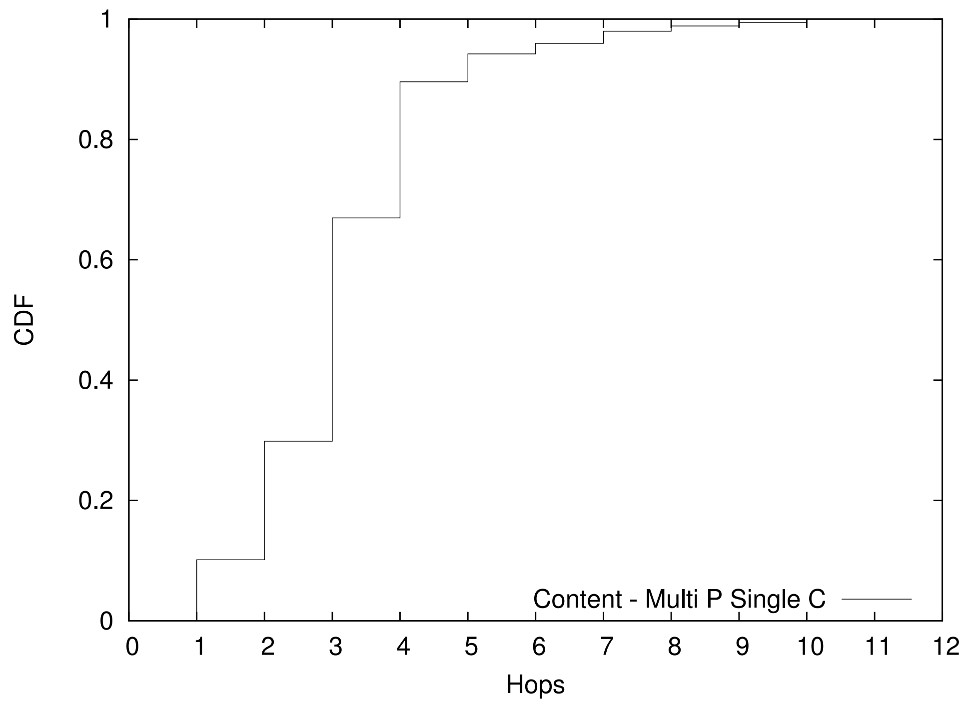


Figura 4.42: Multi P Single C - Scenario 2: hop attraversati (cumulativa)

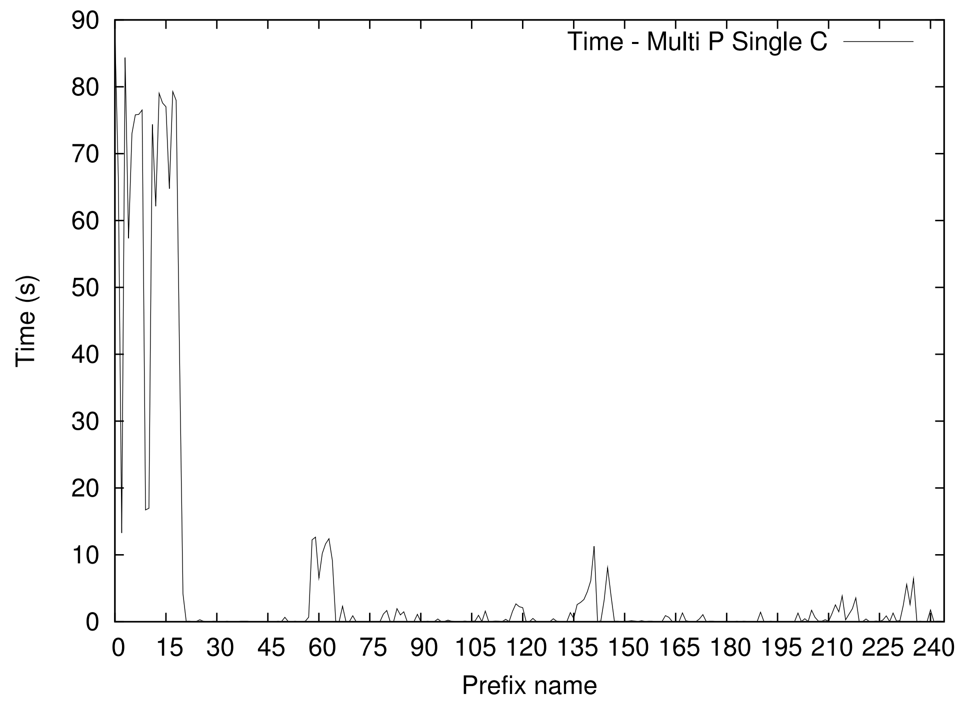


Figura 4.43: Multi P Single C - Scenario 2: tempo andata e ritorno

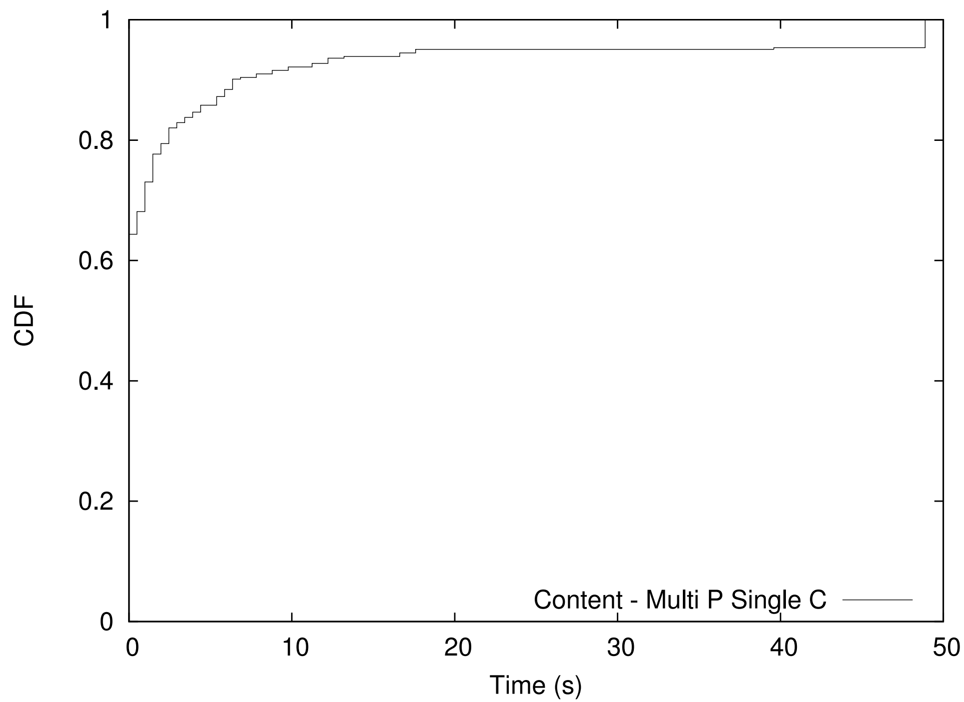


Figura 4.44: Multi P Single C - Scenario 2: tempo andata e ritorno (cumulativa)

il 80,29% è stato soddisfatto dalla cache, il restante 19,71% dal Producer (in questo caso quattro).

4.2.8 Multi Produttore Singolo Consumatore - Scenario 3

I grafici relativi agli Interest sono riportati in figura 4.45 e 4.46, e riportano la distribuzione cumulativa degli hop fatti dagli Interest e del numero di ritrasmissioni di ciascun prefisso, rispettivamente. Nelle figure 4.47 e 4.48 si riportano il numero di hop attraversati dai Content e la relativa distribuzione cumulativa, rispettivamente. Infine, nelle figure 4.49 e 4.50 sono riportati i tempi impiegati ad ottenere i prefissi e la relativa distribuzione cumulativa, rispettivamente.

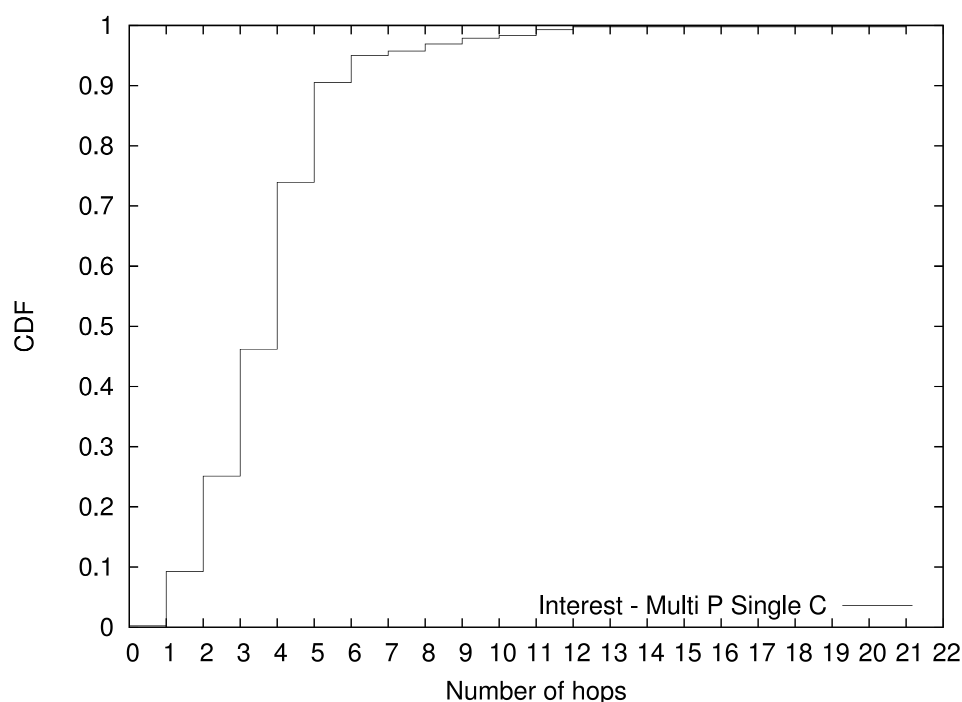


Figura 4.45: Multi P Single C - Scenario 3: hop attraversati (cumulativa)

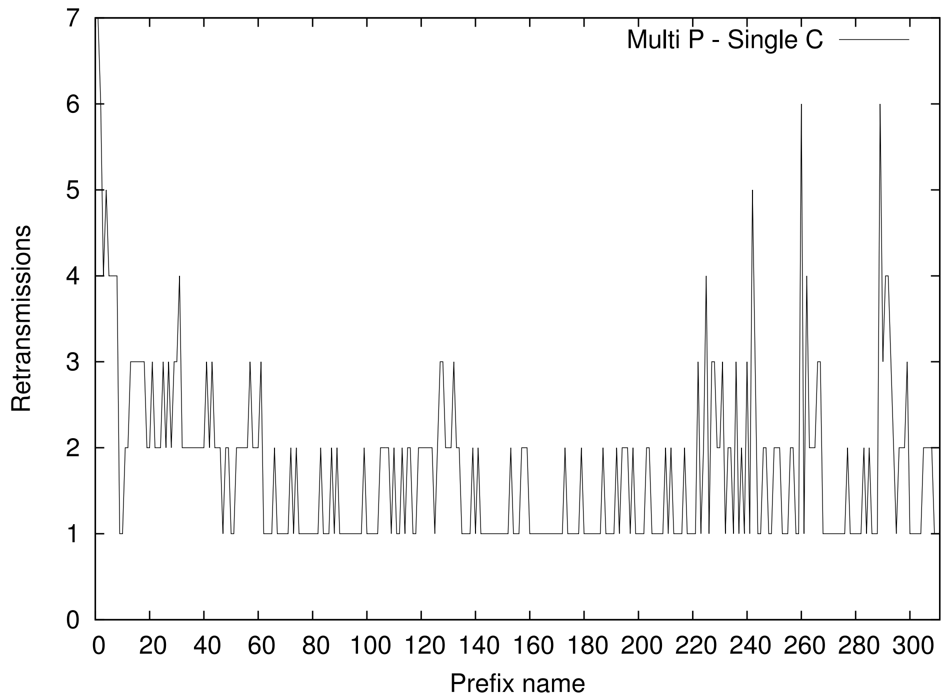


Figura 4.46: Multi P Single C - Scenario 3: numero ritrasmissioni

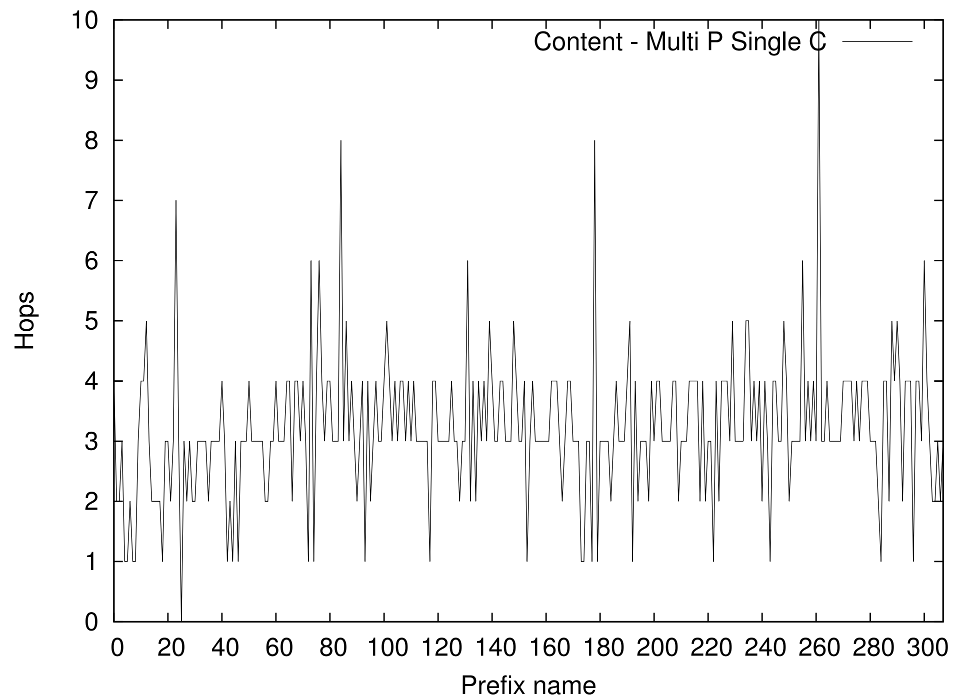


Figura 4.47: Multi P Single C - Scenario 3: hop attraversati

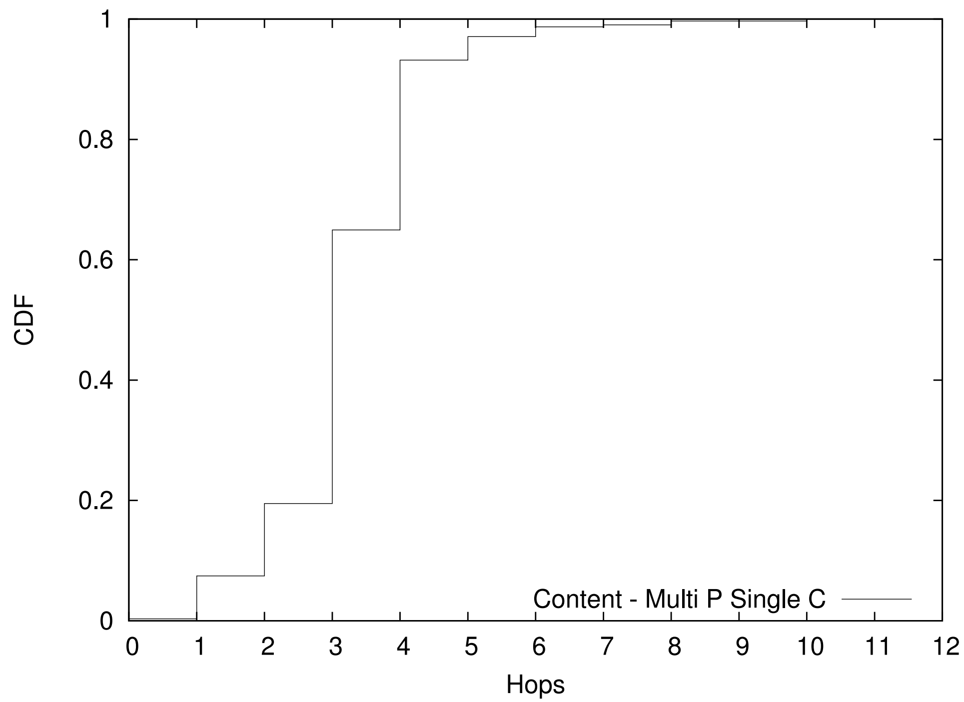


Figura 4.48: Multi P Single C - Scenario 3: hop attraversati (cumulativa)

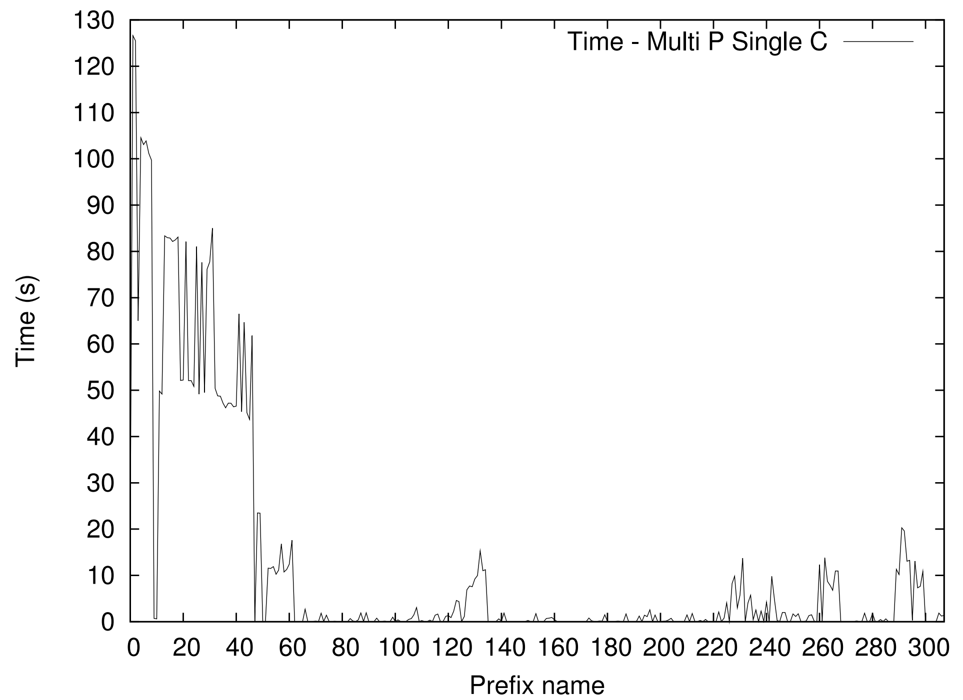


Figura 4.49: Multi P Single C - Scenario 3: tempo andata e ritorno

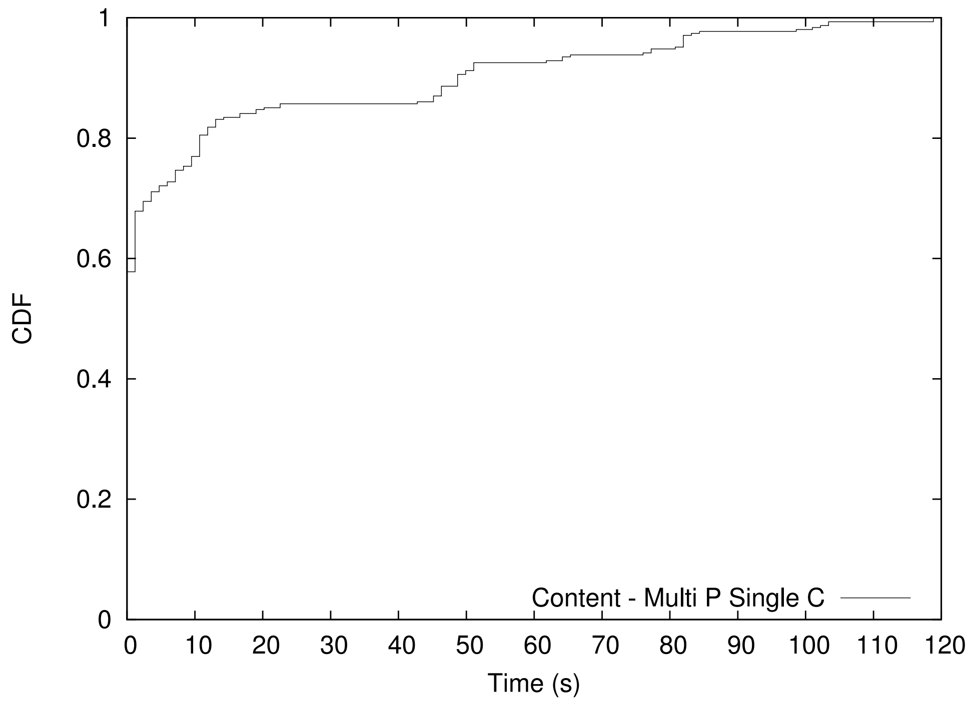


Figura 4.50: Multi P Single C - Scenario 3: tempo andata e ritorno (cumulativa)

In questo scenario sono stati richiesti un numero di prefissi distinti pari a 312; di questi 312 ne sono stati soddisfatti 308, cioè sono stati emessi 308 contenuti che prima o poi sono giunti al Consumer che li aveva richiesti, determinando una percentuale del 98,72% di richieste soddisfatte. Di queste, il 86,36% è stato soddisfatto dalla cache, il restante 13,64% dal Producer.

4.2.9 Multi Produttore Multi Consumatore - Scenario 1

I grafici relativi agli Interest sono riportati in figura 4.51 e 4.52, e riportano la distribuzione cumulativa degli hop fatti dagli Interest e del numero di ritrasmissioni di ciascun prefisso, rispettivamente. Nelle figure 4.53 e 4.54 si riportano il numero di hop attraversati dai Content e la relativa distribuzione cumulativa, rispettivamente. Infine, nelle figure 4.55 e 4.56 sono riportati i tempi impiegati ad ottenere i prefissi e la relativa distribuzione cumulativa, rispettivamente (in questo caso quattro).

In questo scenario sono stati richiesti un numero di prefissi distinti pari a 390; qui però sono presenti quattro Consumer, quindi si considerano un numero superiori di richieste, se si considera che i quattro Consumer possono avere richiesto i medesimi prefissi. Perciò in totale le richieste distinte tra i quattro Consumer sono 1549. Di questi 1549 ne sono stati soddisfatti 1546, cioè sono stati emessi 1546 contenuti che prima o poi sono giunti ciascuno al Consumer che li aveva richiesti, determinando una percentuale del 99,81% di richieste soddisfatte. Di queste, il 86,87% è stato soddisfatto dalla cache, il restante 13,13% dal Producer (in questo scenario sono tre).

4.2.10 Multi Produttore Multi Consumatore - Scenario 2

I grafici relativi agli Interest sono riportati in figura 4.57 e 4.58, e riportano la distribuzione cumulativa degli hop fatti dagli Interest e del numero di ritrasmissioni di ciascun prefisso, rispettivamente. Nelle figure 4.59 e 4.60 si riportano il numero di hop attraversati dai Content e la relativa distribuzione cumulativa, rispettivamente. Infine, nelle figure 4.61 e 4.62 sono riportati i tempi impiegati ad ottenere i prefissi e la relativa distribuzione cumulativa, rispettivamente.

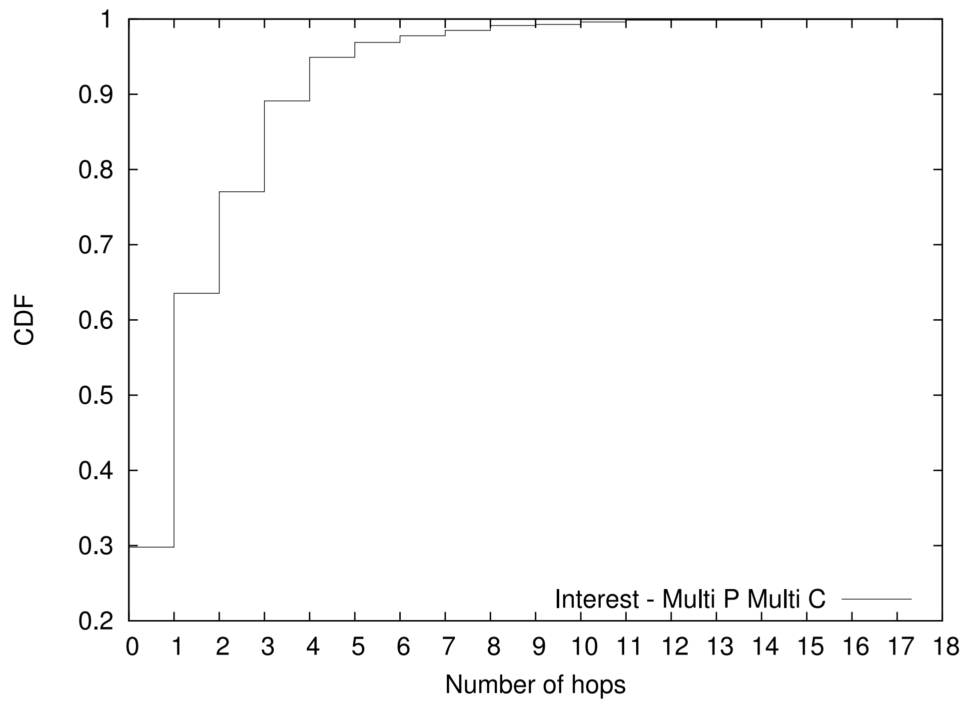


Figura 4.51: Multi P Multi C - Scenario 1: hop attraversati (cumulativa)

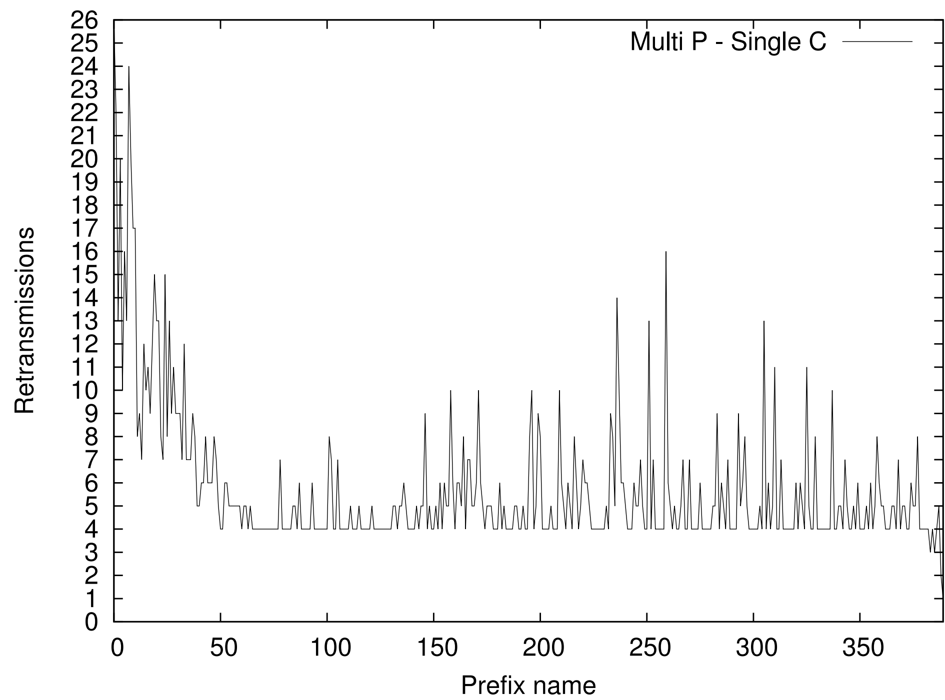


Figura 4.52: Multi P Multi C - Scenario 1: numero ritrasmissioni

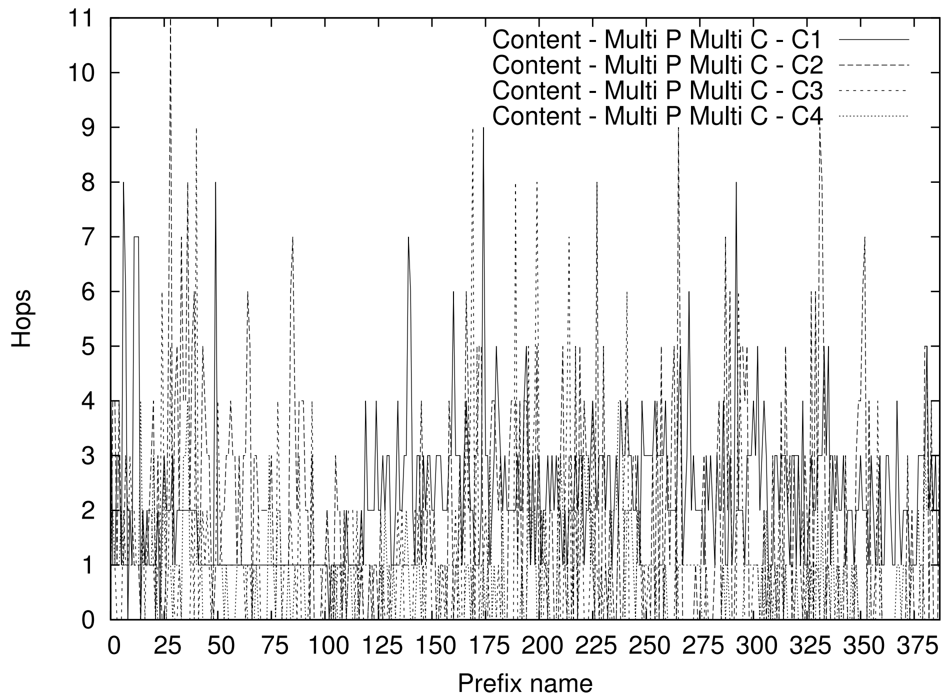


Figura 4.53: Multi P Multi C - Scenario 1: hop attraversati

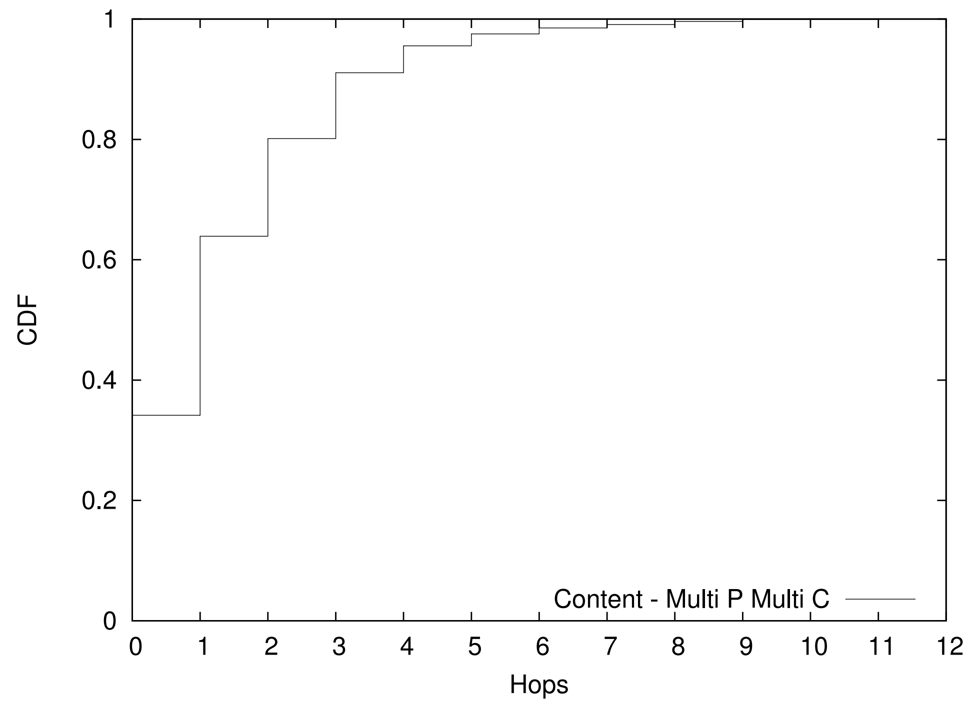


Figura 4.54: Multi P Multi C - Scenario 1: hop attraversati (cumulativa)

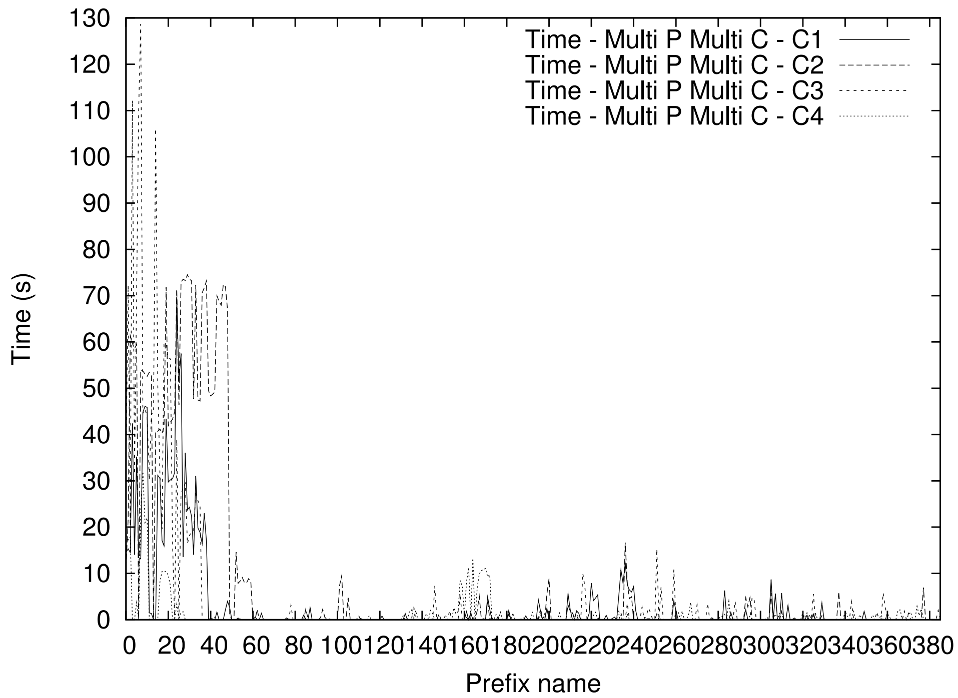


Figura 4.55: Multi P Multi C - Scenario 1: tempo andata e ritorno

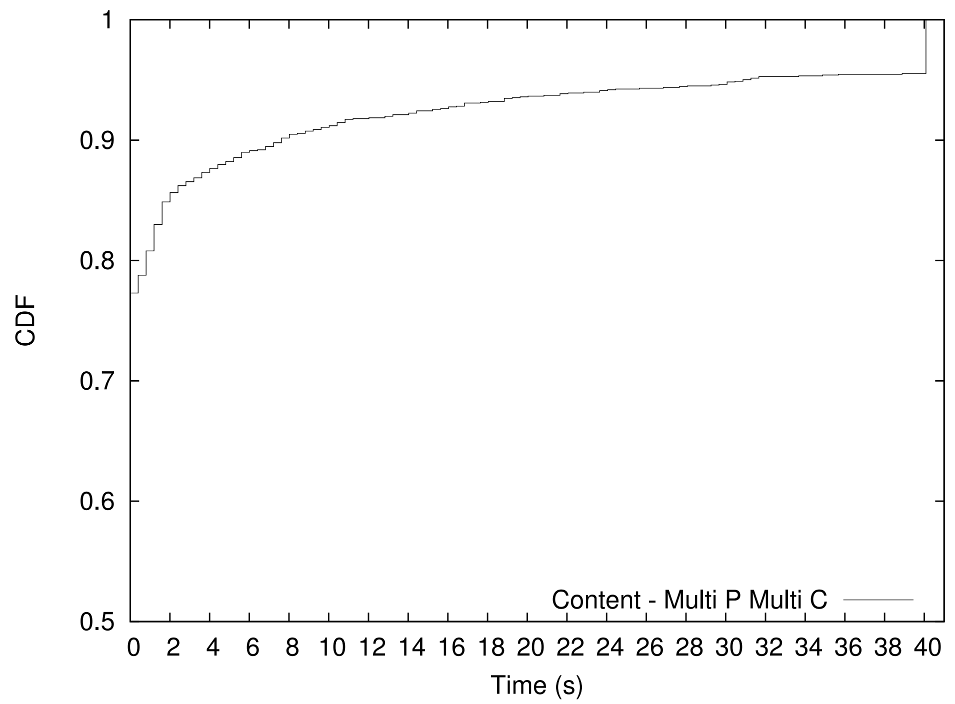


Figura 4.56: Multi P Multi C - Scenario 1: tempo andata e ritorno (cumulativa)

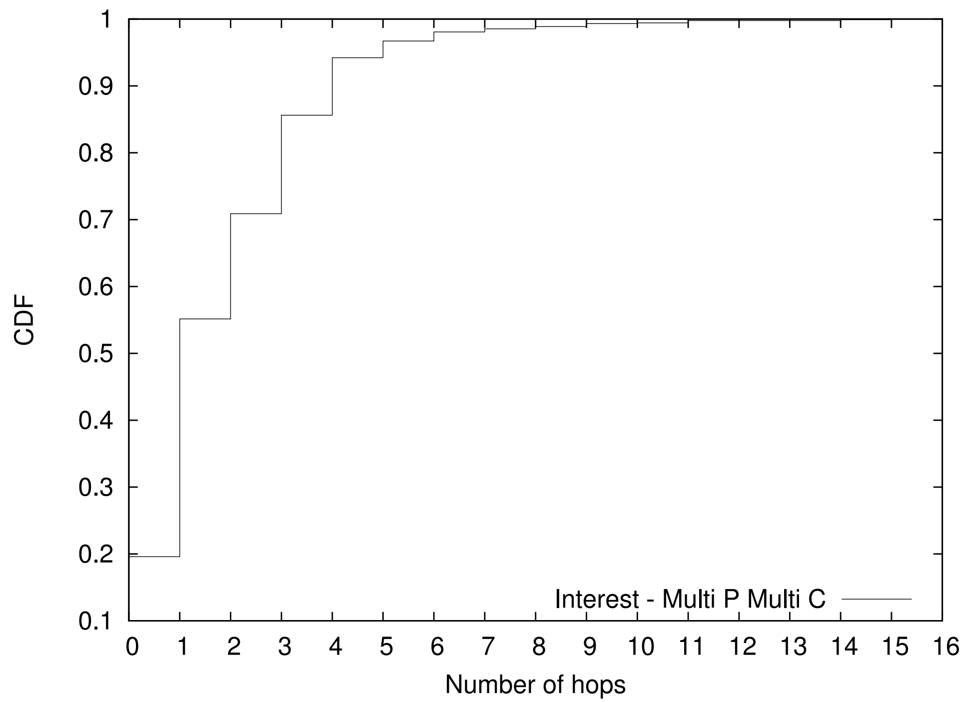


Figura 4.57: Multi P Multi C - Scenario 2: hop attraversati (cumulativa)

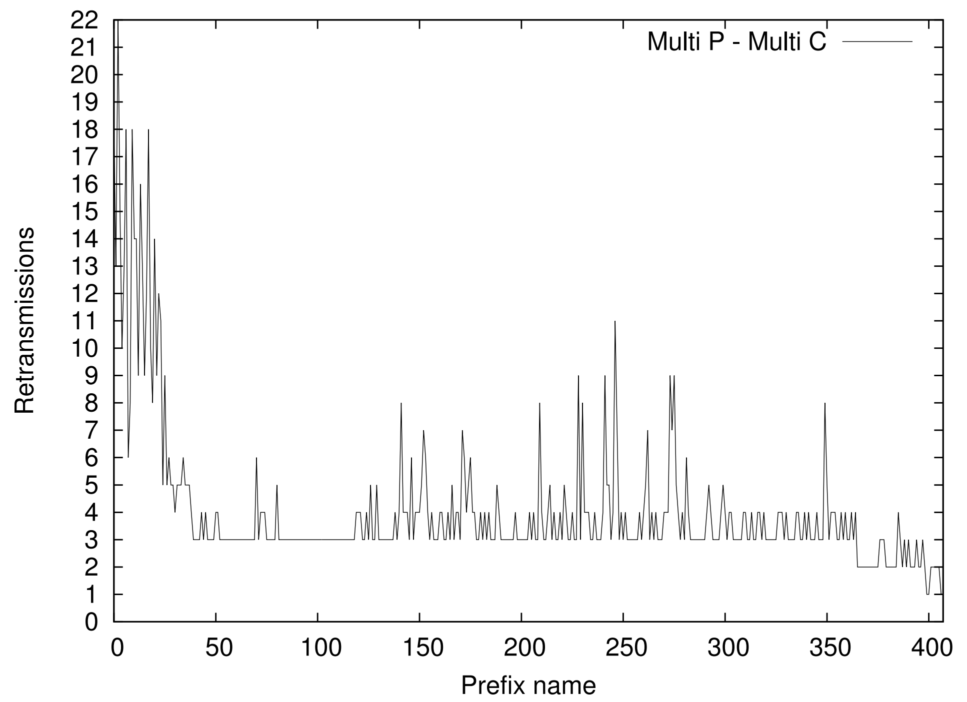


Figura 4.58: Multi P Multi C - Scenario 2: numero ritrasmissioni

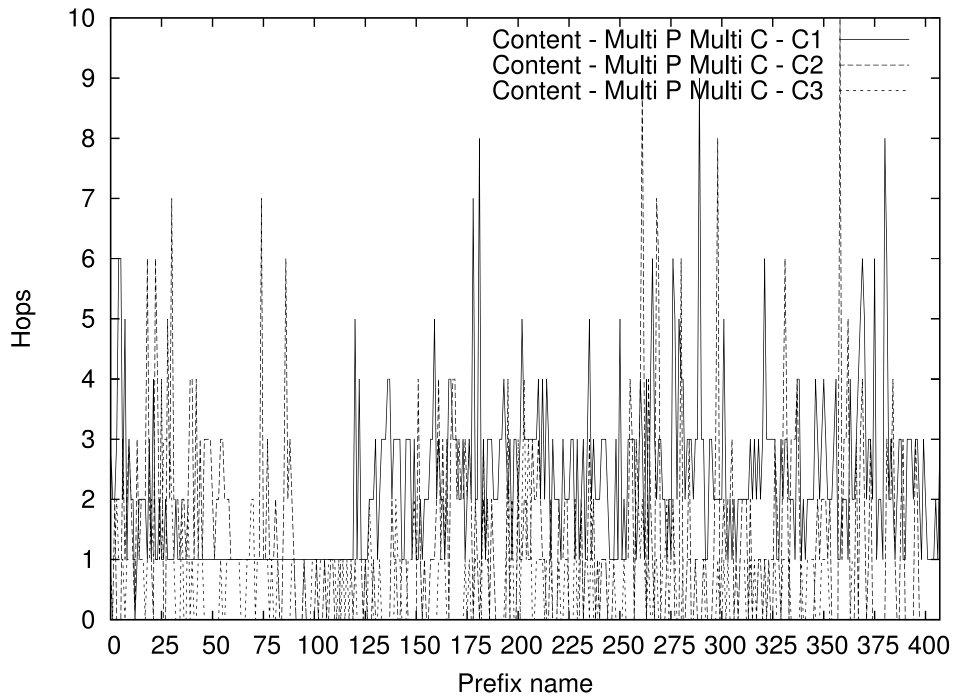


Figura 4.59: Multi P Multi C - Scenario 2: hop attraversati

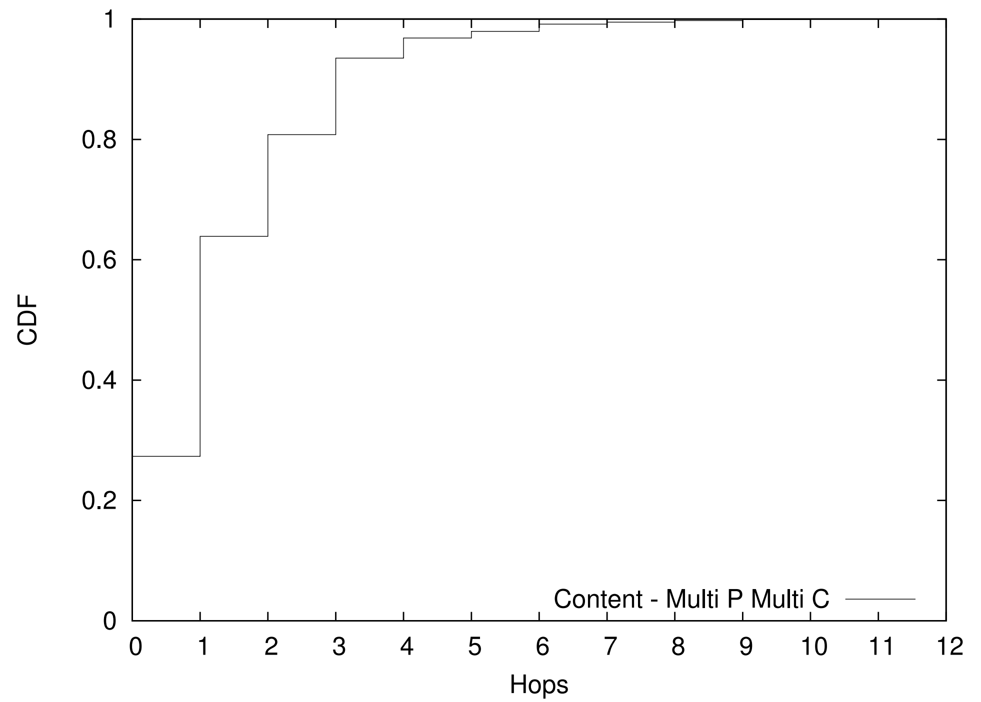


Figura 4.60: Multi P Multi C - Scenario 2: hop attraversati (cumulativa)

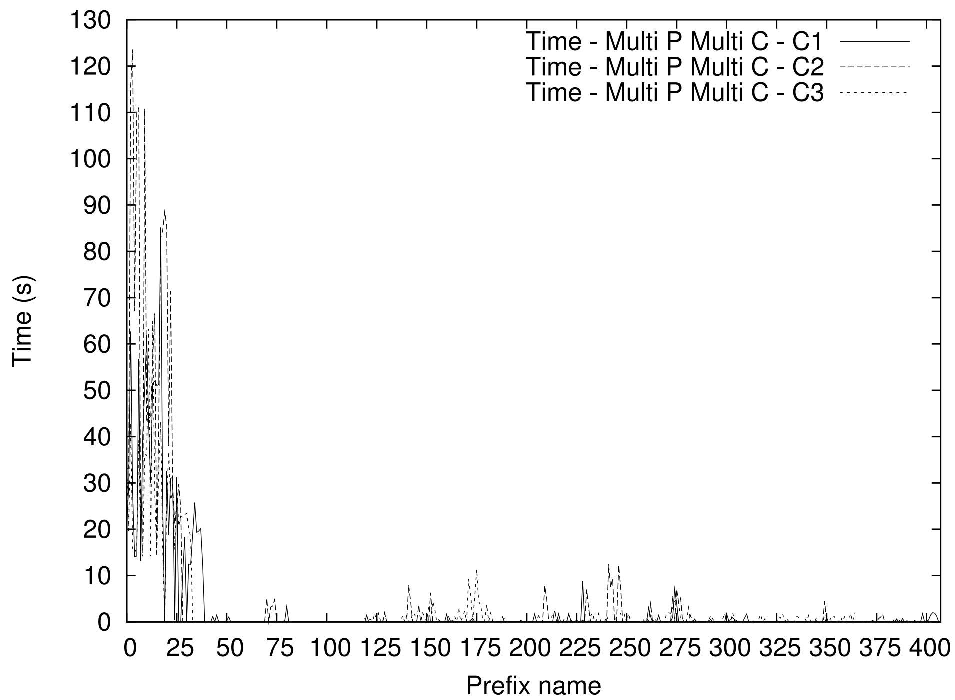


Figura 4.61: Multi P Multi C - Scenario 2: tempo andata e ritorno

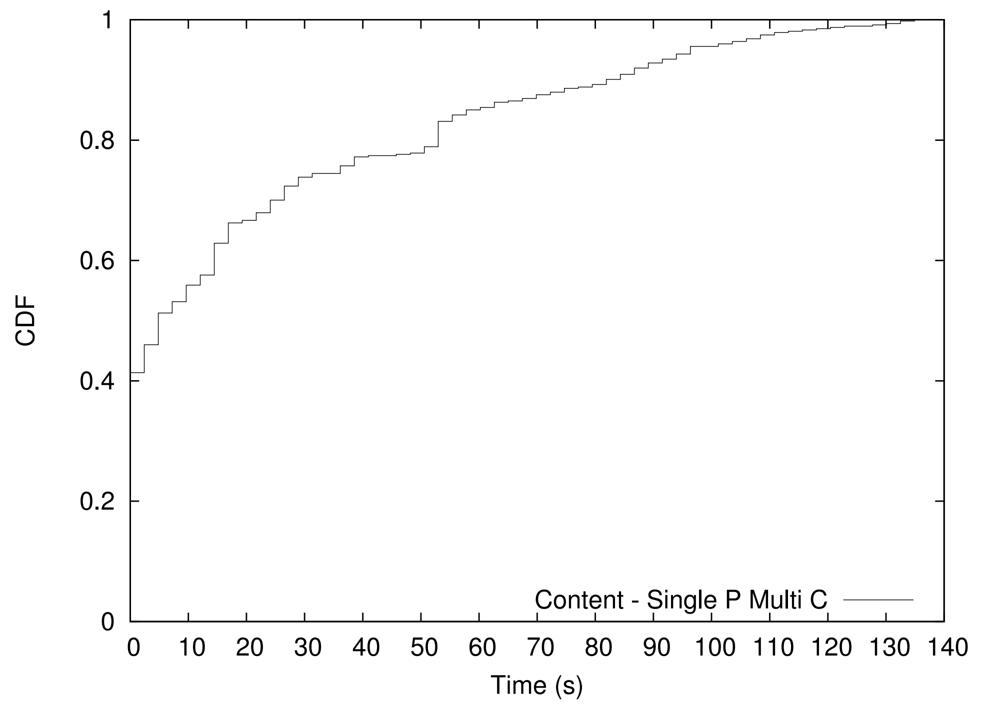


Figura 4.62: Multi P Multi C - Scenario 2: tempo andata e ritorno (cumulativa)

In questo scenario sono stati richiesti un numero di prefissi distinti pari a 408; qui però sono presenti tre Consumer, quindi si considerano un numero superiori di richieste, se si considera che i tre Consumer possono avere richiesto i medesimi prefissi. Perciò in totale le richieste distinte tra i tre Consumer sono 1171. Di questi 1171 ne sono stati soddisfatti 1171, cioè sono stati emessi 1171 contenuti che prima o poi sono giunti ciascuno al Consumer che li aveva richiesti, determinando la totalità di richieste soddisfatte. Di queste, il 84,37% è stato soddisfatto dalla cache, il restante 15,63% dal Producer (in questo scenario sono tre).

4.3 Considerazioni

Osservando i grafici relativi alle ritrasmissioni degli Interest si può notare che le ritrasmissioni sono molto più alte per primi prefissi che sono stati richiesti, per poi tendere a concentrarsi in una fascia più bassa; questo fenomeno è dovuto ad un periodo di transitorio iniziale a causa del fatto che i veicoli che entrano nella mappa partono da confini della mappa stessa, creando quindi una situazione di "vuoto" iniziale. Lo stesso fenomeno può anche riscontrarsi nella parte finale della simulazione, a causa del fatto che quando un veicolo esce dalla mappa, o termina la sua simulazione, non è più in grado di dare contributo alla diffusione dei contenuti. Durante il transitorio il Consumer ha molte meno opportunità di riuscire ad ottenere il contenuto desiderato, ed è quindi costretto ritrasmettere la richiesta più volte. Questo si ripercuote anche sui risultati ottenuti per tempi impiegati ad ottenere il contenuto. La densità dei veicoli ha ovviamente un forte impatto sui tempi impiegati a reperire i contenuti. I tempi per ottenere i contenuti si abbassano notevolmente in una condizione di regime, quando cioè i veicoli sono ben distribuiti sulla mappa e muovendo consento una distribuzione più capillare dell'informazione. Questo lo si può riscontrare sia negli scenari a singolo consumatore, ma ancora di più in quelli multi consumatore; osservando i grafici del numero di hop fatto dai Content e quelli sui tempi i impiegati a soddisfare i contenuti, si può notare in generale un contenuto potrà potenzialmente fare meno hop, poiché esso si è già diffuso precedentemente a causa della richiesta che era già arrivata da un altro Consumer. Nei casi multi Produttore singolo consumatore si ha la conferma del fatto che maggiore sono i distributori, migliori saranno gli

indici di performance. Nel caso multi Produttore multi Consumatore, si può vedere come siano ridotti di molto tutti gli indici di performance, sia in virtù della situazione di regime, che della diffusione dei contenuti dovuta ai molteplici Produttori.
Dall'analisi condotta emerge che la cache svolge un ruolo importantissimo, poiché in tutti gli scenari, la maggior parte delle richieste vengono soddisfatte proprio dalla cache di un veicolo di passaggio, o dalla proprio cache nel caso si verifichi la condizione descritta in precedenza. I casi in cui si crea invece un percorso diretto di uno o più tra Producer e Consumer sono molti di meno.

Capitolo 5

Conclusioni

Il lavoro svolto ha individuato due aspetti: da un lato l'importanza di avere un buon modello della mobilità veicolare, in particolare mettere a punto e configurare opportunamente i tool per fare sì che questi generino la mobilità desiderata in base alle esigenze dell'applicazione oggetto di studio; questo non è comunque sufficiente, perché oltre ai requisiti e alle esigenze dell' applicazione, ci deve essere una interazione con il simulatore di rete, ed è quindi opportuno che il modello della mobilità sia compatibile sia con gli input richiesti dal simulatore, che con gli input richiesti di altri modelli come ad esempio CORNER, per avere una mappa il più realistica possibile. Dall'altro lato, i dati raccolti hanno evidenziato che i tempi registrati per il reperimento dei contenuti sono compatibili con scenari in cui si richiedono e scambiano informazioni di piccole dimensioni, come ad esempio immagini o news relative a eventi locali in zone limitrofe. NDN per il veicolare si adatta bene quindi in scenari orientati ad applicazioni cooperative. Si pensi ad esempio ad uno scenario in cui si è in coda a causa di un incidente e il veicolo più lontano dal luogo dell'accaduto vuole capire cosa sta succedendo. In uno scenario di questo genere è auspicabile pensare che i veicoli più vicini possano generare e pubblicare contenuti che possono essere propagati all'indietro lungo la coda a fronte di eventuali richieste. Oppure si pensi ad uno scenario in cui si è creato un ingorgo a causa di una manifestazione e si vuole quindi pianificare un percorso alternativo. Non sono invece compatibili con scenari per la fruizione di contenuti multimediali in tempo reale, quali ad esempio film e musica, per i quali possono comunque essere usati i metodi tradizionali.

Appendice A

Funzioni di utilità e script

A.1 Esempio di file di configurazione, modalità 1

```
MODE    1
ORIGIN  34040826 -118463221
WIDTH   2100
HEIGHT  2100
INTERNAL_NODES  YES
RT1     "/home/nrl/VMF/test_la/TGR06037.RT1"
RT2     "/home/nrl/VMF/test_la/TGR06037.RT2"
BUILD_SUMO_OUTPUT       YES
SUMO_SIMULATION_TIME    7200
SUMO_NODES_FILE SUMOnodes.xml
SUMO_EDGES_FILE SUMOedges.xml
SUMO_TURNS_FILE SUMOturns.xml
SUMO_FLOWS_FILE SUMOflows.xml
SUMO_NET_FILE SUMO.net.xml
SCENARIO RANDOM
SCENARIO_DIJKSTRA_WEIGHT TIME
RANDOM_SCENARIO_DESTINATION_TYPE RANDOM 0
AVERAGE_ARRIVALS 8000
ENTRY_FLOW_MODE WEIGHTED
WEIGHTED_FLOWS_EXPONENT 0
```

A.2 Esempio di file di configurazione, modalità 7

```
MODE 7
SUMO_INPUT_NODES_FILE "/home/nrl/contolic/mobility_scenario_t/
mob_8000v_0e_bis/SUMOnodes.xml"
SUMO_INPUT_EDGES_FILE "/home/nrl/contolic/mobility_scenario_t
/mob_8000v_0e_bis/SUMOedges.xml"
SUMO_INPUT_NETWORK_FILE "/home/nrl/contolic/mobility_scenario_t/
mob_8000v_0e_bis/SUMO.net.xml"
SUMO_INPUT_TRACE_FILE "/home/nrl/contolic/mobility_scenario_t/
mob_8000v_0e_bis/sumoOutput.xml"
```

A.3 Funzionalità

```
#include <iostream>
#include <fstream>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unordered_map>
#include "utils.h"

using namespace std;
#define MAX_V_TO_ADD 10
#define MAX_V_STAT 10
#define MAX_V_DYNA 100

/*Remove all par from the original file and write the
   proper one*/
void Read_and_write_out_mobility (char* param[]){
        std::ifstream read_mobility;
    std::ifstream res_out;
    std::ofstream write_new_mobility;
        int numLine, i, res_sys;
        int nodeId, time, x, y, z;
```

```
    char bre1, com1, com2, bre2;
        //To avoid hard-coded
        char cmd[200];
        char fout[200];
        strcpy(cmd, "wc -l ");
        strcat(cmd, " > ");
        strcat(cmd, fout);
        cout<<"Command is "<<cmd<<"\n";;
        res_sys = system((char*)cmd);
        cout<<"system executed!!"<<"\n";
        cout<<"res_sys= "<<res_sys<<"\n";
        if(res_sys == -1){
                cout<<"Error occured while executing
                   system"<<"\n";
        }else{
                res_out.open(param[4], std::ifstream::
                   in);
                res_out>>numLine;
                for(i = 0; i < numLine; i++){
                        read_mobility>>nodeId>>time>>
                           bre1>>x>>com1>>y>>com2>>z>>
                           bre2;
                        write_new_mobility<<nodeId<<"\
                           t"<<time<<"\t"<<x<<"\t"<<y
                           <<"\t"<<z<<"\n";
                }
                cout<<"File of numLine= "<<numLine<<"
                   has been written"<<"\n";
        }

}

void AddStaticVehicle (char* param[]){
        int simTime; //Duration of simulation
        int numVtoAdd; //Number of vehicle to add in
           the file
        int carsId[MAX_V_TO_ADD]; //id of added cars
```

```
int posXY[2*MAX_V_TO_ADD]; //(x,y) pos of add
   cars
int i, count;
bool endOfWork = false;
int nodeId, time, x, y, z;
char bre1, com1, com2, bre2;
std::ifstream read_mobility;
std::ofstream write_new_mobility;

count = 0;
simTime = atoi(param[2]);
numVtoAdd = atoi(param[3]);
for(i = 0; i < numVtoAdd; i++){
        carsId[i] = atoi(param[3*i+6]); //id
        posXY[2*i] = atoi(param[3*i+7]); //x
        posXY[2*i+1] = atoi(param[3*i+8]); //y
}

read_mobility.open(param[4], std::ifstream::in
   );
 write_new_mobility.open(param[5], std::
    ofstream::out);
cout<<"simTime="<<simTime<<" numV="<<numVtoAdd
   <<" carsId="<<carsId[0]<<"\n";
while (!endOfWork){
        read_mobility>>nodeId>>time>>bre1>>x>>
           com1>>y>>com2>>z>>bre2;
        if(time == simTime + 1){
                break;
        }
        if(count == time){
                for(i = 0; i < numVtoAdd; i++)
                   {
                        write_new_mobility<<
                           carsId[i]<<"\t"<<
                           count<<"\t"<<bre1<<
                           posXY[2*i]<<com1<<
```

```
                                posXY[2*i+1]<<com2
                                <<z<<bre2<<"\n";
                        }
                        count ++;
                }
                write_new_mobility<<nodeId<<"\t"<<time
                   <<"\t"<<bre1<<x<<com1<<y<<com2<<z<<
                   bre2<<"\n";
        }
        cout<<"All line has been copied.."<<"\n";
}

int main (int argc, char* argv[]){
  unordered_map <int,int> map;
  std::ifstream mobileInput;
  int op;//Identify the method to call
  int posXYstat[2*MAX_V_STAT];
  int id[MAX_V_STAT+MAX_V_DYNA];
  int nStat;
  int nDyna;
  int simTime;
  int startTime;
  int ti;
  bool found;
  bool exit;
  int index, nodeNo;
  double time = 0;
  double x=0,y=0,z=0;
  std::ifstream read_mobility;
  std::ofstream write_new_mobility;
  cout<<"This is a test"<<"\n";
  op = atoi (argv[1]);
  cout<<"Operazione "<<op<<"\n";
  switch (op){
    case 0:
      cout<<"Removing () from the original mobility
         file..."<<"\n";
```

```
      Read_and_write_out_mobility (argv);
      break;
    case 1:
      cout<<"Adding vehicle..."<<"\n";
      AddStaticVehicle (argv);
      break;
    case 2:
      nStat = atoi(argv[2]);
      index = 2+(nStat*3+1);
      nDyna = atoi(argv[index]);
      for(int i = 0; i < nStat; i++){
        id[i] = atoi(argv[3*i+3]);
        posXYstat[2*i] = atoi(argv[3*i+4]);
        posXYstat[2*i+1] = atoi(argv[3*i+5]);
        cout<<"[DEBUG] id["<<i<<"] = "<<id[i]<<"\n";
      }
      for(int i = 0; i < nDyna; i++){
        id[nStat+i] = atoi(argv[index+i+1]);
        cout<<"[DEBUG] id["<<i<<"] = "<<id[nStat+i
           ]<<"\n";
      }
      write_new_mobility.open("vstat_sumo.mobility",
         std::ofstream::out);
      for(int t = 0; t <= 20; t++){
        for(int i = 0; i< (nStat + nDyna); i++){
             if(i < nStat){
                cout<<"[DEBUG] i="<<i<<"\n";
                //Get information from vector
                write_new_mobility<<id[i]<<"\t"<<t<<"\
                   t"<<posXYstat[2*i]<<"\t"<<posXYstat
                   [2*i+1]<<"\t"<<"0"<<"\n";
               }else{
                cout<<"[DEBUG] from file..."<<"\n";
                //Get information from file
                found = false;
                 read_mobility.open("
                    new_wilshire_1l_50p.mobility", std
```

```
                    ::ifstream::in);
                nodeNo = 0;
                while(read_mobility>>nodeNo){
                            read_mobility>> time >> x
                              >> y >> z;
                        if(nodeNo == id[i] && t == time)
                          {
                          found = true;
                          write_new_mobility<<nodeNo<<"\
                             t"<<t<<"\t"<<x<<"\t"<<y<<"\
                             t"<<z<<"\n";
                          read_mobility.close();
                          break;
                      }
                    }
                }
            }
        }
      break;
    case 3:
      nStat = atoi(argv[2]);
      simTime = atoi(argv[5]);
      for(int i = 0; i < nStat; i++){
          id[i] = atoi(argv[3*i+6]);
          posXYstat[2*i] = atoi(argv[3*i+7]);
          posXYstat[2*i+1] = atoi(argv[3*i+8]);
          cout<<"[DEBUG] id["<<i<<"] = "<<id[i]<<"\n";
      }
      write_new_mobility.open(argv[4], std::ofstream::
         out);
      read_mobility.open(argv[3], std::ifstream::in);
       nodeNo = 0;
       read_mobility>> nodeNo>> time >> x >> y >> z;
       for(int t = 0; t <= simTime; t++){
           for(int i = 0; i< nStat; i++){
            write_new_mobility<<id[i]<<"\t"<<t<<"\t"<<
               posXYstat[2*i]<<"\t"<<posXYstat[2*i
```

```
                    +1]<<"\t"<<"0"<<"\n";
            }
            while(time != t + 1){
                write_new_mobility<<nodeNo<<"\t"<<t<<"\t
                    "<<x<<"\t"<<y<<"\t"<<z<<"\n";
                read_mobility>> nodeNo>> time >> x >> y >>
                    z;
             }
        }
        break;
    case 4:
        simTime = atoi(argv[2]);
         write_new_mobility.open(argv[4], std::ofstream
             ::out);
         read_mobility.open(argv[3], std::ifstream::in);
        nodeNo = 0;
          read_mobility>> nodeNo>> time >> x >> y >> z;
            for(int t = 0; t <= simTime; t++){
                while(time != t + 1){
                        write_new_mobility<<nodeNo<<"\t
                            "<<t<<"\t"<<x<<"\t"<<y<<"\t"<<
                            z<<"\n";
                  read_mobility>> nodeNo>> time >> x >> y
                      >> z;
              }
            }
            break;
    case 5:
             startTime = atoi(argv[2]);
              simTime = atoi(argv[3]);
              write_new_mobility.open(argv[5], std::
                  ofstream::out);
             read_mobility.open(argv[4], std::ifstream::
                 in);
              nodeNo = 0;
                    read_mobility>> nodeNo>> time >> x >>
                        y >> z;
```

```
while(time != startTime){
        read_mobility>> nodeNo>> time >> x >>
           y >> z;
}
for(int t = startTime, i = 0; i <= simTime+1;
   t++, i++){
        while(time != t + 1){
                        write_new_mobility<<
                           nodeNo<<"\t"<<t<<"\
                           t"<<x<<"\t"<<y<<"\t
                           "<<z<<"\n";
                read_mobility>> nodeNo>> time
                   >> x >> y >> z;
                }
 }
 break;
case 6:
  exit = false;
 nStat = atoi(argv[2]);
 for(int i = 0; i < nStat; i++){
     id[i] = atoi(argv[3*i+5]);
     posXYstat[2*i] = atoi(argv[3*i+6]);
    posXYstat[2*i+1] = atoi(argv[3*i+7]);
   }
  write_new_mobility.open(argv[4], std::
     ofstream::out);
   read_mobility.open(argv[3], std::ifstream::
      in);
  read_mobility>>nodeNo>> time >> x >> y >> z;
  while(true){
      ti = time;
     for(int i = 0; i< nStat; i++){
        write_new_mobility<<id[i]<<"\t"<<time
           <<"\t"<<posXYstat[2*i]<<"\t"<<
           posXYstat[2*i+1]<<"\t"<<"0"<<"\n";
      }
     while(time != ti + 1){
```

```
                write_new_mobility<<nodeNo<<"\t"<<
                   time<<"\t"<<x<<"\t"<<y<<"\t"<<z
                   <<"\n";
              if(read_mobility>> nodeNo){
                read_mobility>> time >> x >> y >> z;
               }else{
                exit = true;
                break;
               }
               }
              if(exit){
                break;
               }
      }
      break;
    case 7:
      mobileInput.open(argv[1], std::ios::in);
      int nodeNo = 0;
      while(mobileInput>>nodeNo){
        double time = 0;
        double x=0,y=0,z=0;
        int count = 0;
       int  n = 0;
       mobileInput >> time >> x >> y >> z;
       if(map.count(nodeNo)){
         count = map[nodeNo];
         count++;
          map[nodeNo] = count;
       }else{
         map[nodeNo] = 1;
       }
      }
      for(std::unordered_map<int,int>::iterator it
         =map.begin();  it!=map.end(); it++){
       cout<<"Key= "<<it->first <<" Value= "<<it->
          second<<"\n";
     }
```

```
        cout<<"Size= "<<map.size()<<"\n";
        default:
         cout<<"You haven't specified any number...\n
            ";
          break;
  }
  cout<<"New file written"<<"\n";
  return 0;
}
```

Il codice di cui sopra contiene le seguenti funzionalità:

1. partendo da una mobilità generata da SUMO che è nel formato *nodeID time (x,y,z)* , genera un altro file che può essere letto dal formato desiderato dal simulatore: *nodeID time x y z*

2. aggiungere un numero prefissato di veicoli statici (id del nodo e la posizione in coordinate) e generare un nuovo file nel formato di SUMO

3. scrive una mappa leggibile dal simulatore con un numero prefissato di veicoli statitici e dinamici

4. partendo da una mappa che è già in formato leggibile dal simulatore, è possibile aggiungere un numero prefissato di veicoli statici

5. partendo da una mappa che è già in formato leggibile dal simulatore, è possibile estrarre una parte della simulazione specificando l'ammontare di secondi che si vogliono considerare

6. partendo da una mappa che è già in formato leggibile dal simulatore, è possibile estrarre una parte della simulazione specificando il tempo di inizio e l'ammontare di secondi che si vogliono considerare

7. partendo da una mappa che è già in formato leggibile dal simulatore, è possibile aggiungere un numero prefissato di veicoli statici (id del nodo e la posizione in coordinate)

8. partendo da una mappa che è già in formato leggibile dal simulatore, conteggia il numero di veicoli

Al fine di eseguire le operazioni sopra citate è necessario specificare una modalità di lavoro, specificando la corretta modalità da linea di comando. Per eseguire la funzionalità (1) (il percorso deve contenere anche i nomi dei files):

```
./utils 0 /percorso/del/file.mobility/da/leggere /
    percorso/del/nuovo/file/mobility/da/scrivere /
    percorso/del/file/in/cui/scrivere/il/numero/di/
    linee
```

Per eseguire la funzionalità (2):

```
./utils 1 simTime nVeichToAdd /percorso/del/file.
    mobility/da/leggere /percorso/del/nuovo/file/
    mobility/da/scrivere idNode1 x1 y1 idNode2 x2 y2
    ...
```

simTime è l'ammontare di secondi della simulazione che si vogliono considerare, nVeichToAdd è il numero di nodi statici che si vogliono aggiungere, idNode, x e y sono l'identificativo del nodo e le coordinate.
Per eseguire la funzionalità (4):

```
./utils 3 NVeichToAdd /percorso/del/file.mobility/da/
    leggere /percorso/del/nuovo/file/mobility/da/
    scrivere  simTime idNode1 x1 y1 idNode2 x2 y2 ...
```

Per eseguire la funzionalità (5):

```
./util 4 simTime  /percorso/del/file.mobility/da/
    leggere /percorso/del/nuovo/file/mobility/da/
    scrivere
```

Per eseguire la funzionalità (6):

```
./util 5 startTime simTime /percorso/del/file.mobility
    /da/leggere /percorso/del/nuovo/file/mobility/da/
    scrivere
```

startTime è il tempo dal quale si vuole considerare la simulazione.
Per eseguire la funzionalità (7):

```
./util 6 NVehicToAdd /percorso/del/file.mobility/da/
   leggere /percorso/del/nuovo/file/mobility/da/
   scrivere idNode1 x1 y1 idNode2 x2 y2 ...
```

Per eseguire la funzionalità (8):

```
./util 7 /percorso/del/file.mobility/da/leggere
```

Un'altra utile funzionalità è lo script che consente la conversione da coordinate Latitudine/Longitudine a coordinate UTM della mappa che è usata dalla simulazione.

```
#!/usr/bin/perl
  use strict;
  use Geo::Proj4;
  use IO::File;
  my $proj = Geo::Proj4->new ( proj => "utm", zone =>
     $ARGV[3] );
  my $data = $ARGV[0];

  open(DATA, "< ".$data);
  my @data_raw = <DATA>;
  close(DATA);
  my $south = @data_raw[7];
  my $west = @data_raw[8];
  print "South line : $south\n";
  print "West line : $west\n";
  #SOUTH
  #Extract north coordinate
  my @res_south = split(/\t/, $south);
  my $split_coord = @res_south[0];
  print "Extracted element: $split_coord\n";
  my @res_coord = split (/[:]/, $split_coord);
  print "This is a test: @res_coord[1]\n";
  my $utm_S_north = @res_coord[1];
  #Extract  east coordinate
  my $split_coord2 = @res_south[1];
  print "Second Extracted element: $split_coord2\n";
  my $utm_S_east =  @res_south[1];
```

```
#WEST
print("\n");
print("WEST\n");
print("\n");
#Extract north coordinate
my @res_west = split(/\t/, $west);
my $split_coord2 = @res_west[0];
 print "Extracted element: $split_coord2\n";
my @res_coord2 = split (/[:]/, $split_coord2);
print "This is a test: @res_coord2[1]\n";
my $utm_W_north = @res_coord2[1];
#Estrazione coordinata east
my $split_coord3 = @res_west[1];
print "Second Extracted element: $split_coord3\n";
my $utm_W_east =  @res_west[1];
#coordinate estratte
print "Corner coordinates:\n";
print "SOUTH: $utm_S_north $utm_S_east\n";
print "WEST: $utm_W_north $utm_W_east\n";
my $xCoordZero = $utm_W_east;
my $yCoordZero = $utm_S_north;

my($real_utm_e, $real_utm_n) = $proj->forward($ARGV
    [1], $ARGV[2]);
my $utm_map_x = $real_utm_e - $xCoordZero;
my $utm_map_y = $real_utm_n - $yCoordZero;
print "map coordinate UTM x: ".$utm_map_x."\nmap
    coordinate UTM y: ".$utm_map_y."\n";
my $i_x = sprintf("%.0f", $utm_map_x);
my $i_y = sprintf("%.0f", $utm_map_y);
print ($i_x." ".$i_y."\n");
```

Per eseguire questa funzionalità lo script necessita dei seguenti file:

- Data.txt : file generate day tools SUMO/VERGILIUS
- latitudine
- lognitudine

- time zone

comando:

```
perl find_coord.pl /percorso/di/Data.txt/file LAT LONG
    TimeZone
```

A.4 Configurazione e codice simulazione

Le caratteristiche tecniche configurate per la simulazione e il codice della simulazione è elencato di seguito:

```
//
//   ndn_v2v_simulation.cpp
//
//
//   Created by Chiara Contoli on 25/02/13.
//
//

//#include "ndn_v2v_simulation.h"

#include <boost/shared_ptr.hpp>
#include <boost/make_shared.hpp>
#include <boost/lexical_cast.hpp>
#include <boost/tokenizer.hpp>

#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/mobility-module.h"
#include "ns3/config-store-module.h"
#include "ns3/wifi-module.h"
#include "ns3/internet-module.h"
#include "ns3/ndnSIM-module.h"
#include "ns3/corner-propagation-loss-model.h"
#include "ns3/ndn-v2v-net-device-face.h"

#include <iostream>
```

```
#include <fstream>
#include <vector>
#include <string>
#include <map>

/*Test traces*/
#include "../src/ndnSIM/utils/tracers/ndn-l3-aggregate
   -tracer.h"
#include "../src/ndnSIM/utils/tracers/ndn-l3-rate-
   tracer.h"
#include "../src/ndnSIM/utils/tracers/v2v-tracer.h"

////./waf --run="ndn_v2v_simulation --numNodes=357 --
   mobilityFile=/home/nrl/VMF/maps_utils/
   wilshire_1lane_50flow_fixed-src-dst.mobility"
NS_LOG_COMPONENT_DEFINE ("Wifi_v2v");

using namespace ns3;
#define MAX_NODES 1002

static int packet_count=0;

//Variable for simulation-traces

//----------------------------------------------//

int mac_tx_count = 0; //counts how many packets are
   received from the higher layers in order to be
   enqueued for transmission
int mac_tx_drop = 0; //counts how many packets are has
    been dropped in the Mac layer before beeing queued
    for transmission
int mac_rx_count = 0; //counts how many packets are
   received from the phy layer and is being forwarded
   to the local protocol stack
int mac_rx_drop = 0; //counts how many packtes has
   been dropped in the Mac layer after has been passed
```

```
    up from phy layer

//-----------------------------------------//

//Callback for simulation-traces

//-----------------------------------------//

 void MacTxCount (Ptr<const Packet> p){
   mac_tx_count++;
   NS_LOG_DEBUG("[MAC_TX_COUNT] Packet is goig to be
      transmitted... mac_tx_count= " << mac_tx_count);
 }

 void MacTxDrop (Ptr<const Packet> p){
   mac_tx_drop++;
   NS_LOG_DEBUG("[MAC_TX_DROP] Packet has been dropped
       before transmission... mac_tx_drop= " <<
      mac_tx_drop);
 }

 void MacRxCount (Ptr<const Packet> p){
   mac_rx_count++;
   NS_LOG_DEBUG("[MAC_RX_COUNT] Packet has been
      received, going to forward... mac_rx_drop= " <<
      mac_rx_count);
 }

 void MacRxDrop (Ptr<const Packet> p){
   mac_rx_drop++;
   NS_LOG_DEBUG("[MAC_RX_DROP] Packet has been dropped
       by Mac before forward... mac_rx_drop= " <<
      mac_rx_drop);
 }

 void PrintTrasmission (){
 //std::cout<<Simulator::Now().GetSeconds()<<"\t"<<
```

```
    mac_tx_count<<"\t"<<mac_tx_drop<<"\t"<<
    mac_rx_count<<"\t"<<mac_rx_drop<<"\n";
  NS_LOG_DEBUG ("[PRINT-TRANSMISSION] Time= "<<
      Simulator::Now().GetSeconds()<<" Mac_tx_count=
      "<<mac_tx_count<<" Mac_tx_drop= "<<mac_tx_drop
      <<" Mac_rx_count= "<<mac_rx_count<<" Mac_rx_drop
      = "<<mac_rx_drop);
  Simulator::Schedule(Seconds(5.0), &PrintTrasmission
      );
}

void OutInterest (Ptr<const ns3::ndn::InterestHeader>
   interestHeader, Ptr<const ns3::ndn::Face> face){
        NS_LOG_DEBUG ("[OUT_INTEREST] Time= "<<
            Simulator::Now().GetSeconds()<<" NID= "<<
            face->GetNode()->GetId()<<" Nonce= "<<
            interestHeader->GetNonce()<<" Name= "<<
            interestHeader->GetName()<<"\n");
}

void InInterest (Ptr<const ns3::ndn::InterestHeader>
   interestHeader, Ptr<const ns3::ndn::Face> face){
        NS_LOG_DEBUG ("[IN_INTEREST] Time= "<<
            Simulator::Now().GetSeconds()<<" NID= "<<
            face->GetNode()->GetId()<<" Nonce= "<<
            interestHeader->GetNonce()<<" Name= "<<
            interestHeader->GetName()<<" Position= "<<
            face->GetNode()->GetObject<MobilityModel>
            ()->GetPosition()<<"\n");
}

void DropInterest (Ptr<const ns3::ndn::InterestHeader>
    interestHeader, Ptr<const ns3::ndn::Face> face){
        NS_LOG_DEBUG ("[DROP_INTEREST] Time= "<<
            Simulator::Now().GetSeconds()<<" NID= "<<
            face->GetNode()->GetId()<<" Nonce= "<<
            interestHeader->GetNonce()<<" Name= "<<
```

```
            interestHeader->GetName()<<"\n");
}

void InData (Ptr<const ns3::ndn::ContentObjectHeader>
   contentHeader, Ptr<const ns3::Packet> packet, Ptr<
   const ns3::ndn::Face> face){
        NS_LOG_DEBUG ("[IN_DATA] Time= "<<Simulator::
           Now().GetSeconds()<<" NID= "<<face->GetNode
           ()->GetId()<<" Name= "<<contentHeader->
           GetName()<<" Position= "<<face->GetNode()->
           GetObject<MobilityModel> ()->GetPosition()
           <<"\n");
}

void OutData (Ptr<const ns3::ndn::ContentObjectHeader>
    contentHeader, Ptr<const ns3::Packet> packet, bool
    value, Ptr<const ns3::ndn::Face> face){
        NS_LOG_DEBUG ("[OUT_DATA] Time= "<<Simulator::
           Now().GetSeconds()<<" NID= "<<face->GetNode
           ()->GetId()<<" Name= "<<contentHeader->
           GetName()<<"\n");
}

void DropData (Ptr<const ns3::ndn::ContentObjectHeader
   > contentHeader, Ptr<const ns3::Packet> packet, Ptr
   <const ns3::ndn::Face> face){
        NS_LOG_DEBUG ("[DROP_DATA] Time= "<<Simulator
           ::Now().GetSeconds()<<" NID="<<face->
           GetNode()->GetId()<<" Name= "<<
           contentHeader->GetName()<<"\n");
}

Ptr<ndn::NetDeviceFace> V2vNetDeviceFaceCallback (Ptr<
   Node> node, Ptr<ndn::L3Protocol> ndn, Ptr<NetDevice
   > device){
        //NS_LOG_UNCOND ("Creating ndn::
           V2vNetDeviceFace on node " << node->GetId()
```

```
            );
        Ptr<ndn::NetDeviceFace> face = CreateObject<
            ndn::V2vNetDeviceFace> (node, device);
        NS_LOG_DEBUG ("Node= " << node->GetId() << "
            MAC= " << device->GetAddress());
        ndn->AddFace (face);
        return face;
}

void printPosition(Ptr<const MobilityModel>  mobility)
{
    NS_LOG_INFO("Time= "<<Simulator::Now().GetSeconds
        () <<" Position of:"<< mobility->GetObject<Node
        >()->GetId()<< ":" <<mobility->GetPosition());
}


//-----------------------------------------------------------------//

void ReceivePacket (Ptr<Socket> socket)
{
    //NS_LOG_UNCOND ("Received one packet!");
    packet_count++;
}
int main (int argc, char *argv[])
{
    //NS_LOG_UNCOND("NDN SIMULATION ------> Started
        <-----");
    std::string phyMode ("OfdmRate6Mbps"); //("
        DsssRate1Mbps");
    double rss = -80;  // -dBm
    uint32_t packetSize = 512; // bytes
    uint32_t numPackets = 50;
    //double interval = 0.5; // seconds
    bool verbose = false;

    //Param for simulation and tracing
```

```
bool pcapOn = true; //Tracing is enabled by
   default
uint32_t numberOfConsumer = 1; //Dafault value: 1
uint32_t numberOfProducer = 1; //Default value: 1
uint32_t numberOfRuns = 50; //Default value: 50
uint32_t numOfNodes = 357; //Default value: 12
std::string mobFile ("/home/nrl/contolic/vndn-sim/
   ns-3/scratch/mob2bis_sim.mobility");
double time = 0;
double x=0,y=0,z=0;
int nodeNo = 0;

CommandLine cmd;

cmd.AddValue ("phyMode", "Wifi Phy mode", phyMode)
   ;
cmd.AddValue ("rss", "received signal strength",
   rss);
cmd.AddValue ("packetSize", "size of application
   packet sent", packetSize);
cmd.AddValue ("numPackets", "number of packets
   generated", numPackets);
//cmd.AddValue ("interval", "interval (seconds)
   between packets", interval);
cmd.AddValue ("verbose", "turn on all
   WifiNetDevice log components", verbose);

//new param
cmd.AddValue ("pcapOn", "Turn on/off pcap tracing.
    Tracing is enabled by default", pcapOn);
cmd.AddValue ("numConsumer", "Number of consumer
   in the simulation", numberOfConsumer);
cmd.AddValue ("numProducer", "Number of producer
   in the simulation", numberOfProducer);
cmd.AddValue ("numRun", "Number of run per
   simulation", numberOfRuns);
cmd.AddValue ("mobilityFile", "Insert full path",
```

```
      mobFile);
  cmd.AddValue ("numNodes", "Total number of nodes
      in the simulation", numOfNodes);


  cmd.Parse (argc, argv);
  // Convert to time object
  //Time interPacketInterval = Seconds (interval);

  //  NS_LOG_UNCOND("true");
  // disable fragmentation for frames below 2200
      bytes
  Config::SetDefault ("ns3::WifiRemoteStationManager
      ::FragmentationThreshold", StringValue ("2200")
      );
  // turn off RTS/CTS for frames below 2200 bytes
  Config::SetDefault ("ns3::WifiRemoteStationManager
      ::RtsCtsThreshold", StringValue ("2200"));
  // Fix non-unicast data rate to be the same as
      that of unicast
  Config::SetDefault ("ns3::WifiRemoteStationManager
      ::NonUnicastMode",
                      StringValue (phyMode));
  // enable cache unsolicited Data
  Config::SetDefault ("ns3::ndn::ForwardingStrategy
      ::CacheUnsolicitedData", BooleanValue(true));
  //Read number of cars directly from file
  std::ifstream readNcars;
  std::map <int,int> map_count;
  int count = 0;
  readNcars.open(mobFile.c_str(),std::ios::in);
  while(readNcars>>nodeNo){
      readNcars>>time >> x >> y >> z;
      if(map_count.find(nodeNo) != map_count.end()){
          //cout<<"Key is present: "<<nodeNo<<" is
              present!\n";
          count = map_count[nodeNo];
```

```
        count++;
        map_count[nodeNo] = count;
    }else{
        //cout<<"Key not present: "<<nodeNo<<" is
            not present!\n";
        map_count[nodeNo] = 1;
    }
}
numOfNodes = map_count.size();
NodeContainer c;
c.Create (numOfNodes);
// The below set of helpers will help us to put
    together the wifi NICs we want
WifiHelper wifi;
if (verbose)
{
     wifi.EnableLogComponents (); // Turn on all
        Wifi logging
}
wifi.SetStandard (WIFI_PHY_STANDARD_80211a);//
    WIFI_PHY_STANDARD_80211b

YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::
    Default ();
// This is one parameter that matters when using
    FixedRssLossModel
// set it to zero; otherwise, gain will be added
wifiPhy.Set ("RxGain", DoubleValue (0) );
wifiPhy.Set ("EnergyDetectionThreshold",
    DoubleValue(-93)); //dBm, default value is -96
    dBm

// ns-3 supports RadioTap and Prism tracing
    extensions for 802.11b
wifiPhy.SetPcapDataLinkType (YansWifiPhyHelper::
    DLT_IEEE802_11_RADIO);
```

```
YansWifiChannelHelper wifiChannel;
wifiChannel.SetPropagationDelay ("ns3::
   ConstantSpeedPropagationDelayModel");
    //Proptagation model is CORNER
wifiChannel.AddPropagationLoss("ns3::
   CornerPropagationLossModel");
wifiPhy.SetChannel (wifiChannel.Create ());

// Add a non-QoS upper mac, and disable rate
   control
NqosWifiMacHelper wifiMac = NqosWifiMacHelper::
   Default ();
wifi.SetRemoteStationManager ("ns3::
   ConstantRateWifiManager",
                                  "DataMode",
                                     StringValue (
                                     phyMode),
                                  "ControlMode",
                                     StringValue (
                                     phyMode));
// Set it to adhoc mode
wifiMac.SetType ("ns3::AdhocWifiMac");
NetDeviceContainer devices = wifi.Install (wifiPhy
   , wifiMac, c);

//Create file for the capture
if(pcapOn == true){
     //wifiPhy.EnablePcap ("capture_examp", devices
        );
     //wifiPhy.EnablePcapInternal ("
        capture_examp_node_0", devices.Get(0),true)
        ;
     //wifiPhy.EnablePcapInternal ("
        capture_examp_node_1", devices.Get(0),true)
        ;
}
```

```
//the format of corner.mobility
//NodeNum time x y z
//by reading each line, we can get a detailed
   trace of nodes' mobility
MobilityHelper mobility;
mobility.SetMobilityModel ("ns3::
   WaypointMobilityModel");
mobility.Install (c);
std::ifstream mobileInput;
mobileInput.open(mobFile.c_str(),std::ios::in);
if(!mobileInput.is_open()){
    NS_LOG_ERROR("Invalid map. The path ("<<
       mobFile<<" is wrong. Please check it again
       ");
    return -1;
}
//With map
std::map <int, Ptr<WaypointMobilityModel> >
   wayMobility;
//NS_LOG_UNCOND ("nNodes = "<<nNodes);
uint32_t index_c = 0;
for(std::map<int,int>::iterator it=map_count.begin
   (); it!=map_count.end(); it++, index_c++){
    wayMobility[it->first] = c.Get(index_c)->
       GetObject<WaypointMobilityModel>();
    //cout<<"Key= "<<it->first <<" Value= "<<it->
       second<<"\n";
}

//setting initial position to all the nodes. We
   put all the cars far away from the map (100Km +
    some random value)
double sinkX = 100000;
double sinkY = 100000;
double sinkZ=0;
index_c = 0;
for(std::map<int,int>::iterator it=map_count.begin
```

```
    (); it!=map_count.end(); it++, index_c++){
     Waypoint waypoint(Seconds(0.1),Vector3D(sinkX
        +(index_c*10000),sinkY+(index_c*10000),
        sinkZ));
     //wayMobility[i]->AddWaypoint(waypoint);
     Simulator::Schedule(Seconds(0.1),&
        WaypointMobilityModel::SetPosition,
        wayMobility[it->first],Vector3D(sinkX+(
        index_c*10000),sinkY+(index_c*10000),sinkZ)
        );
}

std::map<int,int> endTime;
for(std::map<int,int>::iterator it=map_count.begin
    (); it!=map_count.end(); it++){
     endTime[it->first]=0;
}


while(mobileInput>>nodeNo)
{
     mobileInput >> time >> x >> y >> z;
     if(time<=0.1) {
         //discarding the position at sec 0; we've
            substituted them with sink coordinates
         continue;
     }
     endTime[nodeNo]=time;

     Waypoint waypoint(Seconds(time),Vector3D(x,y,z
        ));

     //NS_LOG_UNCOND("Reading mobility file Id="<<
        nodeNo<<" time="<<time<<" ("<<x<<","<<y<<")
        ");
     wayMobility[nodeNo]->AddWaypoint(waypoint);
```

```
        /*if (nodeNo == 0)
         {
         wayMobility0->AddWaypoint(waypoint);
         } else if (nodeNo == 1)
         {
         wayMobility1->AddWaypoint(waypoint);
         }*/

    }

  // std::vector< Ptr< Node > >::const_iterator it =
     c.Begin();
  //setting sink coordinates when a car mobility
     terminates
    uint32_t numberOfWaypoint;
    Waypoint point;
    for(std::map<int,int>::iterator it=map_count.begin
       (); it!=map_count.end(); it++){
        numberOfWaypoint = wayMobility[it->first]->
           WaypointsLeft(); //TODO check if it goes to
            the last element
        if(numberOfWaypoint==0){
             continue;
        }
        index_c = 0;
        Simulator::Schedule(Seconds(endTime[it->first
           ]+0.0001),&WaypointMobilityModel::
           SetPosition, wayMobility[it->first],
           Vector3D(sinkX+(index_c*10000),sinkY+(
           index_c*10000),sinkZ) );
    }

    //NS_LOG_UNCOND ("Mobility read!");

    // Install Ndn stack on all nodes
    //NS_LOG_INFO ("Installing Ndn stack");
    ndn::StackHelper ndnHelper;
```

```
  ndnHelper.AddNetDeviceFaceCreateCallback (
     WifiNetDevice::GetTypeId (), MakeCallback (
     V2vNetDeviceFaceCallback));
  //...setting ad hoc forwarding
  ndnHelper.SetForwardingStrategy ("ns3::ndn::fw::
     V2v");
  ndnHelper.SetDefaultRoutes (true);
  ndnHelper.InstallAll ();

 //Test with more than one consumer
 std::ifstream fileConsumer;
 fileConsumer.open("/home/nrl/contolic/vndn-sim/ns
    -3/scratch/consumer.txt",std::ios::in);
 if(!fileConsumer.is_open()){
      NS_LOG_ERROR("Invalid name. The name "<<
         fileConsumer<<" is wrong. Please check it
         again");
      return -1;
  }
  int idConsumer = 0;
  int countCons = 0;
  int indexCNodes = numOfNodes - 1;
  while(fileConsumer>>idConsumer){
     //NS_LOG_UNCOND("idConsumer " << idConsumer);
     std::string consumerName = "c"+idConsumer;
     Ptr<Node> nodeConsumer = CreateObject<Node>();
     //Names::Add(consumerName, nodeConsumer);
     ndn::AppHelper consumerHelper ("ns3::ndn::
        ConsumerCbr");
     consumerHelper.SetPrefix ("/prefix");
     consumerHelper.SetAttribute ("Frequency",
        StringValue ("1"));
     consumerHelper.SetAttribute ("Randomize",
        StringValue ("uniform"));
     ApplicationContainer consumers = consumerHelper
        .Install (c.Get (indexCNodes - countCons));
     consumers.Start (Seconds (50)); //(50) -
```

```
        countCons*10
    consumers.Stop  (Seconds (450)); //(600) - 30
    countCons++;
}

//Test with more than one producer
std::ifstream fileProducer;
fileProducer.open("/home/nrl/contolic/vndn-sim/ns
    -3/scratch/producer.txt",std::ios::in);
if(!fileProducer.is_open()){
     NS_LOG_ERROR("Invalid name. The name "<<
        fileProducer<<" is wrong. Please check it
        again");
     return -1;
}

int idProducer = 0;
int countProd = 0;
int indexPNodes = 0;
while(fileProducer>>idProducer){
    //NS_LOG_UNCOND("idProducer " << idProducer);
    std::string producerName = "c"+idProducer;
    Ptr<Node> nodeProducer = CreateObject<Node>();
    //Names::Add(producerName, nodeProducer);
    ndn::AppHelper producerHelper ("ns3::ndn::
        Producer");
    producerHelper.SetPrefix ("/prefix");
    producerHelper.SetAttribute ("PayloadSize",
        StringValue("1024"));
    ApplicationContainer producers = producerHelper
        .Install (c.Get (indexPNodes + countProd));
    //producers.Start (Seconds (countProd*0.5));
    countProd++;
}

/*ndn::AppHelper producerHelper ("ns3::ndn::
    Producer");
```

```
// Producer will reply to all requests starting
   with /prefix
producerHelper.SetPrefix ("/prefix");
producerHelper.SetAttribute ("PayloadSize",
   StringValue("1024"));

//ApplicationContainer producers = producerHelper.
   Install (c.Get (0)); // Almost at Lincoln Blvd
   on Wilshire Blvd*/

//NS_LOG_UNCOND ("Application installed!!");

//Trace of level 2.5
Config::ConnectWithoutContext("/NodeList/*/
   DeviceList/*/$ns3::WifiNetDevice/Mac/MacTx",
   MakeCallback(&MacTxCount));
Config::ConnectWithoutContext("/NodeList/*/
   DeviceList/*/$ns3::WifiNetDevice/Mac/MacRx",
   MakeCallback(&MacRxCount));
Config::ConnectWithoutContext("/NodeList/*/
   DeviceList/*/$ns3::WifiNetDevice/Mac/MacTxDrop
   ", MakeCallback(&MacTxDrop));
Config::ConnectWithoutContext("/NodeList/*/
   DeviceList/*/$ns3::WifiNetDevice/Mac/MacRxDrop
   ", MakeCallback(&MacRxDrop));

//Trace
Config::ConnectWithoutContext("/NodeList/*/$ns3::
   ndn::ForwardingStrategy/OutInterests",
   MakeCallback(&OutInterest));
Config::ConnectWithoutContext("/NodeList/*/$ns3::
   ndn::ForwardingStrategy/InInterests",
   MakeCallback(&InInterest));
Config::ConnectWithoutContext("/NodeList/*/$ns3::
   ndn::ForwardingStrategy/DropInterests",
   MakeCallback(&DropInterest));
Config::ConnectWithoutContext("/NodeList/*/$ns3::
```

```
    ndn::ForwardingStrategy/InData", MakeCallback(&
    InData));
Config::ConnectWithoutContext("/NodeList/*/$ns3::
    ndn::ForwardingStrategy/DropData", MakeCallback
    (&DropData));
Config::ConnectWithoutContext("/NodeList/*/$ns3::
    ndn::ForwardingStrategy/OutData", MakeCallback
    (&OutData));

/**Tracing the car position*/
Config::ConnectWithoutContext("/NodeList/*/$ns3::
    MobilityModel/CourseChange", MakeCallback(&
    printPosition));


Simulator::Stop (Seconds(600.0)); //(600.0) - 30


/*Testing traces*/
/*boost::tuple< boost::shared_ptr<std::ostream>,
    std::list<Ptr<ndn::L3AggregateTracer> > >
  aggTracers = ndn::L3AggregateTracer::InstallAll
     ("/mnt/space/simulation_log/ndn-600s-aggregate
     -trace-v2v.txt", Seconds (0.5));*/

/*boost::tuple< boost::shared_ptr<std::ostream>,
    std::list<Ptr<ndn::L3RateTracer> > >
  rateTracers = ndn::L3RateTracer::InstallAll ("/
     mnt/space/simulation_log/ndn-600s-rate-trace-
     v2v.txt", Seconds (0.5));*/

/*   boost::tuple< boost::shared_ptr<std::ostream>,
     std::list<boost::shared_ptr<ndn::V2vTracer> >
   >
  v2vtracing = ndn::V2vTracer::InstallAll ("v2v.
     tracer.txt");*/
```

```
    Simulator::Run ();
    Simulator::Destroy ();

    return 0;
}
```

A.5 Analisi dei dati

Il seguente script si occupa di estrapolare dal file di log della simulazione solamente le informazioni oggetto di interesse, che poi saranno analizzare in seguito da un altro sript.

```
#Created by Chiara Contoli
#This script will extract information of interest by
   the log file, and will write a new log with less
   information

import sys
import string

#open the log file: name of the file to open will be
   passed as first argument
log_file = open(sys.argv[1], 'r')

#open a new log file where only informations of
   interest will be saved
new_log_file = open(sys.argv[2], 'w')

s = log_file.readline()
while s != '':
        str_split = string.split(s)
        #scan file: searching information of interests
        if len(str_split) != 0 and str_split[0] == 'D
           :':
                        new_log_file.write(s)
        if len(str_split) >= 6:
                app = string.split(str_split[2], ":")
```

```
if str_split[4] == 'Node=' or
    str_split[4] == '[IN_INTEREST]' or
    str_split[4] == '[OUT_INTEREST]' or
     str_split[4] == '[IN_DATA]' or
    str_split[4] == '[OUT_DATA]' or
    str_split[4] == '[DROP_INTEREST]'
    or str_split[4] == '[DROP_DATA]' or
     str_split[4] == '[DROP]' or
    str_split[4] == '[ON_DATA]' or
    str_split[4] == '[SATISFACTION]' or
     str_split[4] == '[
    FORWARDING_INTEREST]' or str_split
    [4] == '[FORWARDING_DATA]' or
    str_split[4] == '[DISTANCE-CHECK]'
    or str_split[4] == '[
    SATISFYING_DATA]':
        new_log_file.write(s)
else:
        #Check if is a line of the log
            of Producer or Consumer
        if len(str_split) >= 3:
                application = string.
                    split(str_split[2],
                    ":")
        if application[0] == 'ndn.
            Producer' or (application
            [0] == 'ndn.Consumer' and
            str_split[5] != 'Got'):
                        new_log_file.
                            write(s) #
                            Information
                             about
                            Producer or
                             Consumer:
                            save it!
        else:
                if application[0] == '
```

```
                                                ndn.
                                                V2vNetDeviceFace'
                                                and (str_split[4]
                                                == '[MAC-SENDER]'
                                                or str_split[4] ==
                                                '[POSITION-CHECK]')
                                                :
                                                     new_log_file.
                                                        write(s)
                                             elif application[0] ==
                                                 'ndn.pit.PitImpl'
                                                and application[1]
                                                == 'Create()':
                                                     new_log_file.
                                                        write(s)
        s = log_file.readline()
print 'File ended\n'
```

Per eseguire lo script:

```
python create_new_log.py /percorso/del/file/di/log/
   nome_file.txt nuovo_file_di_log.txt
```

Lo script utilizzato per l'elaborazione dei dati è il seguente:

```
#Created by Chiara Contoli
#This script will read the information of interest
   from the (small) log and will perform some
   statistics
'''
This script was designed to analyze the output of each
    simulation in order to have some statistics about
   the number of hops made by Interest, Content and
   the time elapsed for satisfying each request.

The script needs as input the log file to be analyzed,
    and the path of all the files that will be used to
    store all the informations gathered by running the
    script.
```

```
The log file is read from the first line to the last
   one, and is read in a way that allow to gather all
   the informations needed. Some line are voluntarily
   ignored.

The command line parameter must be passed in the
   following order, path of:
- log file to analyze;
- file in which statistics of hops made by Interest
   will be saved;
- file in which cdf of previous computation will be
   saved;
- file in which data about time satisfaction will be
   saved;
- file in which cdf of previuos computation will be
   saved;
- file in which for each consumer-prefix, time elapsed
    will be save in crescent order;
- file in which statistics of hops made by Content
   will be saved;
- file in which cdf of previous computation will bw
   saved;
'''

import sys
import string
import fpformat
import gc

log_file = open(sys.argv[1], 'r')

dictionary_mac = {} #keep memory of node and their MAC
    address (key is node id, value is MAC address)
dictionary_path = {} #keep memory of all the nodes
   reached by an Interest (for each Interest, keep
   tracks of the list of the receiver; key is Nonce,
   value is node id)
```

```
dictionary_node = {} #keep memory of the nodes (their
    id) traversed by each Nonce (need as en help)
dictionary_node_forwarder = {} #keep memory of ...
dictionary_count_retx = {} #keep memory of re-
    trasmission for each Interest (for each prefix name
    , tells the number of retransmission; key is prefix
     name, value is numer of retransmission)
dictionary_forwarder_interest = {} #keep memory of
    nodes that forward an interest (for each Interest,
    keep track of the list of the forwarder; key is
    nonce, value is all information gathered)

count_retx = 0 #count re-transmission
hop_node = []   #all informationa about the tracking
hop_id = [] #just information about the id
hop_forwarder_i = [] #information about forwarder of
    interest

mapping_name_nonce = {} #keep memory of the nonce used
     for each name (nonce used for each Interest; key
    is prefix name, value is list of Nonce used for
    that prefix)
tmp_name_nonce = [] #temp list of nonce used for each
    Interest

list_of_content_issued = [] #list of the name of all
    the Content that has been issued by the Producer
list_of_interest_satisfied = [] #list of the name of
    all the Interest that has been satisfied

''''FOR MULTI_PATH: '''
mapping_mac_node = {} ##keep memory of node and their
    MAC address (this time, key is MAC id, value is
    node id)
dictionary_nonce_list = {} #for each Nonce, realize
    the graph to be traversed for searching all
    possible path, shortest path from Consumer to each
```

```
    Producer or mule node
dictionary_time_issued_first_interetst = {} #keep
    memory, for each prefix name (and consumer), of the
     time of the first request sent
dictionary_time_content_received = {} #keep memory,
    for each prefix name (and consumer), of the time at
     which a content for a certain prefix is received
shortest_path_info = {} #final result, for each Nonce
    contains all the detailed information about the
    shortest path made by the Interest
current_nonce = '' #nonce currently tracked
temp_list = []
elem_list = []
list_sender = []

content_satisfied_by_producer = {} #keep memory of
    prefix satisfied by Producer and how many times
    they were satisfied by Producer
content_satisfied_by_cache = {} #keep memory of prefix
     satisfied by CS and how many times they were
    satisfied by CS
count_sat = 0
list_of_sender_for_content = {} #tell who has sent
    that prefix (node of receiver + prefix name is the
    key, node id is the value)
list_of_sender_id = []

file_prob_hop = open(sys.argv[2], 'w') #path of the
    file in which statistics about hops will be saved
file_cdf = open(sys.argv[3], 'w') #path of the file in
     which cdf will be saved
file_time_satisfaction = open(sys.argv[4], 'w') #path
    of file on which time of satisfaction will be saved
file_cdf_time = open(sys.argv[6], 'w')

file_prob_hop_data = open(sys.argv[7], 'w') #path of
    the file in which statistics about hops of content
```

```
   will be saved
file_cdf_data = open(sys.argv[8], 'w') #path of the
   file in which cdf of content will be saved

check_hop = 0

'''MOD MULTI-CONSUMER and MULTI-PRODUCER'''
list_of_consumer = ['693', '694', '695', '696'] #keep
   memory of the node id of all the Consumer
list_of_producer = ['0', '1', '2'] #keep memory of the
    node id of all the Producer

consumer_request = {} #keep memory of distinct prefix
   name request by each Consumer (node id is the key,
   list of prefix name is the value)
list_request = [] #prefix request by each consumer
consumer_request_satisfied = {} #keep memory of
   distinct prefix that have been satisfied for each
   consumer (key is Consumer's node id, value is list
   of all prefix name for which a Content has been
   received)
list_request_satisfied = [] #prefix satisfied for each
    consumer

dic_of_number_of_data_hop = {} #keep memory, for each
   prefix, of how many hop have been done (key is
   prefix name, value is the number of hop made by
   Conten, for each Consumer that issued that prefix
   name)
list_of_number_of_data_hop = []
shortest_list_number_of_data_hop = {} #shortest number
    of hop made by each prefix

dictionary_time_satisfaction = {} #keep memory of all
   the times needed to satisfy each request (key is
   Consumer's node id + prefix name, value is the time
    elapse between de first request for that prefix
```

```
   and the Contet for that prefix)

list_nonce_auto_satisfaction = [] #keep memory of the
   list of nonce auto-satisfied by each Consumer
dic_auto_satisfaction = {} #keep memory for each
   Consumer of the nonce auto-satisfied (key is node
   id, value is list of Nonce auto-satisfied by that
   Consumer)

list_nonce_mule_satisfaction = [] #keep memory of the
   list of the nonce satisfied by a mule
dic_mule_satisfaction = {} #for each Nonce, keep track
    of the node id of which node issued that packet (
   key is Nonce, value is list of node id)

s = log_file.readline()

while s != '':
        str_split = string.split(s)
        if len(str_split) == 2 and str_split[0] == 'D
           :': #a content is issued by a Producer
                list_of_content_issued.append(
                   str_split[1]) #keep memory of all
                   the content issued
if len(str_split) == 15 and str_split[2] == 'ndn.
   Consumer:OnContentObject():' and str_split[6] == '
   DATA': #Interest satisfied (a Consumer has received
    a Content)
        full_name = "/prefix/"+ str_split[8] # "save"
           full prefix name
        list_of_interest_satisfied.append(full_name) #
           make a list of all prefix name satisfied
        '''New'''
        key_d = str_split[1] + "-" + full_name #id of
           node Consumer that received Content plus
           prefix name is used as key
```

```
if not dictionary_time_content_received.
   has_key(key_d): #keep memory of the time at
    which the Conten is received....
        str_c_received = str_split[0] #save
           time
        dictionary_time_content_received[key_d
           ] = str_c_received[:-1] #remove (s)
            from the time
#Content tracking
typesat = str_split[12] #check if the Content
   is coming from a ContentStore or from a
   Producer
if typesat == 'CS': #content satisfied by CS
        #Count how many times that content
           comes from a CS
        if not content_satisfied_by_cache.
           has_key(full_name):
                content_satisfied_by_cache[
                   full_name] = 1
        else:
                count_sat =
                   content_satisfied_by_cache[
                   full_name]
                count_sat = count_sat + 1
                content_satisfied_by_cache[
                   full_name] = count_sat
elif typesat == 'Producer': #Content satisfied
    by Producer
        #Count how many times that content
           comes from a Producer
        if not content_satisfied_by_producer.
           has_key(full_name):
                content_satisfied_by_producer[
                   full_name] = 1
        else:
        count_sat =
           content_satisfied_by_producer[
```

```
            full_name]
        count_sat = count_sat + 1
        content_satisfied_by_producer[
            full_name] = count_sat
#keep memory of who sends the content for that
    prefix
if not list_of_sender_for_content.has_key(
    full_name):
        list_of_sender_id.append(str_split[1])
        list_of_sender_for_content[full_name]
            = list_of_sender_id
else:
        list_of_sender_id =
            list_of_sender_for_content[
            full_name]
        list_of_sender_id.append(str_split[1])
        list_of_sender_for_content[full_name]
            = list_of_sender_id
list_of_sender_id = []
'''ENew'''

''''New: for multi-conusmer-multi-producer'''
#keep memory of each prefix name that have
    been satisfied (for each consumer (needed
    to compute the percentage))
if not consumer_request_satisfied.has_key(
    str_split[1]):
        list_request_satisfied.append(
            full_name)
        consumer_request_satisfied[str_split
            [1]] = list_request_satisfied
else:
        list_request_satisfied =
            consumer_request_satisfied[
            str_split[1]]
        if full_name not in
            list_request_satisfied:
```

```
                    list_request_satisfied.append(
                       full_name)
                    consumer_request_satisfied[
                       str_split[1]] =
                       list_request_satisfied
        list_request_satisfied = []

        if not dic_of_number_of_data_hop.has_key(
           full_name):#compute how many hop have been
           done by that Content
                if int(str_split[14]) - 1 == 0: #DEBUG
                   : check if the Content was already
                   present in the CS of the receiver.
                        check_hop = check_hop + 1 #For
                           debug
                list_of_number_of_data_hop.append(int(
                   str_split[14]) - 1) #added -1 (
                   number of hop is number of hop - 1)
                dic_of_number_of_data_hop[full_name] =
                    list_of_number_of_data_hop
        else:
                if int(str_split[14]) - 1 == 0:
                        check_hop = check_hop + 1 #for
                           debug
                list_of_number_of_data_hop =
                   dic_of_number_of_data_hop[full_name
                   ]
                list_of_number_of_data_hop.append(int(
                   str_split[14]) - 1) #added -1 (
                   number of hop is number of hop - 1)
                dic_of_number_of_data_hop[full_name] =
                    list_of_number_of_data_hop
        list_of_number_of_data_hop = []
        ''''ENew: for multi-conusmer-multi-producer'''
       #If there is the following sequence of line,
          it means that the requested DATA was
          already in the Content Store (case of auto-
```

```
            satisfaction)
        s = log_file.readline() #ignore the next line:
            don't care
        s = log_file.readline() #ignore the next line:
            don't care
        s = log_file.readline()
        str_split = string.split(s)
        if str_split[2] == 'ndn.fw:OnInterest():' and
           str_split[4] == '[SATISFACTION]': #auto-
           satisfaction
        if not dic_auto_satisfaction.has_key(str_split
           [1]):
                list_nonce_auto_satisfaction.append(
                   str_split[9]) #10
                dic_auto_satisfaction [str_split[1]] =
                    list_nonce_auto_satisfaction
        else:
                list_nonce_auto_satisfaction =
                   dic_auto_satisfaction [str_split
                   [1]]
                list_nonce_auto_satisfaction.append(
                   str_split[9]) #10
                dic_auto_satisfaction [str_split[1]] =
                    list_nonce_auto_satisfaction
        list_nonce_auto_satisfaction = []
if len(str_split) == 10 and str_split[2] == 'ndn.fw:
   OnInterest():' and str_split[4] == '[SATISFACTION
   ]': #mule (or Producer satisfaction)
        if not dic_mule_satisfaction.has_key(str_split
           [9]):
                if str_split[1] not in
                   list_of_producer and str_split[1]
                   not in list_of_consumer: #exclude
                   Producer case, it's a mule
                   satisfaction
                        list_nonce_mule_satisfaction.
                           append(str_split[1])
```

```
                    dic_mule_satisfaction[
                       str_split[9]] =
                       list_nonce_mule_satisfaction

        else:
                list_nonce_mule_satisfaction =
                   dic_mule_satisfaction[str_split[9]]
                if str_split[1] not in
                   list_nonce_mule_satisfaction and
                   str_split[1] not in
                   list_of_producer and str_split[1]
                   not in list_of_consumer: #if not
                   already present and node not a
                   producer
                        list_nonce_mule_satisfaction.
                           append(str_split[1])
                        dic_mule_satisfaction[
                           str_split[9]] =
                           list_nonce_mule_satisfaction

        list_nonce_mule_satisfaction = []
        if len(str_split) >=7:
        if str_split[4] == 'Node=' and str_split[6] ==
            'MAC=': #keep memory of Nodes and their
           MAC
                if not dictionary_mac.has_key(
                   str_split[5]): #adding Node_id:
                   mac_address
                        dictionary_mac[str_split[5]] =
                            str_split[7]
                '''New'''
                if not mapping_mac_node.has_key(
                   str_split[7]):
                        mapping_mac_node[str_split[7]]
                            = str_split[5] #key is
                           mac_address, value is node
                           id
```

```
        '''E-New'''
elif str_split[6] == 'Interest' and str_split
   [4] != '[DROP]': #keep memory of potential
   re-transmission
last_part_of_name = str_split[8]
name = "/prefix/"+last_part_of_name

''''New: for multi-conusmer-multi-producer'''
if not consumer_request.has_key(str_split[1])
   :# for each Consumer, mem all the prefix
   sent (just once)
        list_request.append(name)
        consumer_request[str_split[1]] =
           list_request
else:
        list_request = consumer_request[
           str_split[1]]
        if name not in list_request:
                list_request.append(name)
                consumer_request[str_split[1]]
                   = list_request
list_request = []
''''ENew: for multi-conusmer-multi-producer'''

'''New'''
key = str_split[1] + "-" + name
#mem time of first transmission for that
   prefix
if not dictionary_time_issued_first_interetst.
   has_key(key):
        str_f_time = str_split[0]
        dictionary_time_issued_first_interetst
           [key] = str_f_time[:-1]
'''ENew'''
if not dictionary_count_retx.has_key(name): #
   compute the number of retransmission for
   each prefix
```

```
            count_retx = count_retx + 1
            dictionary_count_retx[name] =
               count_retx
            count_retx = 0
    else:
            count_retx = dictionary_count_retx[
               name]
            count_retx = count_retx + 1
            dictionary_count_retx[name] =
               count_retx
            count_retx = 0
    s = log_file.readline() #ignore the next line:
        don't care
    s = log_file.readline() #ignore the next line:
        don't care
    s = log_file.readline()
    str_split = string.split(s)
    if str_split[4] == '[IN_INTEREST]' and
       str_split[9] == 'Nonce=': #tracking
       Interest: it's the Consumer request
            '''New'''
            current_nonce = str_split[10]
            dictionary_nonce_list[current_nonce] =
                {str_split[1]:[]} #starting
               creating graph
            #collect detailed information...
            time = str_split[6] #time at which
               interest is received
            node = str_split[8] #node id of the
               receiver
            hop_id.append(node)
            dictionary_node[str_split[10]] =
               hop_id #keep track of all the node
               id that received that Interest
            hop_id = []
            mac_sender = dictionary_mac[str_split
               [8]] #who sent this interest (in
```

```
    this case, is the consumer)...
mac_receiver = mac_sender #...that is
    also the receiver
pos = string.split(str_split[14],":")
    #position of the node that received
     the interest
posx = pos[0]
posy = pos[1]
posz = pos[2]
list_detail = []
elem = time+"*"+mac_receiver+"*"+
    mac_sender+"*"+repr(posx)+"*"+repr(
    posy)+"*"+repr(posz) #put together
    all the information
list_detail.append(elem)
'''ENew'''
if not mapping_name_nonce.has_key(
    str_split[12]): #new name of
    Interest encountered
        tmp_name_nonce.append(
            str_split[10])
        mapping_name_nonce[str_split
            [12]] = tmp_name_nonce
else:
        tmp_name_nonce =
            mapping_name_nonce[
            str_split[12]]
        if str_split[10] not in
            tmp_name_nonce:
                tmp_name_nonce.append(
                    str_split[10])
                mapping_name_nonce[
                    str_split[12]] =
                    tmp_name_nonce
tmp_name_nonce = []
elem = ""
hop_id = []
```

```
        elif str_split[4] == '[OUT_INTEREST]':
            #tracking forwarder of each
            Interest
        node = str_split[1] #id of the node
            that forward the Interest
        time = str_split[6] #time at which the
             content is forwarded
        s = log_file.readline()
        str_split = string.split(s)
        if str_split[4] == '[
            FORWARDING_INTEREST]':
if len(str_split) == 19: #it's not the
   consumer, so mac_in_face is present (if ==
   17, it's the consumer (or producer), and
   mac_in_face is missing)
        mac_in_face = str_split[12] #mem the
            mac of the sender
        mac_out_face = str_split[16] #mem the
            mac of the receiver
        pos = string.split(str_split[18],":")
            #mem the position of the forwarder
        posx = pos[0]
        posy = pos[1]
        posz = pos[2]
else:
        #if it is the consumer, mac of the
            sender is not present (if needed,
            keep it form the mapping between id
             node and mac address, but works
            with just one face)
        if node in list_of_consumer:
                mac_in_face = ""
                mac_out_face = str_split[14] #
                    mac of the forwarder
        else:
                mac_in_face = str_split[12]
                mac_out_face = ""
```

```
          pos = string.split(str_split[16],":")
          posx = pos[0]
          posy = pos[1]
          posz = pos[2]
    elem = time+"*"+mac_in_face+"*"+mac_out_face
       +"*"+repr(posx)+"*"+repr(posy)+"*"+repr(
       posz) #posx_y_z: position of the forwarder
    if not dictionary_forwarder_interest.has_key(
       str_split[8]): #new Interest
          hop_forwarder_i.append(elem) # mem who
              forward that interest with that
             Nonce
          dictionary_forwarder_interest[
             str_split[8]] = hop_forwarder_i
    else:
    hop_id = dictionary_node_forwarder[str_split
       [8]]
    if node not in hop_id: #the node is not in the
        path yet
          hop_forwarder_i =
             dictionary_forwarder_interest[
             str_split[8]]
          hop_forwarder_i.append(elem)
          dictionary_forwarder_interest[
             str_split[8]] = hop_forwarder_i
    #update the list of id node traversed by a
       certain nonce
    hop_id.append(node)
    dictionary_node_forwarder[str_split[8]] =
       hop_id
    hop_forwarder_i = []
    hop_id = []
    elif str_split[4] == '[POSITION-CHECK]': #used
        to keep track of detailed information
    pos = string.split(str_split[7],":")
    posx = pos[0]
    posy = pos[1]
```

```
posz = pos[2]
'''New'''
str_f_time = str_split[0]
time = str_f_time[:-1]
mac_receiver = dictionary_mac[node]
elem = time+"*"+mac_receiver+"*"+mac_sender
   +"*"+repr(posx)+"*"+repr(posy)+"*"+repr(
   posz)
#temp_detail_information =
   dictionary_detail_information[current_nonce
   ]
tmp_dict_node = dictionary_nonce_list[
   current_nonce] #used later to avoid insert
   information due to loop
if tmp_dict_node.has_key(str_split[1]):
        temp_list = tmp_dict_node[str_split
           [1]]
        if mapping_mac_node[mac_sender] in
           temp_list and temp_list.index(
           mapping_mac_node[mac_sender]) == (
           len(temp_list)-1):
                if len(elem_list) == (len(
                   temp_list) - 1): #
                   information not added yet
                        elem_list.append(elem)
elem_list = []
'''ENew'''
elif str_split[4] == '[DISTANCE-CHECK]':
        #TODO
        what_to_do = ''
elif len(str_split) != 0 and str_split[4] ==
   '[IN_INTEREST]': #tracking Interest
hop_id = dictionary_node[str_split[10]]
if node not in hop_id: #the node is not in the
    path yet
        #update the the list of id node
           traversed for a certain nonce
```

```
            hop_id.append(node)
            dictionary_node[str_split[10]] =
               hop_id
      hop_node = []
      elem = ""
      hop_id = []
      elif len(str_split) >=6 and str_split[4] == '[
         MAC-SENDER]' and str_split[6] == 'Nonce=':
         #Interest tracking
      '''New'''
      temp_dict_node = dictionary_nonce_list[
         str_split[7]] #get the dictionary relative
         the current nonce
      if not temp_dict_node.has_key(str_split[1]):
            elem_list.append(mapping_mac_node[
               str_split[5]]) #create the list of
               node for the new element
            temp_dict_node[str_split[1]] =
               elem_list #add the new element to
               the dictionary
      else:
            temp_list = temp_dict_node[
               mapping_mac_node[str_split[5]]]
            if str_split[1] not in temp_list: #
               avoid loop
                  list_sender = temp_dict_node[
                     str_split[1]]
                  if mapping_mac_node[str_split
                     [5]] not in list_sender: #
                     avoid to insert node
                     already in the path
                        list_sender.append(
                           mapping_mac_node[
                           str_split[5]])
                        temp_dict_node[
                           str_split[1]] =
                           list_sender
```

```
                temp_list = []
                elem_list = []
                '''ENew'''
                mac_sender = str_split[5]
                node = str_split[1]
                current_nonce = str_split[7]

        s = log_file.readline()

#print 'File ended', '\n', '\n'
log_file.close()

del tmp_dict_node

#sorted_dictionary_mac = sorted(dictionary_mac.items()
    , key=lambda t: t[0])
#print 'MAPPING NODE - MAC ADDRESS:', '\n' #IT WORKS
#print dictionary_mac
#print sorted_dictionary_mac
#print '\n'

print 'MAPPING NAME - NONCE:' #IT WORKS
print mapping_name_nonce
print 'Number of Interest Requested (from mapping name
    -nonce): ', len(mapping_name_nonce) #Number of
    Interest requested (without counting retransmission
    )
print '\n'
del mapping_name_nonce

print 'POTENTIAL RE-TRANSMISSION:' #IT WORKS
print dictionary_count_retx
print 'Number of Intrest Requested (from count-
    retrasmission): ', len(dictionary_count_retx)
print '\n'
del dictionary_count_retx
```

```
#print 'INTEREST PATH:' #IT WORKS
#print dictionary_path
#print 'Number of Interest send by the Consumer (not
   sure that reach the Producer): ', len (
   dictionary_path)
#print '\n'

print 'DICTIONARY NODE:'
print dictionary_node
print '\n'
del dictionary_node

#print 'INTEREST FORWARDER:' #IT WORKS
#print dictionary_forwarder_interest
#print '\n'

print 'CONTENT ISSUED:' #IT WORKS
print list_of_content_issued
print 'Number of Content issued by the Producer: ',
   len(list_of_content_issued) #Number of Content
   issued by the Producer (not sure if reach the
   Consumer)
print '\n'
del list_of_content_issued

#print 'SATISFIED INTEREST:'
#print list_of_interest_satisfied
#print 'Number of Interest satisfied: ', len(
   list_of_interest_satisfied) #Number of Interest
   satisfied by the Producer
#print '\n'

print 'PREFIX NAME REQUEST BY EACH CONSUMER:'
print consumer_request
for i in consumer_request:
        print 'Number of distinct prefix sent by
           Consumer ', i, ': ', len(consumer_request[i
```

```
            ])
print '\n'

print 'PREFIX NAME SATISFIED FOR EACH CONSUMER:'
print consumer_request_satisfied
for i in consumer_request_satisfied:
        print 'Number of requests satisfied for
           Consumer ', i, ': ', len(
           consumer_request_satisfied[i])
print '\n'

print 'NONCE AUTO-SATISFIED BY EACH CONSUMER:'
print dic_auto_satisfaction
print '\n'

tot_auto_sat = 0
for i in dic_auto_satisfaction:
        print 'Number of nonce auto-satisfied by
           Consumer ', i, ': ', len(
           dic_auto_satisfaction[i])
        tot_auto_sat = tot_auto_sat + len(
           dic_auto_satisfaction[i])
print 'Check: total number of nonce auto-satisfied: ',
    tot_auto_sat

print 'MULE SATISFACTION:'
print dic_mule_satisfaction
print '\n'

'''New: for multi-conusmer-multi-producer'''
#compute the number of total request
total_request = 0
for i in consumer_request.keys():
        total_request = total_request + len(
           consumer_request[i])
print 'Number of total requests: ', total_request
#compute the number of total request satisfied
```

```
total_request_satisfied = 0
for i in consumer_request_satisfied:
        total_request_satisfied =
            total_request_satisfied + len(
            consumer_request_satisfied[i])
print 'Number of total requests satisfied: ',
   total_request_satisfied
del consumer_request_satisfied
del consumer_request

print 'PERCENTAGE OF SATISFIED INTEREST:'
perc = float(total_request_satisfied) / float(
   total_request)
print "{0:.4f}".format(float(perc))
'''ENew: for multi-conusmer-multi-producer'''

'''New'''
print 'MAPPING NONCE - DICTIONARY LIST:'
print dictionary_nonce_list
print 'Len= ', len(dictionary_nonce_list)
print '\n'

print 'TIME OF FIRST SEND FOR EACH PREFIX:'
print dictionary_time_issued_first_interetst
print 'Number of prefix send: ', len(
   dictionary_time_issued_first_interetst)
print '\n'

print 'TIME AT WHICH CONTETNT GETS TO CONSUMER:'
print dictionary_time_content_received
print 'Number of content arrived: ', len(
   dictionary_time_content_received)
print '\n'

'''print 'DICTIONARY PATH (DETAIL):'
print dictionary_detail_information
print '\n' '''
```

```
'''ENew'''
#==========================================================
#Verify: number of retransmission that is in
   dictionary_count_retx must be equal to the number
   of elements containd in the list
#for each key of the mapping name-conce: the check is
   ok -> same value as expected (dict_verify_retx is
   equal to dictionary_count_retx)
'''dict_verify_retx = {}
num_retx = 0
for i in mapping_name_nonce.keys():
        num_retx = len(mapping_name_nonce[i])
        dict_verify_retx[i] = num_retx
#print 'VERIFY NUMEBER OF RETRANSMISSION:' '''
#print dict_verify_retx

'''NEW COMPUTATION STATISTIC'''
file_prob_hop.write("#Hop\tP(hop)\tOccorrenze\n")
file_cdf.write("#Hop\tCdf\n")

#FUNCTION: the first one find all possible path from a
    source to a destination, the second one find the
   shortest
def find_all_paths(graphI, start, end, path=[]):
        path = path + [start]
        if start == end:
                return [path]
        if not graphI.has_key(start):
                return []
        paths = []
        for node in graphI[start]:
                if node not in path:
                        newpaths = find_all_paths(
                           graphI, node, end, path)
                        for newpath in newpaths:
                                paths.append(newpath)
        return paths
```

```
def find_shortest_path(graphI, start, end, path=[]):
        path = path + [start]
        if start == end:
                return path
        if not graphI.has_key(start):
                return None
        shortest = None
        for node in graphI[start]:
                if node not in path:
                        newpath = find_shortest_path(
                           graphI, node, end, path)
                        if newpath:
                                if not shortest or len
                                   (newpath) < len(
                                   shortest):
                                        shortest =
                                           newpath
        return shortest

#Find all path for each Interest, from Consumer to
   Producer, if present, and then find the shortest
dict_all_possible_path = {} #keep track, for each
   Interest, of all the path from Consumer to Producer
   , if present
dict_all_possible_shortest_path = {} #keep track, for
   each Interest, of the shortest path
found = 'false'
#debug_count = 0
for nonce in dictionary_nonce_list.keys():
found = 'false'
graph = dictionary_nonce_list[nonce]
for i in list_of_consumer:
if graph.has_key(i):
        val = graph[i]
        if len(val) == 0: #consumer: search for all
           possible path to destination
```

```
found = 'true'
for j in list_of_producer:
#print 'result= ',dict_all_possible_path[nonce
   ]
shortest_result = find_shortest_path(graph, j,
    i)
if not dict_all_possible_shortest_path.has_key
   (nonce):
        dict_all_possible_shortest_path[nonce]
           = {j:shortest_result}
else:
e_short = dict_all_possible_shortest_path[
   nonce]
e_short[j] = shortest_result
#check if the content was satisfied by
   Consumer itself (auto-satisfied)
if dic_auto_satisfaction.has_key(i):
tmp_list = dic_auto_satisfaction[i]
        if nonce in tmp_list:
        shortest_result = find_shortest_path(
           graph, i, i)
        if not dict_all_possible_shortest_path
           .has_key(nonce):
                dict_all_possible_shortest_path
                   [nonce] = {i:
                   shortest_result}
        else:
        e_short =
           dict_all_possible_shortest_path[
           nonce]
        e_short[i] = shortest_result
        dic_auto_satisfaction[i].remove(nonce)

        #check if the content was satisfied by
            a Mule
        if dic_mule_satisfaction.has_key(nonce
           ):
```

```
                list_of_mule = dic_mule_satisfaction[
                   nonce]
                for j in list_of_mule:
                shortest_result = find_shortest_path(
                   graph, j, i)
                if not dict_all_possible_shortest_path
                   .has_key(nonce):
                dict_all_possible_shortest_path[nonce]
                    = {j:shortest_result}
                else:
                e_short =
                   dict_all_possible_shortest_path[
                   nonce]
                e_short[j] = shortest_result
                del dic_mule_satisfaction[nonce]

                if found == 'true':
                        del dictionary_nonce_list[
                           nonce]
                        break
del dictionary_nonce_list
del dic_mule_satisfaction
del dic_auto_satisfaction

#print 'ALL POSSIBLE PATH FOR EACH INTEREST: '
#print dict_all_possible_path
#print 'Len(dict_all_possible_path)= ', len(
   dict_all_possible_path)
#print '\n'

print 'SHORTEST PATH FOR EACH INTEREST (All shortest
   path per nonce to each destination):'
print dict_all_possible_shortest_path
print 'Len(dict_all_possible_shortest_path)= ', len(
   dict_all_possible_shortest_path)
print '\n'
```

```
#print 'Length of previous dictionary: ', len(
   dic_all_shortest_path_to_p)
#Find the shortest between all the shortest
shortest = {}
array_key = []
array_len = []
min_len = 0
nearer_p = 0
l = None
for nonce in dict_all_possible_shortest_path.keys():
        l = dict_all_possible_shortest_path[nonce]
        for i in l.keys():
                e = l[i]
                #print 'e= ', e, ' i= ', i
                if e != None:
                        array_key.append(i)
                        array_len.append(len(e))
        if len(array_len) != 0:
                #print 'len(array_len)= ', len(
                   array_len)
                interval = (1, len(array_len)-1)
                min_len = array_len[0]
                nearer_p = array_key[0]
                for j in interval:
                        if len(array_len) != 1:
                                if array_len[j] <=
                                   min_len:
                                        min_len =
                                           array_len[j
                                           ]
                                        nearer_p =
                                           array_key[j
                                           ]
                shortest[nonce] = l[nearer_p]
        array_key = []
        array_len = []
del dict_all_possible_shortest_path
```

```
del l
print 'THE SHORTEST: ', shortest
print 'len(shortest)= ', len(shortest)

#Counting hop based on shortest path
occorrenze = {}
count = 0
hop_count = 0
prob_hop = {}
for i in shortest.keys():
        elem = shortest[i]
        hop_count = len(elem) - 1
        if not occorrenze.has_key(hop_count):
                occorrenze[hop_count] = 1
        else:
                count = occorrenze[hop_count]
                count = count + 1
                occorrenze[hop_count] = count
del shortest

max_intervals = 101
index = range(0, max_intervals)
for i in index:
        if not occorrenze.has_key(i):
                occorrenze[i] = 0
somma = 0
for i in index:
        somma = somma + occorrenze[i]
#compute probability of each number of hops
for i in index:
        prob_hop[i] = float(occorrenze[i])/float(somma
           )
        file_prob_hop.write(repr(i)+"\t"+"{0:.4f}".
           format(float(prob_hop[i]))+"\t"+repr(
           occorrenze[i])+"\n")
file_prob_hop.close()
```

```
#Verify: sum of probability must be 1
tot = 0
index = range(0, len(prob_hop))
for i in index :
        tot = tot + prob_hop[i]
print "Verify sum of probability: ", tot, "\n"

sum_prob = 0
res = 0
index2 = occorrenze.keys()
index2.sort()
#Compute CDF for Interest hop
for i in index2:
        sum_prob = sum_prob + occorrenze[i]
        res = float(sum_prob) / float(somma)
        file_cdf.write(repr(i)+"\t"+"{0:.4f}".format(
           float(res))+"\n")
file_cdf.close()

#Compute time of satisfaction
file_time_satisfaction.write("Consumer\tInterest\
   tFirstSend\tReceivedAt\tTimeSatisfaction\n")
for i in dictionary_time_content_received.keys():
        who = string.split(i, "-")
        time_elapsed = float(
           dictionary_time_content_received[i]) -
           float(
           dictionary_time_issued_first_interetst[i])
        file_time_satisfaction.write(who[0]+"\t"+who
           [1]+"\t"+"{0:.4f}".format(float(
           dictionary_time_issued_first_interetst[i]))
           +"\t"+"{0:.4f}".format(float(
           dictionary_time_content_received[i]))+"\t
           "+"{0:.4f}".format(float(time_elapsed))+"\n
           ")
        dictionary_time_satisfaction[i] = time_elapsed
        val = string.split(who[1], "/")
```

```
        del dictionary_time_content_received[i]
del dictionary_time_content_received
del dictionary_time_issued_first_interetst
file_time_satisfaction.close()

print 'DICTIONARY TIME SATISFACTION:'
print dictionary_time_satisfaction
print 'Len(dictionary_time_satisfaction)= ', len(
   dictionary_time_satisfaction)
print '\n'

temp_file = open(sys.argv[5], 'w')
temp_file.write("PrefixName\tTimeElapsed\n")

for i in dictionary_time_satisfaction.keys():
    temp_file.write(i+"\t"+"{0:.4f}".format(float(
       dictionary_time_satisfaction[i]))+"\n")
temp_file.close()
print 'Len(DICTIONARY TIME SATISFACTION)= ', len(
   dictionary_time_satisfaction)

#Compute CDF for the time of satisfaction
sum_time = 0
print 'Size of dict of time: ', len(
   dictionary_time_satisfaction)
for i in dictionary_time_satisfaction:
        sum_time = sum_time +
           dictionary_time_satisfaction[i]

media = float(sum_time) / len(
   dictionary_time_satisfaction)

print  'M: ', media
print '\n'

delta = float(media) / 10
print 'Delta: ', delta
```

```
occ_time = {}
count_time = 0
for i in dictionary_time_satisfaction:
        val = int(dictionary_time_satisfaction[i] /
           float(delta))
        if val >= max_intervals:
                val = max_intervals - 1
        if not occ_time.has_key(val):
                occ_time[val] = 1
        else:
                count_time = occ_time[val]
                count_time = count_time + 1
                occ_time[val] = count_time
del dictionary_time_satisfaction

index_time = range(max_intervals)
for i in index_time:
        if not occ_time.has_key(i):
                occ_time[i] = 0

somma = 0
for i in index:
        somma = somma + occ_time[i]

print 'Sum occ: ', somma
print '\n'

file_cdf_time.write("Interval \t ti \t DensProb \t
   Prob \tCdf \n")
prob_time = {}
dens_prob_time = {}

for i in index_time:
        prob_time[i] = float(occ_time[i]) / float(
           somma)
```

```
        dens_prob_time[i] = float(prob_time[i]) /
           float(delta)

tot = 0
index = range(0, len(prob_time))
for i in index :
        tot = tot + prob_time[i]
print "Verify sum of probability: ", tot, "\n"

res = 0
index_time2 = occ_time.keys()
index_time2.sort()
sum_prob = 0

#Compute CDF (for time satisfaction)
ti = 0
for i in index_time2:
        ti = (i + 1/2)*delta;
        sum_prob = sum_prob + prob_time[i]
        file_cdf_time.write(repr(i)+" \t "+"{0:.4f}".
           format(float(ti))+" \t "+"{0:.4f}".format(
           float(dens_prob_time[i]))+" \t "+repr(
           prob_time[i])+" \t "+"{0:.4f}".format(float
           (sum_prob))+"\n")
file_cdf_time.close()

print 'LIST OF NUMBER OF HOP MADE BY CONTENT FOR EACH
   PREFIX NAME:'
print dic_of_number_of_data_hop
tot = 0
for i in dic_of_number_of_data_hop:
        tot = tot + len(dic_of_number_of_data_hop[i])
print 'Total: ', tot
print '\n'

#compute the minimum number of hop made by each
   content
```

```
min = 0

for prefix in dic_of_number_of_data_hop.keys():
        l = dic_of_number_of_data_hop[prefix]
        min = l[0]
        interval = range(1, len(l))
        for i in interval:
                if l[i] <= min:
                        min = l[i]
        shortest_list_number_of_data_hop[prefix] = min

#print 'MINIMUM NUMBER OF HOP (FOR CONTENT):'
#print shortest_list_number_of_data_hop
#print 'Len(shortest_list_number_of_data_hop)= ', len(
   shortest_list_number_of_data_hop)
#print '\n'

n_data_by_p = 0 #number of content satisfied by
   Producer
for i in content_satisfied_by_producer:
        n_data_by_p = n_data_by_p +
           content_satisfied_by_producer[i]
print 'CONTENT SATISFIED by Producer: ',
   content_satisfied_by_producer
print 'Number of content satisfied by Producer: ',
   n_data_by_p
print '\n'
del content_satisfied_by_producer

n_data_by_cs = 0 #number of content satisfied by CS
for i in content_satisfied_by_cache:
        n_data_by_cs = n_data_by_cs +
           content_satisfied_by_cache[i]
print 'CONTENT SATISFIED by CS: ',
   content_satisfied_by_cache
print 'Number of content satisfied by Cache: ',
   n_data_by_cs
```

```
print '\n'
del content_satisfied_by_cache


print 'Verify: NCP + NCCS = ', n_data_by_p, '+',
   n_data_by_cs, '= ', n_data_by_p + n_data_by_cs ,'.
   Must be equals to ', total_request_satisfied
print '\n'

perc_cs = float(n_data_by_cs) / float(
   total_request_satisfied) #percentage of content
   satisfied by CS
perc_p = float(n_data_by_p) / float(
   total_request_satisfied) #percentage of content
   satisfied by Producer
print 'Percentage of Content by CS: ', "{0:.4f}".
   format(float(perc_cs))
print 'Percentage of Content by Producer: ', "{0:.4f
   }".format(float(perc_p))

print 'List of who has issued that prefix: ',
   list_of_sender_id

#Compute probability and CDF for hop made by Content
file_prob_hop_data.write("#Hop\tP(hop)\tOccorrenze\n")
file_cdf_data.write("#Hop\tCdf\n")

occorrenze_data = {}
count_data = 0
hop_count_data = 0
prob_hop_data = {}
for i in dic_of_number_of_data_hop.keys():
        elem = dic_of_number_of_data_hop[i]
        for j in elem:
                if not occorrenze_data.has_key(j):
                        occorrenze_data[j] = 1
                else:
```

```
                        count_data = occorrenze_data[j
                           ]
                        count_data = count_data + 1
                        occorrenze_data[j] =
                           count_data
del dic_of_number_of_data_hop

index_data = range(0, 101)
for i in index_data:
        if not occorrenze_data.has_key(i):
                occorrenze_data[i] = 0
somma_data = 0
for i in index_data:
        somma_data = somma_data + occorrenze_data[i]

#compute probability of each number of hops
for i in index_data:
        prob_hop_data[i] = float(occorrenze_data[i])/
           float(somma_data)
        file_prob_hop_data.write(repr(i)+"\t"+"{0:.4f
           }".format(float(prob_hop_data[i]))+"\t"+
           repr(occorrenze_data[i])+"\n")
file_prob_hop_data.close()

#Verify: sum of probability must be 1
tot_data = 0
index_data = range(0, len(prob_hop_data))
#print 'Len prob_hop_data: ', len(prob_hop_data)
for i in index_data :
        tot_data = tot_data + prob_hop_data[i]
print "Verify sum of probability for hop made by
   Content: ", tot_data, "\n"

sum_prob_data = 0
res_data = 0
index2_data = occorrenze_data.keys()
index2_data.sort()
```

```
#Compute CDF for Content hop
for i in index2_data:
        sum_prob_data = sum_prob_data +
            occorrenze_data[i]
        res_data = float(sum_prob_data) / float(
            somma_data)
        file_cdf_data.write(repr(i)+"\t"+"{0:.4f}".
            format(float(res_data))+"\n")
file_cdf_data.close()

'''END NEW COMPUTATION STATISTIC'''

print 'Check_hop= ', check_hop
```

Bibliografia

[1] Van Jacobson and Diana K. Smetters and James D. Thornton and Nicholas H. Briggs and Rebecca L. Braynard Networking Named Content. In Proceeding *CoNEXT '09 Proceedings of the 5th international conference on Emerging networking experiments and technologies*, Rome, Italy, December 2009.

[2] Ryuji Wakikawa and Romain Kuntz and Rama Vuyyuru and Lucas Wang and Lixia Zhang Data Naming in Vehicle-to-Vehicle Communications newblock In *Computer Communications Workshops (INFOCOM WKSHPS), 2012 IEEE Conference* , 328-33, March 2012.

[3] Jérome Harri and Fethi Filali and Christian Bonnet Mobility Model for Vehicular Ad Hoc Networks: A survey and Taxonomy newblock In *Communications Surveys and Tutorials, IEEE*, Volume 11, 19-41, 2009.

[4] Traffic and Network Simulation Environment
`http://wiki.epfl.ch/trans`.

[5] M. Fiore and J. Harri and C. Bonnet Understanding Vehicular Mobility in Network Simulation newblock In Proceeding*1st IEEE Workshop on Mobile Vehicular Networks (MoveNet'07, in conjunction with IEEE MASS' 07)*, October, 2007.

[6] The network simulator NS3
`https://www.nsnam.org/`.

[7] Retirement of the TIGER Map Service
`http://www.census.gov/geo/www/tigerms_redir.html`.

[8] TIGER Products
`http://www.census.gov/geo/maps-data/data/tiger.html`.

[9] TIGER/Line Files Technical Documentation Sito http://www.census.gov/geo/www/tiger/tiger2002/tgr2002.pdf. http://www.census.gov/geo/www/tiger/tiger2002/tgr2002.pdf.

[10] Simulation of Urban MObility
http://sumo.sourceforge.net/.

[11] SUMO Main Page
http://sourceforge.net/apps/mediawiki/sumo/index.php?title=Main_Page.

[12] SUMO Tutorial
http://sumo.sourceforge.net/doc/current/docs/userdoc/Tutorials.html.

[13] Eugenio Giordano and Raphael Frank and Giovanni Pau and Mario Gerla CORNER: A Radio Propagation Model for VANETs in Urban Scenarios newblock In *Vehicular Technology Conference (VTC 2010-Spring), 2010 IEEE 71st*, Pages 1-5, May, 2010.

[14] Eugenio Giordano and Raphael Frank and Giovanni Pau and Mario Gerla VERGILIUS: a scenario Generator for VANET newblock In *Vehicular Technology Conference (VTC 2010-Spring), 2010 IEEE 71st*, Pages 1-5, May, 2010.

[15] VERGILIUS http://vehicular.cs.ucla.edu/index.php/projects/vergilius.

[16] Alexander Afanasyev and Ilya Moiseenko and Lixia Zhang ndnSIM: NDN simulator for NS-3 newblock In *NDN, Technical Report NDN-0005*, October, 2012.

[17] ndnSIM documentazione e codice
http://irl.cs.ucla.edu/ndnSIM/.

[18] netconvert tool
http://sumo.sourceforge.net/doc/current/docs/userdoc/NETCONVERT.html.

[19] jtrrouter tool
http://sumo.sourceforge.net/doc/current/docs/userdoc/JTRROUTER.html.

[20] File TGR
http://www2.census.gov/geo/tiger/tiger2006se/CA/.

[21] SUMO - GUI
http://sumo-sim.org/doc/current/docs/userdoc/SUMO-GUI.html.