Politecnico di Torino

POLITECNICO DI TORINO

Laurea Magistrale in Ingegneria Informatica
A.a 2020/2021
Luglio 2021

Selezione automatica di funzioni di sicurezza in reti virtualizzate

Relatori

Prof. Riccardo SISTO

Prof. Guido MARCHETTO

Prof. Fulvio VALENZA

Dott. Daniele BRINGHENTI

Candidato

Cosimo MANISI

Sommario

L'avvento e la diffusione di Internet, e dei suoi servizi, hanno rappresentato una vera e propria rivoluzione a partire dagli inizi degli anni novanta che ha cambiato il nostro modo di relazionarci ed interfacciarci al mondo.

Oggi usiamo dispositivi elettronici, come computer e telefoni cellulari in qualsiasi momento della giornata, sia nella nostra vita privata che nella nostra vita lavorativa. Li usiamo per lo shopping online ,per svolgere attività di smart-working ecc. Quante volte abbiamo sentito le frasi: "Vai su internet", "Cerca su Google", "collegati col Wi-Fi", "ti invio tramite Whatsapp la posizione!", "Ti ho inviato una mail!".

Sono solo alcune delle frasi che sono ormai entrate nel vocabolario quotidiano del nostro Paese. Tutte queste azioni sono possibili proprio perchè tutti i dispositivi sono collegati nel mondo attraverso una grande rete: Internet .

Internet, agli inizi è stato progettato per interconnettere diversi computer alla stessa rete utilizzando una serie di protocolli, ciascuno protocollo faceva capo ad uno strato nello stack di protocolli ISO / OSI. Questa suddivisione in livelli permetteva di costruire dispositivi relativamente semplici che erano in grado di funzionare solo su un livello specifico, nascondendo le informazioni non necessarie degli altri livelli. Oggi le reti si stanno evolvendo rapidamente e la necessità di una rete più dinamica e automatizzata sta assumendo un ruolo molto importante.

Queste motivazioni hanno portato, negli ultimi anni, allo sviluppo di tre nuovi paradigmi: Network Function Virtualization (NFV) e Software Defined Networking (SDN), Service Fucntion Chain (SFC).

Il paradigma SDN definisce un nuovo tipo di architettura per la realizzazione di una rete direttamente programmabile ottenuta separando il piano di controllo dal piano dati (rompendo quella che è l'integrazione verticale di un dispositivo di rete), passando ad un approccio più centralizzato. In particolare, il piano di controllo è affidato ad un unico controller (SDN Controller) in modo da ottenere una maggiore scalabilità e sicurezza a prescindere dell'apparato utilizzato per il piano dati, favorendo una rete ancora più dinamica.

NFV mira a virtualizzare le funzioni di rete, servizi e/o applicazioni che tendenzialmente sono eseguiti su dispositivi hardware, esse prendono il nome di Network Function Virtualization (VNF).

Le funzioni sono allocate in macchine virtuali quindi sono separati dall'hardware che diventa un semplice host. In questo modo possono essere dinamicamente aggiunte, clonate e rimosse su richiesta delle esigenze dei clienti. Questo è un grande vantaggio perché si riducono i costi operativi, i tempi di risposta e la quantità di lavoro svolto dagli amministratori di rete.

Questo problema è legato soprattutto nelle reti aziendali, dove la configurazione di sistema potrebbe raggiungere agevolmente un livello elevato di complessità e potrebbero esserci alcuni errori imprevisti che causerebbero comportamenti indesiderati o addirittura malfunzionamenti dell'infrastruttura di rete. L'azione combinata tra SDN e NFV permette la realizzazione di una rete programmabile tra funzioni virtualizzate.

SFC permette di costruire servizi più complessi, sfruttando il concetto di "concatenazione di servizi", in cui sono coinvolte più VNF in sequenza in modo che l'interazione complessiva fornisca il servizio desiderato.

Il framework di partenza è Verifuse (VERInet FUNction SElection and placement),in grado di scegliere automaticamente quante e quali funzioni di sicurezza sono necessarie e allocarle tra i server fisici disponibili.

Si vuole proporre un'estensione del framework in grado di soddisfare il lavoro degli amministratori di rete, aiutandoli a definire un requisito di sicurezza e determinare le rispettive configurazioni da assegnare alle VNFs disponibili nel catalogo. Si propone un modo innovativo per affrontare le esigenze del cliente e per configurare automaticamente la Service Chain di VNFs.

L'idea si fonda sul concetto di Functionality: la capacità di una funzione di rete (VNFs) di adempiere alle specifiche di un requisito di rete. Ogni requisito puo essere supportato da una o piu funzionalità o in alcuni casi da nessuna di queste.

La fase di ricerca e di ottimizzazione delle funzionalità in possesso dei requisiti validi spetta al risolutore Gurobi, che con l'aiuto di formule e modelli matematici cerca tutte le soluzioni idonee.

Le soluzioni individuate non sono nient'altro che sequenze di functionality (Service Chain). La scelta di quale tra queste soluzione individuate è la migliore per la rete in esame spetta al un modulo interno del framework VERIFUSE chiamato *Allocation and configuration generator*.

Ringraziamenti

Una dedica speciale alla mia famiglia e ai miei amici, che ogni giorno hanno condiviso con me gioie, sacrifici e successi. L'affetto e il sostegno che mi hanno dimostrato rendono questo traguardo ancora più prezioso.

Cosimo Manisi

"If you can dream it, you can do it"
Walt Disney

Indice

Elenco delle figure

Abbreviazioni

NFV Network Function Virtualization

VNF Virtual Network Function

SDN Software Define Networking

SFC Service Function Chain

ISP Internet Service Provider

VPN Virtual Private Network

COTS Commercial Off-the-Shelf Component

API Application Programming Interface

ETSI European Telecommunications Standards Institute

MANO Management and Orchestration

VEREFUSE Verinet Function Selection and Placement

NSF Network Security Function

MaxSMT Maximum Satisfiability Problem

NSR Network Security Requirement

RDF Requirement Driven Functionality

LER Logic Enforce Requirement

COF Chain Of Functionality

Capitolo 1

Introduzione

L'avvento e la diffusione di Internet, e dei suoi servizi, hanno rappresentato una vera e propria rivoluzione, tecnologica, che ha cambiato il nostro modo di relazionarci ed interfacciarci al mondo.

Oggi usiamo dispositivi elettronici, come computer e telefoni cellulari in qualsiasi momento della giornata, sia nella nostra vita privata che nella nostra vita lavorativa. Li usiamo per lo shopping online, per svolgere attività di smart-working ecc. Quante volte abbiamo sentito le frasi: "Vai su internet", "Cerca su Google", "collegati col Wi-Fi", "ti invio tramite Whatsapp la posizione!", "Ti ho inviato una mail!".

Sono solo alcune delle frasi che sono ormai entrate nel vocabolario quotidiano del nostro Paese. Tutte queste azioni sono possibili proprio perchè tutti i dispositivi sono collegati nel mondo attraverso una grande rete: Internet (detta anche Big Internet).

Internet, agli inizi è stato progettato per interconnettere diversi computer alla stessa rete utilizzando una serie di protocolli, ciascuno protocollo faceva capo ad uno stack di protocollo ISO / OSI. Questa suddivisione in livelli permetteva di costruire dispositivi relativamente semplici che erano in grado di funzionare solo su un livello specifico, nascondendo le informazioni non necessarie degli altri livelli. Oggi le reti si stanno evolvendo rapidamente e la necessità di una rete più dinamica e automatizzata sta assumendo un ruolo molto importante.

Queste motivazioni hanno portato, negli ultimi anni, allo sviluppo di tre nuovi paradigmi: Network Function Virtualization (NFV) e Software-Defined Networking (SDN), Service Fucntion Chain (SFC).

Il paradigma SDN [1] definisce un nuovo tipo di architettura per la realizzazione di una rete direttamente programmabile ottenuta separando il piano di controllo dal piano dati (rompendo quella che è l'integrazione verticale di un dispositivo

di rete), passando ad un approccio più centralizzato. In particolare, il piano di controllo è affidato ad un unico controller (SDN Controller) in modo da ottenere una maggiore scalabilità e sicurezza a prescindere dell'apparato utilizzato per il piano dati, a favore di una rete ancora più dinamica.

NFV mira al virtualizzare le funzioni di rete, servizi e/o applicazioni che tendenzialmente sono eseguiti su dispositivi hardware, esse prendono il nome di Network Function Virtualization (VNF). Le funzioni sono allocate in macchine virtuali quindi sono separati dall'hardware che diventa un semplice host. In questo modo possono essere dinamicamente aggiunte, clonate e rimosse su richiesta dei clienti. Questo è un grande vantaggio perché si riducono i costi operativi, i tempi di risposta e la quantità di lavoro svolto dagli amministratori di rete.

Questo problema è legato soprattutto alle reti aziendali, dove la configurazione di sistema potrebbe raggiungere un livello elevato di complessità e potrebbero esserci alcuni errori imprevisti che causerebbero comportamenti indesiderati o addirittura malfunzionamenti dell'infrastruttura di rete. L'azione combinata tra SDN e NFV permette la realizzazione di una rete programmabile tra funzioni virtualizzate.

SFC [2] permette di costruire servizi più complessi, sfruttando il concetto di ("concatenazione di servizi") in cui sono coinvolte più VNF in sequenza in modo che l'interazione complessiva fornisca il servizio desiderato.

Lo scopo della tesi è estendere ulteriolmente il framework Verifuse (VERInet FUnction SElection and placement) [3]. Esso nasce per agevolare gli amministratori di rete nel loro compito, proponendo un modo innovativo di rispondere alle esigenze del cliente. L'estensione consiste nel determinare, partendo da un requisito di sicurezza, quanti sono, quali sono e come devono essere configurate le catene di servizi (VNF) per soddfisfare un requsito.
La tesi è suddivisa nei seguenti capitoli:

- Capitolo 1: Introduzione alle nuove tecnologie, andando ad analizzare come questi nuovi paradigmi stanno continuando a modificare la percezione di una rete.
- Capitolo 2: Analisi in dettaglio di come questi si integrano nell'estenzione del tool, e come questi nuovi paradigmi stanno continuando a modificare la percezione di una rete.
- Capitolo 3: Motivazioni che hanno spinto allo sviluppo del framework.
- Capitolo 4: Si analizzano i concetti chiave che hanno permesso di estendere ulteriolmente il tool, seguite dalle ragioni e motivazioni che hanno spinto

l'implementazione con annessi vantaggi e svantaggi. Si procede con una panoramica su quali sono i blocchi principali e quale è il loro ruolo nel workflow.

- Capitolo 5: Verranno analizzati i modelli esppressi con il linguaggio XML, per un requisito , per un manifest di una VNF e una functionality.
- Capitolo 6: verranno descritte le formulazioni matematiche, vincoli utili per applicare il processo di selezione ottimizzata delle funzionalità.
- Capitolo 7: Validazione e testing su alcuni casi d'uso. Risultati ottenuti con i test di scalabilità
- Capitolo 8: Conclusione e considerazioni sul lavoro svolto nel percorso di sviluppo.
- Appendice A: Intregrazione di moduli aggiuntivi per sviluppi futuri. Si descrive in sintesi un possibile modulo aggiuntivo (Action Order Logic).
- Appendice B: Guida su come configurare e installare i vari componenti per poter eseguire il tool.
- Appendice C: Elenco completo di tutti gli elementi modellati con il linguaggio XML per i requisiti e le VNFs

Capitolo 2

Background

Questo capitolo introduce quali sono i concetti che verrano utilizzati per lo sviluppo della tesi. Si fa riferimento a (NFV) Network Function Virtualization e (SDN) Software defined network, nuovi paradigmi di rete nati dalla crescente richiesta di avere una rete agile e capace di rispondere automaticamente alle esigenze del traffico e dei servizi che le attraversano. Queste due tecnologie nascono quasi in contemporanea, partendo da due visioni diverse di trattare i dispositive di rete. Negli ultimi anni questi vengono usati spesso insieme per via della forte relazione tra di loro, in quanto complementari ma sempre piu co-dipendenti. Il successo di queste due tecnologie ha introdotto di conseguenza un nuovo paradigma Service Function Chain. Spesso queste tecnologie introducono quella che è l'attuale direzione della nuova visione della rete, **"rete softwarizzata"**.

2.1 Software Networking

2.1.1 Network Function Virtualization

Al giorno d'oggi i servizi informatici vengono distribuiti su infrastrutture di rete virtualizzate, ma le funzione di rete (e.g. firewall, NAT, proxy, VPN, etc.) continuano ad essere dei dispositivi hardware dedicati. Una rete cablata con singole funzioni è noiosa da mantenere, lenta nell'evoluzione e impedisce ai fornitori di servizi di offrire servizi dinamici. Negli ultimi anni in cui il mondo digitale prende sempre più piede, i fornitori di servizi Internet service provider (ISP) sentono l'esigenza di adattare la loro rete alle richieste da parte dei customer. Spesso queste richieste comportano installazioni di nuove apparecchiature, cambiamenti topologici, spazi aggiuntivi costi aggiuntivi e personale qualificato per la manutenzione di un'infrastruttura che con il tempo diventa sempre piu complessa da gestire e garantire

un corretto funzionamento. Per cercare di risolvere in parte queste problematiche nasce Network Function Virtualization conosciute sotto il nome di: **NFV**.

La virtualizzazione delle funzioni di rete è la capacità di eseguire qualsiasi funzione di rete su un hardware standard, possibilmente con l'aiuto della virtualizzazione informatica per ottenere un uso efficiente delle risorse.

Tendenzialmente le VNF sono implementate come macchine virtuali, pertanto possono essere aggiunte e rimosse dinamicamente su richiesta riducendo le attività amministrative, i tempi di risposta e i costi.

Recentemente le VNF permettono l'inserimento di funzioni di rete personalizzate all'interno dell'infrastruttura di rete .Questa possibilità consente anche l'implementazione di funzioni di sicurezza (come firewall, proxy, concentratori VPN).

Quando il cliente richiede un servizio, chiede di creare o clonare una istanza di VNF e di allocarla su un server general purpose senza preoccuparsi di andare a scegliere qual è il miglior hardware per effettuare il dispiegamento.[4]

I 4 punti chiave su cui si basa NFV:

1. Fast standard hardware (e.g. Intel servers): si riferisce a componenti hardware e software disponibili sul mercato utilizzabili nei loro progetti. Spesso si riassume con l'espressione COTS, in inglese (Commercial) Off-the-Shelf component.

2. Funzioni di rete basate su software: precedentemente eseguite su un'appliance dedicata, ora viene considerata come un'immagine software, in esecuzione su un server standard, invece che su specifico hardware.

3. Virtualizzazione : Tutti i vantaggi della virtualizzazione (provisioning rapido, scalabilità, mobilità, multi-tenancy)

4. Standard API: ETSI framework

I principali vantaggi ottenibili con NFV:

- Scalabilità: ampliare o ridurre in modo dinamico la capacità allocata in base al carico effettivo, permette di ottimizzare l'uso delle risorse, quali CPU, memoria, storage e network, attivando sullo stesso server fisico più VNF che implementano diverse tipologie di servizio, in modo da sfruttare appieno la capacità disponibile e ridurre il consumo energetico

- Affidabilità: a fronte di un malfunzionamento hardware o semplicemente un cambio topologico della rete le VM possono essere facilmente migrate da un server all'altro;

- Flessibilità: riconfigurare la topologia della rete quasi in tempo reale in seguito a nuove installazioni di nuovi servizi nuovi dispositivi di rete installati, permette di ottimizzare le prestazioni.

- Costi: i costi di manutenzione di una rete spesso non sono bassi, uno degli obiettivi è ridurre il TCO (Total Cost of Ownership), ridurre COTS e migliorare il Time-to-Market, sfruttando la maggiore agilità e flessibilità offerta da NFV nel dispiegamento dei servizi.

NFV Framework

ETSI NFV Industry Specification Group (ISG) nasce per definire i requisiti e l'architettura del framework per garantire l'interfacciamento con la tecnologia NFV. ETSI, l'Istituto Europeo per le norme di Telecomunicazioni, in inglese European Telecommunications Standards Institute, è un organismo internazionale, responsabile della definizione e dell'emissione di standard nel campo delle telecomunicazioni.

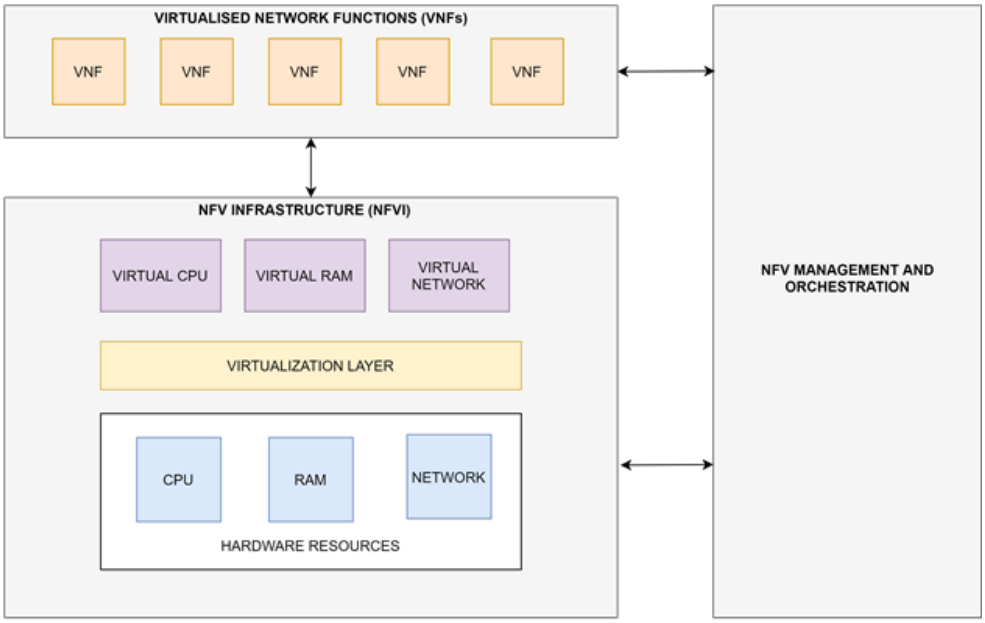

Figura 2.1: Network Function Virtualization (NFV) - Architettura del framework

I tre macro-blocchi funzionali presenti nella figura cooperano insieme e sono tra loro coordinati. Questo framework[5] descrive, ad alto livello, l'implementazione delle funzioni di rete su infrastrutture fisiche e virtuali. Questo si basa su tre blocchi di lavoro NFV, coordinati da un entità chiamata orchestratore:

- VNF (Virtual Network Function): un'implementazione software di una funzione di rete in grado di funzionare su NFVI. Essa è composta da un insieme di VNF.

- NFVI (NFV Infrastructure): esso rappresenta parte core del framework in cui vengono definite insieme di risorse (fisiche o virtuali) utilizzate per ospitare e connettere VNF, come elaborazione, archiviazione o rete. Particolare importanza ha il visualization layer.
- NFV Management and Orchestration (MANO) : autorizza e convalida richieste di risorse NFVI. Copre la gestione e l'orchestrazione necessarie per il provisioning di VNF, insieme alla loro gestione del ciclo di vita.
  - NFV Orchestrator: responsabile per l'onboarding di nuovi servizi di rete (NS) gestione delle risorse globali (VNF);
  - VNF Manager: sovrintende alla gestione del ciclo di vita delle istanze VNF, come la configurazione e la segnalazione degli eventi;
  - Virtualized Infrastructure Manager (VIM): alloca le risorse per una VNF, e gestisce l'interazione di una VNF con risorse di elaborazione, archiviazione e di rete, fornendo informazioni sui eventuali guasti, seguite da informazioni sul monitoraggio .

2.1.2 Software Defined Network

La rete tradizionale non è del tutto sufficiente a causa della mutazione della società sempre più improntata nel digitale dove (quasi) tutto è connesso in rete ed è accessibile da qualsiasi luogo o da qualsiasi dispositvo. Tuttavia nonostante la loro ampia diffusione, le reti IP tradizionali sono complesse e molto difficili da gestire. Oltre alla complessità di configurazione, gli ambienti di rete devono resistere alla dinamica dei guasti e adattarsi ai cambiamenti dei carichi di lavoro. Rendere tutti questi meccanismi automatici sfruttando le attuali rete ip è quindi altamente impegnativo per gli amministratori di rete, e richiede un enorme dispendio di energie e costi.

Software defined network (SDN) è un nuovo paradigma di rete emergente che dà la possibilità di superare i limiti che le attuali infrastrutture di rete impongono, introducendo nuove astrazioni in networking, semplificando la gestione e l'evoluzione della rete. [6] SDN si bassa principalmente su tre pilastri:

- Separazione verticale di un dispositivo di rete: un apparato di rete non viene più visto come un middle box con una specifica funzione ma viene suddiviso in più componenti ottenendo cosi una separazione tra la logica di controllo e logica di inoltro dei dati.
- Semplicità dispositivo: con la separazione verticale il dispositivo diventa più semplice e si trasforma in un dispositivo di inoltro. La parte di logica del dispositivo viene trasferita su un controller centralizzato.

- Logica centralizzata: la presenza di un controller centralizzato permette una visione astratta della rete a **"Big switch"**. Questo semplifica i processi di applicazione di nuovi criteri di inoltro, (ri)configurazione ed evoluzione delle rete. E' importante sottolineare come una logica centralizzata non significhi sistema centralizzato. Infatti per garantire adeguati livelli di performance , scalabilità e affidabilità, specialmente in ambiente di produzione, si utilizza un approccio "logicamente centralizzato ma fisicamente distribuito".

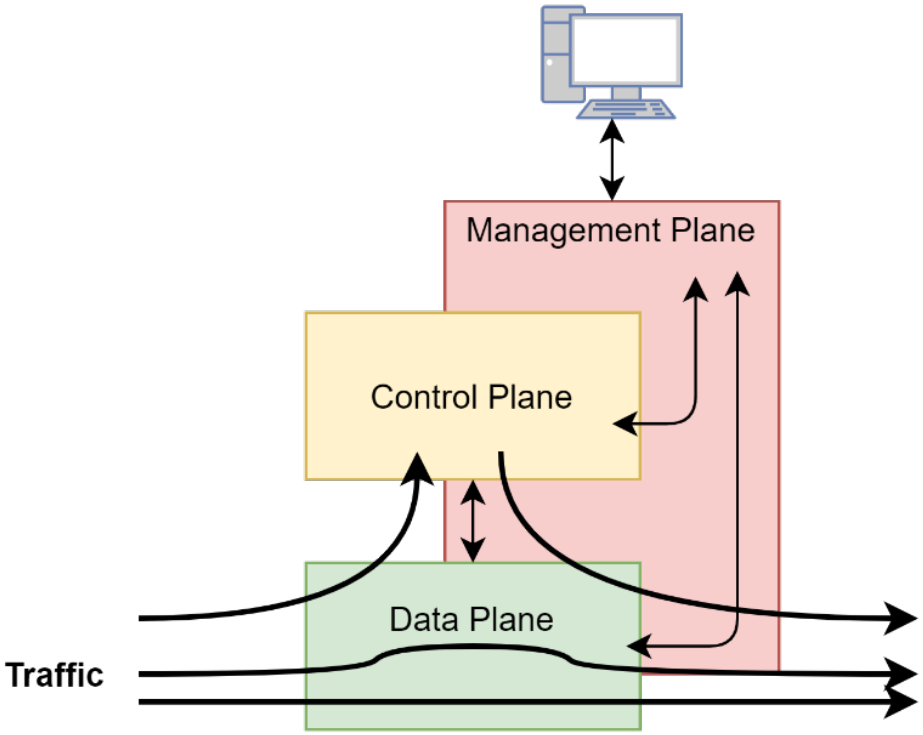

Figura 2.2: Struttura di un dispositivo di rete

Un dispositivo di rete viene logicamente partizionato in :

- Data plane: è la parte hardware che si occupa dell 'inoltro dei pacchetti svolgendo la funzione di forwarding del traffico in base alle decisioni imposte dal control plane

- Control plane: rappresenta l'insieme dei protocolli usati per riempire le tabelle di inoltro del data plane

- Management plane: l'insieme dei servizi software o tool di base utilizzati per monitorare e configurare il funzionamento della rete logica. Include applicazione come algoritmi di routing, firewall, load balancer ecc.

Questo tipo di separazione di un apparato di rete tradizionale non esiste, tutto è inglobato in un unico dispositivo, spesso chiamato integrazione verticale.

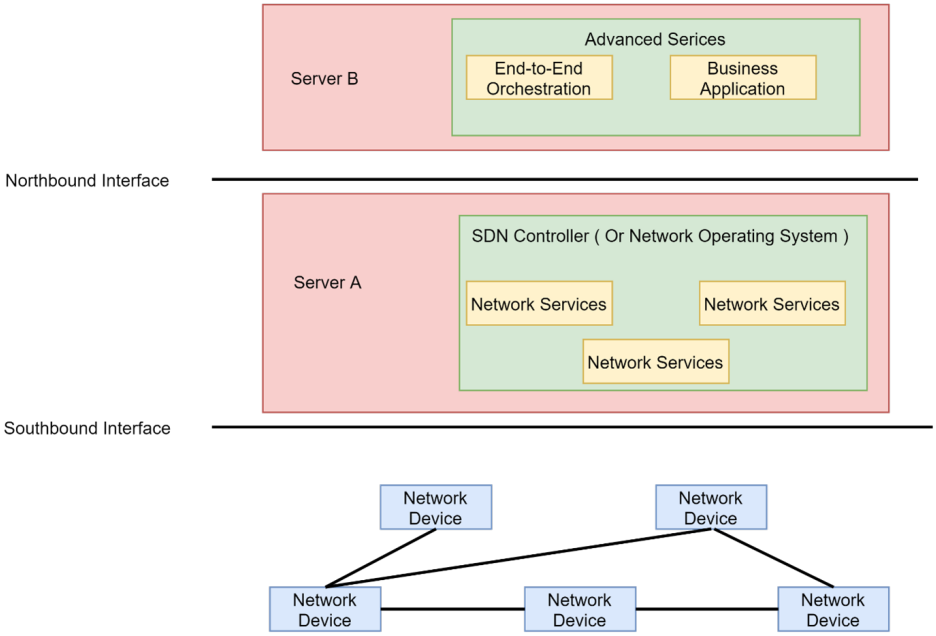

Figura 2.3: Architettura (SDN) Software Defined Network

L'architettura della rete è formata principalmente da tre strati, interconnesse attraverso interfacce API chiamate northbound e southbound.

- Application Layer: include una serie di applicazioni e servizi di rete avanzate che permettono di migliorare le prestazioni delle applicazioni. Queste possono essere eseguite localmente sull SDN controller o su un'altra macchina. Le reti tradizionali utilizzano un'appliance hardware specializzata per queste funzioni, mentre una rete definita dal software utilizza il controller per gestire il comportamento. Application layer comunica istruzioni di rete specifiche al controller SDN, facendo uso dell'interfaccia southbound.

- The control layer: formato essenzialmente dal SDN controller. Puo essere visto semplicemente come un Network Operator System. Questo livello elabora i requisiti inviati dal livello dell'applicazione tramite l'API e li trasmette all'effettiva infrastruttura di rete tramite API. Inoltre, estrae le informazioni

dal livello dell'infrastruttura e le comunica al livello dell'applicazione per ottimizzare la funzionalità.

- Infrastucture layer: contiene dispositivi fisici della rete nel data center. Questi apparati di rete controllano importanti funzioni di inoltro e capacità di elaborazione dei dati e sono responsabili della raccolta di informazioni critiche. Es: variazione della topologia della rete inseguito a dei guasti.

2.1.3 Service Function Chain

Le due innovative strategie emergenti SDN e NFV hanno introdotto una terza architettura, SFC.

SFC[7] fornisce un'alternativa flessibile ed economica all'ambiente per i provider di servizi cloud (CSP), provider di servizi applicativi (ASP) e provider di servizi Internet (ISP). Essa nasce dalla richiesta dinamica degli utenti che richiedono la fornitura dei servizi end-to-end di funzione di rete tradizionali quali NAT FireWall VPn ecc.. La definizione di SFC è l'istanziazione di un insieme ordinato di applicazioni o Service Function come NAT, firewall, QoS e così via, in modo che i pacchetti debbano attraversare in sequenza servizi in cascata nel loro percorso dalla sorgente a destinazione.

Implicitamente fornisce un soluzione significativa in termini di flessibilità, ma alla stesso tempo introduce una complessità nella ricerca di un design corretto , per prevenire inconsistenze o conflitti quando entrano in gioco più sotto reti.[2][8][9]

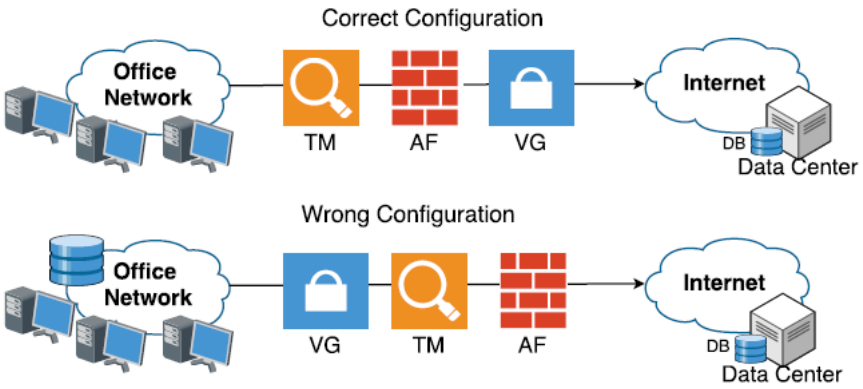

Figura 2.4: Esempio semplice SFC.[10]

Presentiamo uno scenario[10] molto semplice,come mostrato in Figura 2.4, dove gli utenti nella rete dell'ufficio accedono a Internet tramite un breve SFC costituito da un monitor del traffico (TM), un firewall applicativo (AF) e un gateway VPN

(VG). Un database aziendale (DB) è ospitato in un Data Center remoto (DC) e devono essere soddisfatti tre requisiti principali:

1. il traffico crittografato dagli utenti dell'ufficio deve essere scartato
2. il traffico verso DC deve essere crittografato;
3. le connessioni al DB devono essere monitorate.

Gli amministratori di rete possono configurare gli SF in modo che:

- TM implementa la policy "conta le connessione al DB";
- AF implementa il "rimuovi tutto il traffico in uscita crittografato";
- VG implementa il criterio "cripta tutto il traffico verso il controller di dominio".

Questa semplice configurazione è corretta se e solo se l'ordine di funzioni nell'SFC è TM-AF-VG ("Configurazione corretta" nella figura 2.4). Diversamente ordinando la sequenza SFC come VG - TM - AF porterebbe a una situazione di stallo in cui VG crittografa tutto il traffico a DC, ma TM non può contare il numero di connessioni e AF elimina tutto il traffico inviato a DC[11].

Come interagiscono SDN e VNF

Come abbiamo già osservato NVF mette a disposizione funzioni di rete o di sicurezza implementate attraverso applicazioni software note come VNF. Le VNF possono essere codici eseguibili (es. C, C++ python) , virtual machine , docker ospitate su dei server general purpose , dove esse possono essere aggiunte o rimosse on-demand. SDN invece propone un'architettura di rete che permette di fornire mezzi per controllare dinamicamente la rete, grazie alla programmabilità del controllo del traffico, ed effettuarne il provisioning delle reti come servizio .

In questo contesto è possibile vedere come co-operano Software Defined Networking e il Network Function Virtualization. L'amministratore di rete per ogni richiesta da parte dell'utente, in base alle esigenze, provvede a determinare quali sono le catene di funzioni di servizio (linee di blu e linee rosse) adatte per soddisfare tale richiesta. La catena può essere composta da una serie di servizi NFV. Questa richiesta verrà presa in carico dal controller SDN, il quale inoltrerà all 'infrastruttura di rete i comandi per soddisfare la richiesta del cliente. L'infrastruttura si incaricherà di andare ad istanziare funzioni di rete da abilitare e indicarne il path che deve attraversare.[12] Alcune problematiche legate a questi scenari verranno affrontate nei prossimi capitoli.

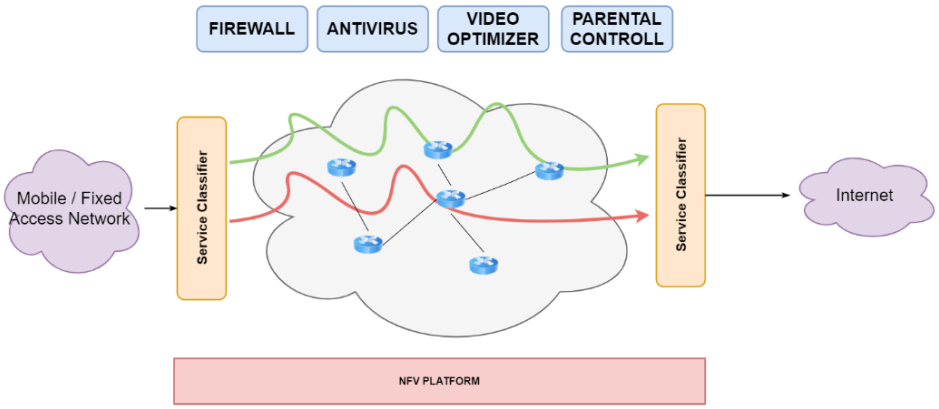

Figura 2.5: Panoramica SFC (Service Function Chain)

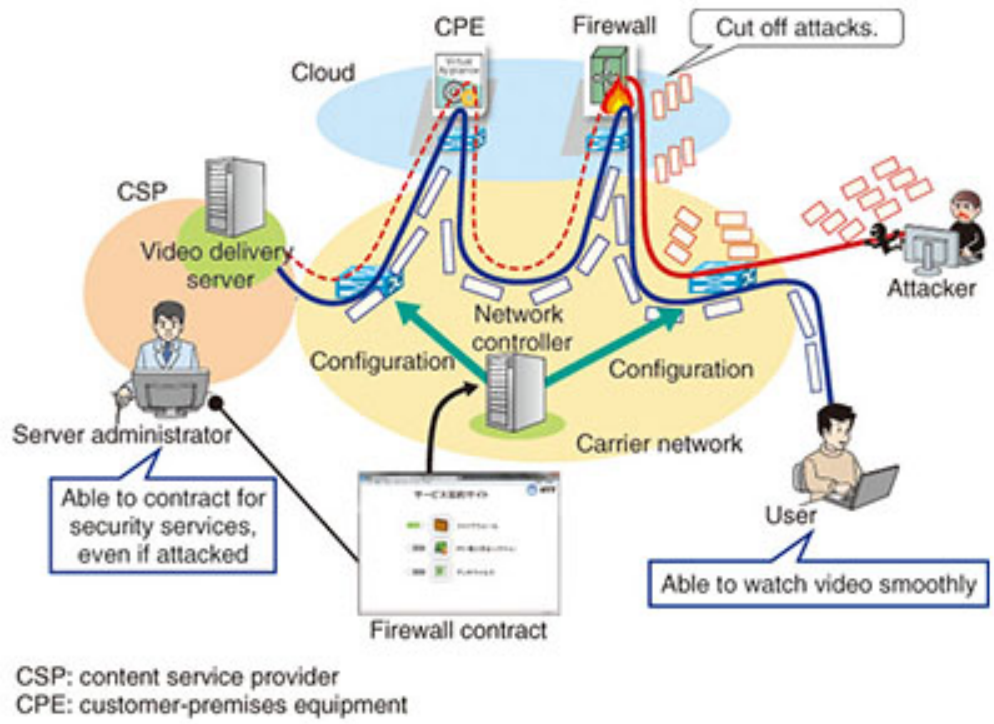

Figura 2.6: Esempio di uno scenario reale - Service Function Chain. [7]

2.2 Automatizzazione delle funzioni di rete

Le tecnologie emergenti come Software-Defined-Networking e la virtualizzazione delle funzioni di rete hanno permesso la configurazione di questi servizi in maniera dinamica, rendendoli così approcci automatici che possono sostituire attività manuali e soggette a errori [13]. Oggi un approccio di tipo automatico è fattibile e porterebbe un grosso vantaggio nell'affrontare attacchi di sicurezza informatica [14],essere robusti a mal configurazioni e/o a mal funzionamenti di rete ecc. Nonostante questi vantaggi siano senza dubbio utili, il problema è la quantità limitata di strumenti che permettano la gestione automatica dei servizi, che troverebbe applicazione in grande scala in ambito cloud.[15]

La maggior parte degli attacchi su una rete hanno successo per l'errata configurazione dei dispositivi di sicurezza e apparati di rete presenti.[10] Questo problema è dovuto al fatto che gli amministratori di rete spesso lavorano con un numero elevato di dispositivi, di diversi marchi, con diversi modi di configurare l'apparato, utilizzando diversi linguaggi, creando conflitti con le politiche di sicurezza ecc. L'automazione sicuramente porterebbe ad una riduzione degli errori umani, riducendo anche i rischi di sicurezza.[16][17][18]

Questo motiva l'introduzione di servizi che permettono la configurazione automatica delle VNF. Da questa necessità nasce l'idea di proporre un framework VEREFUSE.(Schema a blocchi in Fig. 2.7)

Il tool permette la definizione di un requisito di sicurezza espresso con un linguaggio a misura d'uomo che permette la scelta ottimale di schemi di assegnazione e configurazione dei NSF(servizi di rete) I risultati calcolati sono ottimali rispetto a un insieme di costi di funzioni, come la riduzione al minimo del numero funzioni installati o la riduzione del numero di regole.

Il framework è in grado di distribuire le funzioni virtuali e configurarle attraverso un'interazione diretta con alcuni orchestratori, senza alcun intervento manuale.

2.2.1 Quadro sintetico di Verefuse

Per capire meglio il contesto è opportuno spendere due parole presentando una visione superficiale di quello che Verefuse è in grado di fare. VEREFUSE gestisce la creazione, la configurazione e l'orchestrazione di un servizio di sicurezza completo della rete end-to-end seguendo un approccio modulare, pensato sin dall'inizio nella fase di design.[3]

VEREFUSE è in grado automaticamente, fornendo in input un grafo di rete chiamato (Service Graph), di trovare un'allocazione delle funzioni di rete VNFs per soddisfare una serie di input di requisiti di sicurezza di rete (NSR), espressi dal progettista del servizio. Per stabilire l'allocazione ottimale degli VNFs si fa appello al problema di Maximum Satisfiability Modulo Theories (MaxSMT) che si occupa di scegliere automaticamente quali funzioni di rete sono necessarie e dove assegnarle.[17]

Infine, il servizio si interfaccia con agenti di orchestrazione cloud per fornire proprietà di sicurezza per la comunicazione tra punti terminali o reti. Le principali piattaforme di orchestrazione open source supportate sono :

- Open Source MANO;[1]
- Open Baton;[2]
- OpenStack Tacker;[3]
- Kubernetes.[4]

[1]Open Source MANO. [Online]. Available: https://osm.etsi.org

[2]Open Baton. [Online]. Available: https://openbaton.github.io

[3]Tacker. [Online]. Available: https://docs.openstack.org/tacker/latest/

[4]Kubernetes. [Online]. Available: https://kubernetes.io/

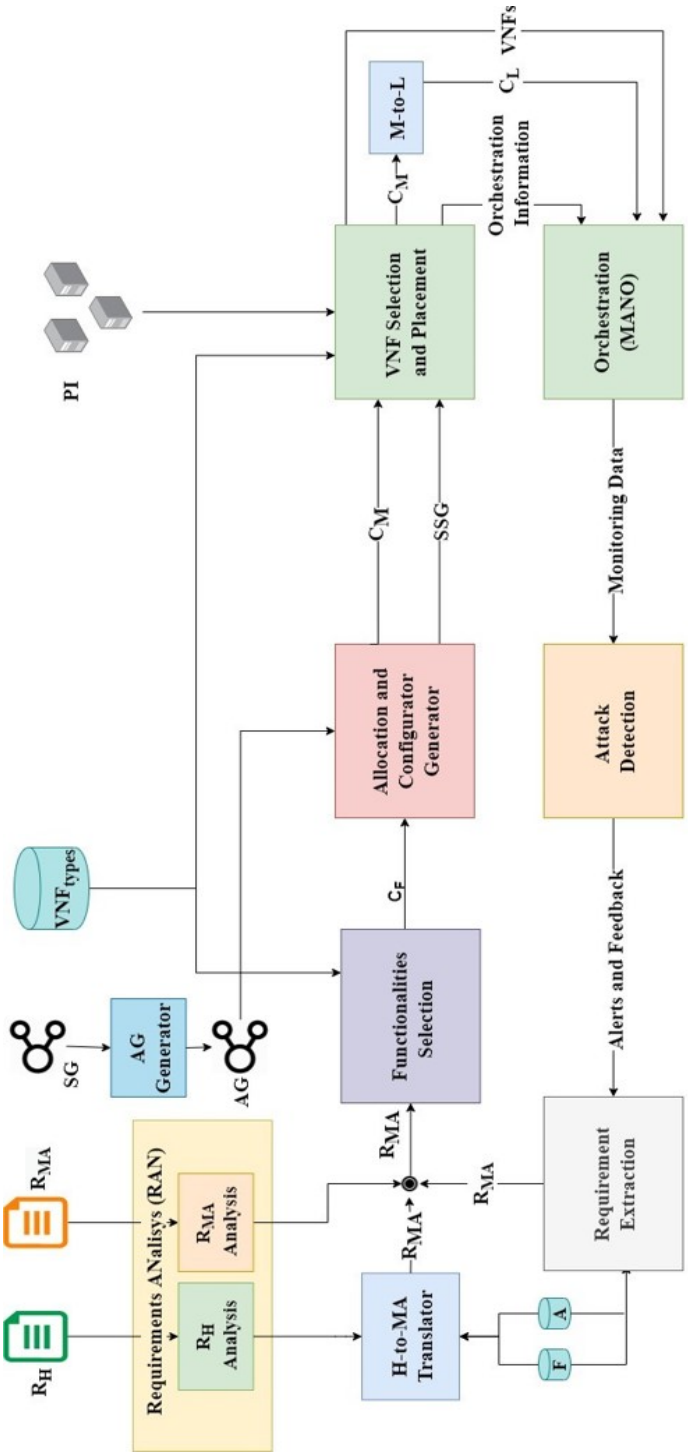

Figura 2.7: Schema a blocchi di Verefuse

Capitolo 3

Obiettivo della tesi

I Due paradigmi architetturali emergenti discussi nel capitolo precedente (capitolo 2) offrono ampie idee di miglioramento delle nuove reti e allo stesso tempo soluzioni accattivanti con notevoli vantaggi in termini di flessibilità, introducendo nuove sfide come la corretta configurazione e il corretto ordinamento di catene di servizi per soddisfare i requisiti di sicurezza piu o meno complessi.

Quello che ancora non è stato sviluppato e che ancora oggi risulta poco presente è l'implementazione di un meccanismo automatico che permetta di evitare l'interazione umana garantendo robustezza ad eventuali mal configurazioni di apparati di rete o utilizzo errato di servizi di rete.

Attualmente, la configurazione della maggior parte delle funzioni di servizio si basa su parametri di basso livello, in cui i valori devono essere selezionati e impostati manualmente. Questo implica un tipico approccio di configurazione per tentativi. Spesso, quando viene rilevata una mal configurazione, gli amministratori di rete cercano di correggere gli errori introducendo regole ad-hoc, ripetendo questo processo fino a quando non si osservano più anomalie.[10]

Questa tecnica, oltre ad essere noiosa e dispendiosa in termini di tempo, manca di una visione completa del comportamento della rete e come tale, è soggetta a errori e può rendere la rete notevolmente difficile da gestire. Ad aggravare lo sfondo è la complessità delle reti moderne, ed è molto difficile come compito, soprattutto senza l'ausilio di idonei software automatizzati.

Questa tesi presenta in primis un modello formale che verrà presentato nel capitolo 4, per poi passare alla logica implementativa della nuova funzionalità che andrà ad estendere il funzionamento di un framework.

L'idea di fondo è consentire l'individuazione delle catene di servizi di rete (che siano virtual machine o container) necessari ad aderire alle esigenze di un requisito di sicurezza. L'attenzione è rivolta principalmente alla scelta corretta delle

applicazioni da coinvolgere e alla configurazione delle funzioni, prestando attenzione all'eventuale ordinamento dei servizi selezionati affinché sia una scelta sensata.

Verranno forniti modelli per rappresentare l'espressione di un requisito di rete, e modelli per una funzione di rete (VNF).

Nel capitolo successivo verranno spiegati i modelli per ciascuno elemento che prende parte all'estensione. Il framework è diviso principalmente in due moduli :

- **RDF** - Requirement Driven Functionality.
- **LER** - Logic Enforce Requirement.

I due moduli sono dipendenti tra di loro. La scelta è voluta in quanto questi dati sono molto correlati tra di loro.
Questi due moduli nonostante la loro dipendenza hanno portato degli ottimi risultati.(Capitolo 7)

Capitolo 4

Approccio

All'attuale workflow di *Verefuse* presente in Figura:2.7, viene integrata una nuova funzionalità automatica per andare a scegliere quali funzioni di rete devono essere usate, come queste funzioni doveno essere configuarate e come devono essere eventualmente concatenate tra di loro al fine di andare a soddisfare le politiche di sicurezza richieste dagli utenti o amministratori di reti.

L'estensione aggiunta è rappresenta nella Figura 2.7 dal blocco viola, chiamato **Functionalities Selection** (Selezioni di Funzionalità).

All' amministraore di rete è concesso definire quali sono le funzioni di rete disponibili nel sistema e un insieme di requisiti di sicurezza che l'amminstratore intende applicare sulla infrastuttura di rete.

4.1 Concetti

Il concetto chiave che ha guidato principalmente lo sviluppo di questo modulo aggiuntivo è ***Functionality***. Possiamo sintetizzare in breve le capacità di una singola funzione di rete di osservare le specifiche di un determinato requisito di sicurezza. La funzionalità possiamo denominarla con un termine generico ' configurazione ' anche se non è un termine del tutto corretto e appropriato.

Il secondo punto chiave è guardare da vicino un requisito di sicurezza.[19] Le domande che sorgono spontanea sono : *"Come immagina un amministratore di rete un requisito ? Che cosa si aspetta di definire in un requisito ? Quali parametri puo andare a definire? Posso andare ad inserire eventuali richieste specifiche per determinate situazioni ?"* Questa serie di domande hanno aiutato a definire un possibile schema di un requisito di siurezza che verrà analizzato e spiegato nel dettaglio nella prossima sezione del capitolo.

L'altro punto chiave molto importante è il concetto di Virtual Function introdotto con l'affermazione del paradigma NFV. Virtual Function è il nuovo modo di andare

ad osservare una funzionalità di rete , vista come entità virtualizzata che puo essere facilmente instaziata sostanzialmente ovunque. La definizione del modello Manifest di una VNF nasce da un serie di domande poste durante la fase di design di questo modulo: *"Come posso configurare queste VNF? Quanti parametri possono essere configurati e quali no? Posso usarle insieme ? Posso metterle in cascata? Quante ne riesco a usare ? Hanno aspetti in comuni tra di loro?"* Il concetto di manifest di una VNF verrà spiegata nella sezione successiva.
Ma il vero cuore di tutto il framework è la capacità di andare ad indivduare,come queste funzionalità di rete disponibili possano essere combinate al fine di garantire l'enforce del requisito espresso dall'amministratore di rete.

Un esempio che chiarisce sin da subito l'idea è la seguente: avendo a disposizione VNF1, VNF2, VNF3 e un requisito R è possibile trovare un modo affinche il requisito R sia soddifsatto? E' sufficiente VNF1? E' sufficiente VNF2 e VNF1? Se considerassi VNF2 e VNF1 otterrei lo stesso risultato ?
La risposta non è sempre scontata, ci si potrebbe trovare nella situazione in cui non si riesce a trovare nessuna soluzione o addirittura trovarci difronte un numero spropositato di soluzioni equivalenti.

4.2 Workflow

Tutti gli elementi citati funzionalità, manifest di una funzione di rete e requisito sono espressi e modellati tramite il linguaggio XML[1] (eXtensible Markup Language). Questo è un linguaggio di markup che permette di modellare la struttura in modo molto preciso e organizzato, grazie anche all'inserimenti di vincoli.
Il framework è stato sviluppato interamente in JAVA insieme ad altri tool di contorno quali , Neo4j, Gurobi e Ant. Tutti gli schemi XML di questi modelli saranno presentati in dettaglio nel Capitolo 5 [1]

Il framework è diviso principalmente in due moduli :

- RDF (Requirement Drifven Functionality) che ha lo scopo di analizzare il requisito e le VNF disponibili nel sistema construendo se possibile un'entità *'Funzionalità'* che permette di identificare quali sono le capacità che hanno in comune con il requisito;

[1]XML è un metalinguaggio per la definizione di linguaggi di markup, ovvero un linguaggio basato su un meccanismo sintattico che consente di definire e controllare il significato degli elementi contenuti in un documento o in un testo.

- LER (Logic Enforce Requirement) che ha lo scopo di determinare e selezionare quanti e quali funzionalità sono necessarie per soddisfare tutte le capacità richieste dal requisito.

I due moduli sono dipendenti tra di loro, ovvero l'input di un modulo è l'output dell'altro, precisamente, l'output del modulo RDF è l'input per il modulo LER. La scelta è voluta in quanto questi dati sono molto correlati tra di loro.

In Figura 4.1 è possible osservare i vari macroblocchi e come questi prendono parte alla esecuzione del framework. Possiamo riassumere gli step principali in:

1. RDF riceve in input un insieme di catalogo di requisiti, contententi una lista di requisiti e il catalogo dell VNF disponibili nel sistema;
2. RDF analizza un singolo requisito alla volta e ne effettua il mapping uno ad uno tra la singola VNF e il requisito;
3. RDF produce un insieme di funzionalità;
4. LER riceve le funzionalità appena ricavate dal modulo precedente, ne verifica se è in grado di trovare almeno una soluzione;
5. LER produce in output catene di funzionalità.

Prima di addentrarci su un esempio pratico, va tenuto in considerazione che questa panoramica fa parte di un progetto più ampio e che gli schemi Requirement e ManifestVNFs possono essere facilmente migliorati in futuro.

I concetti che affronteremo in seguito sono i seguenti:

- Requisito, permette agli amministratori di rete di definire, in maniera semplice ma completa , le informazioni necessarie per soddisfare le esigenze degli utenti;
- ManifestVNF, rappresenta il catalogo delle funzioni di rete disponibili nel sistema;
- Funzionalità, rappresenta il modo in cui una VNF supporta il requisito

La prima operazione da svolgere per un client o amministratore di rete è fornire un requisito al framework. Per esempio possiamo prendere in considerazione per semplicità un requisito in cui un client vuole evitare che durante i giorni lavorativi la rete raggiunga servizi esterni alla rete (esempio facebook o youtube ecc), ma solo servizi presenti nella rete aziendale, per evitare che i lavoratori perdano tempo durante la giornata. Come posso modellare questo semplice requisito? In

maniera piuttosto semplice possiamo definire una serie di dati tra cui in questo specifico caso, l'indirizzo sella sottorete rete, prendiamo come esempio l'indirizzo IpSrc=10.0.1.0/24 e alcuni campi come porta sorgente (pSrc) e porta destinazione (pDst) siano settati con il valore pari a '*'. Il simbolo '*' indica che qualsiasi valore associato al quel campo è concesso.

$$r_1 : ((IP_{Src} = 10.0.1.0/24, IP_{Dst} = *, pSrc = *, pDst = * \\ tProto = TCP, domain = facebook.com, url = *, days = (Monday - Friday) \\ timeInt = *), log, deny) \tag{4.1}$$

Dall'altro lato abbiamo una catalogo di VNF disponibili per gli utenti. Supponiamo di avere a disposizione le seguenti VNF.

1. iptables: firewall stateless e stateful, packet filter (con modulo aggiuntivo per la gestione del timer)
2. Squid: reverse proxy, firewall di livello 7, filtraggio basato su tempo, packet filter
3. OWASP ModSecurity: firewall di livello 7, packet filter
4. Strongwan: protezione VPN

A questo punto vogliamo agevolare l'utente nella scelta delle VNF da prendere in considerazione per portare a termine l'enforce del requisito. Come scegliamo la scelta di queste? Ne basta una ? Due ? Come sarà la configurazione delle rispettive VNF ?
L'indea che è alla base per portare al termine il lavoro è quello di determinare come queste VNF sono in grado di saper fare l'enforce del requisito andando ad estrarre i campi che la VNF avrebbe configurato per essere conforme al requsito. Questo tipo di processo si chiama **Mapping 1-1** tra Requisito e VNF. Il risultato di questa operazione scaturisce la creazione dell'entita funzionalità che riassume come la VNF aderisce al requisito. In maniera piu generica enumeriamo le VNF sopra citate come VNF1, VNF2 ecc.. Supponiamo di avere in output le seguenti funzionalità:

- $f_1 = (IP_{src} = 10.0.1.0/24), (log, deny)$ - VNF1
- $f_2 = (IP_{src} = 10.0.1.0/24), (log)$ - VNF2
- $f_3 = (IP_{src} = 10.0.1.0/24), (deny)$ - VNF3
- $f_4 = (Domain = facebook.com), (deny)$ - VNF4

- $f_5 = (IP_{src} = 10.0.1.0/24, Domain = facebook.com), (deny)$ - VNF5

Un ulteriore step è necessario. Le domande che ci ponimao raggiunto questo traguardo sono: *"Siamo in grado avendo a disposizione queste funzionalità di supportare a pieno il requisito? E' possibile trovare un modo di combinare queste funzionalità al fine di fare l'enfoce del requisito ?"* Alcune possibili soluzioni sono riportate qui sotto:

1. f_1
2. f_2, f_3
3. f_2, f_4
4. f_2, f_5

Alcune di queste soluzioni non sempre sono facili da individuare soprattutto quando in gioco ci sono piu campi e sopratutto piu azioni. Infatti sono proprio questi due parametri che vanno ad dimensionare lo spazio delle soluzioni possibili.

Il risultato di questa operazione di affinamento è un insieme di tutti i possibili elenchi di possibili regole non ottimizzate che potrebbero essere configurate per una funzionalità di sicurezza. In altre parole, è l'insieme di tutte le possibili soluzioni non ottimizzate per l'enforce di un requisito di sicurezza. Questo processo è svolto interamente dal modulo LER. Esso è incaricato con l'aiuto dell' ottimizzatore Gurobi di provvedere alle generazioni di soluzioni fornendogli relazioni matematiche tra le funzionalità, eventuali vincoli e variabili binarie.

Di seguito andremo ad osservare più da vicino i vari modelli di Manifest di una VNF, di Requisito e di funzionalità e infine il modello dei due moduli RDF e LER.

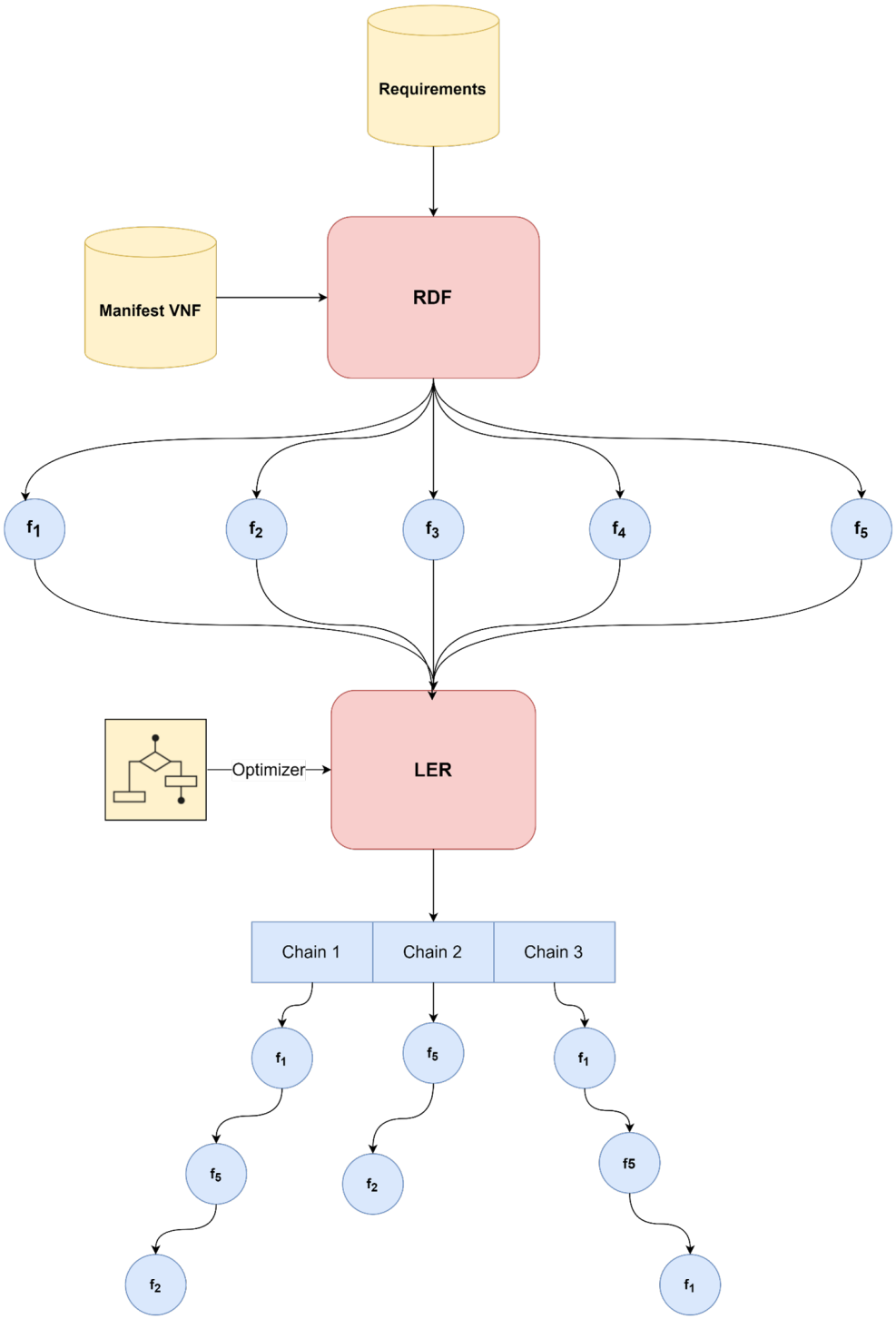

Figura 4.1: Schema del workflow

4.3 Manifest di una VNF

Una Virtual Network Function (VNF) è una specifica implementazione di una funzione di sicurezza tramite software. L'insieme di queste VNF disponibili all'interno del sistema per gli utenti VEREFOO è indicato con V_T.

Una VNF $v \in V_T$ viene modellata attraverso una Manifest espressa da una coppia di dati:

$$M_v = (F_v, A_v) \tag{4.2}$$

- F_v è l'insieme dei Fields. Un campo è un'informazione in base alla quale la VNF può prendere una decisione. Ad esempio un campo di un pacchetto oppure un parametro di configurazione.
- A_v è l'insieme delle azioni supportate. Un'azione è un'operazione che una VNF può effettuare quando tutte le condizioni sui campi sono soddisfatti.

Quando parliamo di un campo di una VNF è necessario andare specificare come questi possono essere configurati dalla VNF stessa. E' possibile identificare tre tipi di insieme di campi di una VNF.

1. F_v+ include tutti quei campi che possono essere assegnati nella configurazione della VNF , e allo stesso tempo quei campi per cui essa è in grado di prendere delle decisioni. Esempio: la 5-tupla dei campi per iptables.
2. F_v* include tutti quei campi che non possono essere assegnati nella configurazione della VNF (tipicamente sono insieriti sotto forma di '*' nella configurazione) e allo stesso prendono parte alle decisioni. Esempio: il campo Dominio or Url per la vnf iptables.
3. F_v- include tutti quei campi che non possono essere assegnati nella configurazione della VNF e che non sono in grado di prendere delle decisioni. Esempio: lunghezza della chiave di cifratura per la vnf iptables.

La suddivisione dei campi in queste tre categorie nasce da una serie di casistiche piuttosto reali che ha spinto alla definizione di questi tre insiemi. Un esempio che mette in luce l'importanza e l'utilità è la Figura 4.2.

In Figura 4.2 possiamo notare in basso a sinistra la presenza di due requisiti molto minimali, in cui abbiamo due campi ed un azione. La domanda che ci poniamo è :

È possibile utilizzare un VNF come iptables per applicare questi requisiti?

Risposta:
NO. Non è possibile utilizzare un packet filter in questa topologia, perché i requisiti

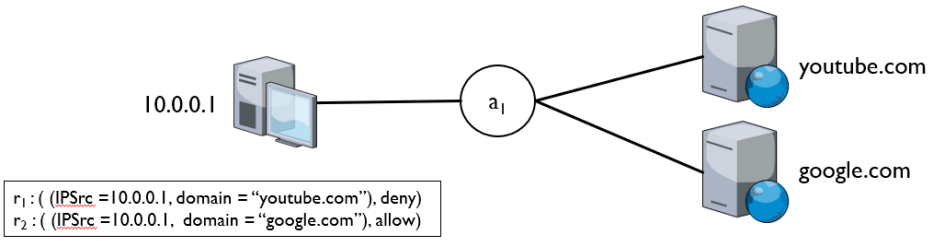

Figura 4.2: Un possibile esempio topologico di una rete

condividono lo stesso indirizzo IP di origine e non vengono fornite informazioni sulle destinazioni degli indirizzi IP.

L'eventuale uso di questa VNF come iptables non avrebbe portato alcun successo, anzi avremmo inserito policy che andrebbero in conflitto tra loro, creando un comportamento indesiderato e del tutto imprevedibile.

Vediamo ora uno scenario simile a quello precedente ma con una leggera variante (Figura 4.3).
La domanda che ci poniamo è sempre la stessa :

È possibile utilizzare un VNF come iptables per applicare questi requisiti?

Risposta:
SI. È possibile allocare un packet filter in a_1 per bloccare tutto il traffico proveniente da 10.0.0.1, perché 'google.com' può essere raggiunto nell'altro percorso.

Un packet filter in questo caso può imporre un requisito con parametri relativi al livello di applicazione, anche se i parametri a livello di applicazione non fanno parte della sua configurazione!

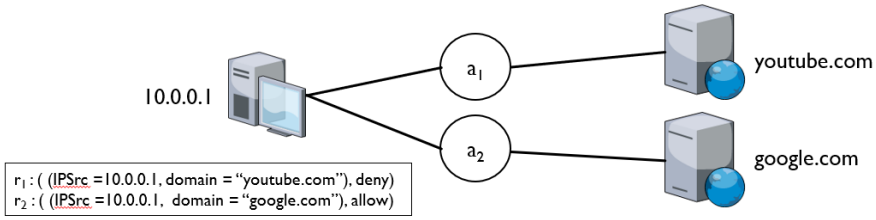

Figura 4.3: Un possibile esempio topologico di una rete

Questi due mini esempi mostrano come la suddivisione dei campi nelle tre categorie ha una ragione ben precisa e permette di andare a sceglire le VNF guardando anche questi aspetti che non sono del tutto trascurabili. Scenari che si possono presentare soprattutto in grandi reti piuttosto che in reti domestiche.

4.4 Requisito

Un requisito è l'espressione di una policy di sicurezza, usando un linguaggio di medio livello astratto. Esso può espresso attraverso due macro insiemi di informazioni:

1. L'insieme dei campi denominato con il nome Fields (F). Sono informazioni utili per poter esprimere il comportamento desiderato di un traffico che proviene da una sorgente e rivolto ad una destinazione. Questa informazione viene indicato con B.

2. L'insieme di azioni A. Azioni da applicare nel momento in cui le condizioni sui campi Fields sono verificate. Questo informazione viene indicato con S

Queste infomazioni possono essere espresse nella forma :

$$r = (B_r, S_r) \tag{4.3}$$

Dove B_r :

- B_r è la coppia campo-valore dove $B_r \subset F$

Mentre $S_r = (a, B_p)$:

- $a \in A_r$ dove A_r è il set di azioni con $A_r \subset A$ - Ogni azione $a \in A_r$ proviene da un insieme di azione A.

- B_p campi parametrici dove $B_p \subseteq B_r$; I campi parametrici sono campi coinvolti in prima persona nell'azione $a \in A_r$

Quando parliamo di coppia di informazioni campo valore, facciamo riferimento al modo in cui è possibile associare un valore o piu valori ad un campo:

- $b_1 = IP_{src} = 1.2.3.4/24$ - definiamo un specifico valore

- $b_2 = Domain = *$ - con il simbolo '*' specificamo qualsiais valore associabile.

- $b_1 = Port_{src} = [80 - 110]$ - definiamo un range di valori

La notazione di S_r è stata introdotta per permette di andare a definire particolari comportamenti che si possono ottenere andando ad esprime un ordinamento delle azioni stesse:

- Ordinato - indicato con le parentisi quadre [ ].
  Esempio : $[(a_1, B_p), (a_2, B_p), (a_3, B_p)]$ - In questo caso si forza l'ordinamento delle azioni. L'applicazione dell'azione a_3 può avverine solo ed esclusivamente dopo l'azione a_2.

- Concorrente - indicato con le parentesi angolari <>.
  Esempio: $< (a_1, B_p), (a_2, B_p), (a_3, B_p) >$ -Indichiamo che queste azioni devono essere svolte insieme.

- Non ordinata : $(a_1, B_p), (a_2, B_p), (a_3, B_p)$ - L'ordine con cui queste azioni devono essere applicate non ha importanza.

La rappresentazione di un requisito in maniera sintetica puo essere vista come una serie di dati utilizzabili nella configurazione delle VNF.

Quando andiamo ad esprimere un requisito con questa rappresentazione non teniamo conto :

1. Se esiste un manifest di una VNF che supporta tutti i campi diversi dal valore '*'

2. Se esiste almeno una manifest di una VNF ingrado di supportare le azioni

3. di eventuali vincoli basati sualla topologia della rete.

L'esempio di un espressione di un requisito è il seguente

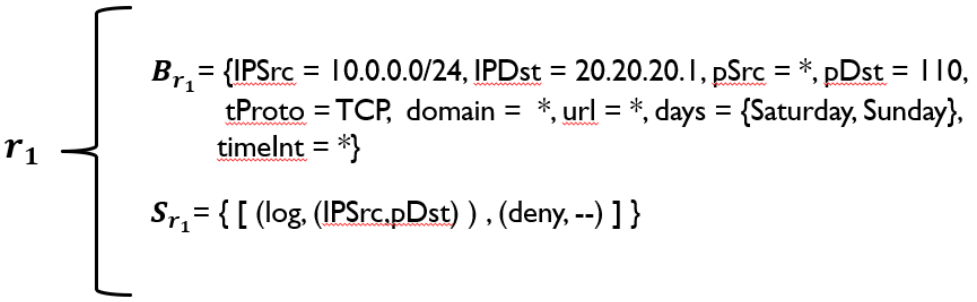

Figura 4.4: Espressione di un requisito.

4.5 Moduli RDF e LER

In questa parte del capitolo andremo a osservare da vicino come i due moduli citati prima lavorano e come cooperano al fine di indentificare le soluzioni se possibile.

4.5.1 RDF - Requirement Driven Functionality

Questa fase è il punto di ingresso del processo e introduce il concetto di Funzionalità. Una funzionalità è una rappresentazione di come un VNF all'interno del catalogo

può soddisfare un requisito. Si ottiene tramite una mappatura tra il requisito e il manifest di una VNF.

Una funzionalità può soddisfare il requisito in maniera:

1. completa: la funzionalità è in grado di comprendere tutte le informazioni e sviluppare una configurazione corrispondente. Con il termite tutte le informazioni si intende tutti i valori associati ai fields e tutte le azioni richieste.

2. parziale: la funzionalità comprende parzialmente le informazioni e può configurare solo un sottoinsieme dei dati. Si intende tutte le azioni richieste, e almeno un valore associato ai fields.

Tutte le funzionalità ottenute attraverso il processo di mapping, non è possibile prevedere a priori se:

- La funzionalità farà parte sicuramente di una soluzione;
- La funzionalità entrerà a far parte in piu di una soluzione;
- In futuro la funzionalità verrà scartata per alcuni vincoli basati sulla topologia.

La rappresentazione della funzionalità è la seguente :

$$f = (B_f, A_f) \tag{4.4}$$

- $B_f = B_r \cap B_v$
- $A_f = A_r \cap A_v$

Una funzionalità viene definita accettabile se è solo se $A_f \neq 0$ e $B_f \neq 0$. Il motivo di questo vincolo è principalmente uno: Una funzionalità in grado di riconoscere solo i campi di un requisito senza saper applicare un azione è inutile. Stesso discorso vale se una funzionalità è in grado di riconoscere le azioni senza saper comprendere i valori dei campi.

Questa operazione comporta conseguentemente operazioni di pruning. Questo processo permette in anticipo di capire se quella funzionalità ha senso di esistere o meno. Alcune operazioni di pruning sono descritte qui di seguito:

- Scartare quelle funzionalità che non possono applicare azioni su un insieme di campi parametrici B_p, se presenti .
- Se il requisito richiede azioni simultanee (ovvero l'insime S_r del requisito si presenta nella forma : $S_r = \{< (a_1, B_p), (a_2, B_p) >\}$), scaratre le funzionalità che non sono in grado di applicare tutte le azioni.

Algorithm 1 Pseudo codice dell'algortimo RDF - Processo di mapping Requisito - VNF

```
public Functionality functionFindFunctionality(
   requirement,manifest){
    /** Verify if there are the minimal condition
   fields to procede to the next step */
    fieldcommon=functionCommonField(requirement_fields,
   manifest_fields);
    if(fieldcommon in empty)
        return; // Functionality has no sense

    for(field in fieldcommon){
        checkSupportOneOrMoreConstraintOnField(field,
   requirement_field))
        checkSupportForTypeConfigValue(field,
   requirement_field)
    }

    /** Verify if there are the minimal conditions on
   action to procede to the next step */

    actioncommon = functionCommonAction(
   requirement_actions, manifest_actions);
    if(actioncommon is empty)
        return; // Functionality has no sense

    for(action in actioncommon)
        checkActionhaveFieldParametric(action,
   manifest_field,requirment_actions)
        checkConstraintAction(action,requirment_actions)

    }
    if(actionRequirmentCocurrent(requirment_actions)){
        checkContainsAllActionConcurretn(actioncommon,
   actionconcurrent)
    }
    return new Functionality(actioncommon,fieldcommon);
}
```

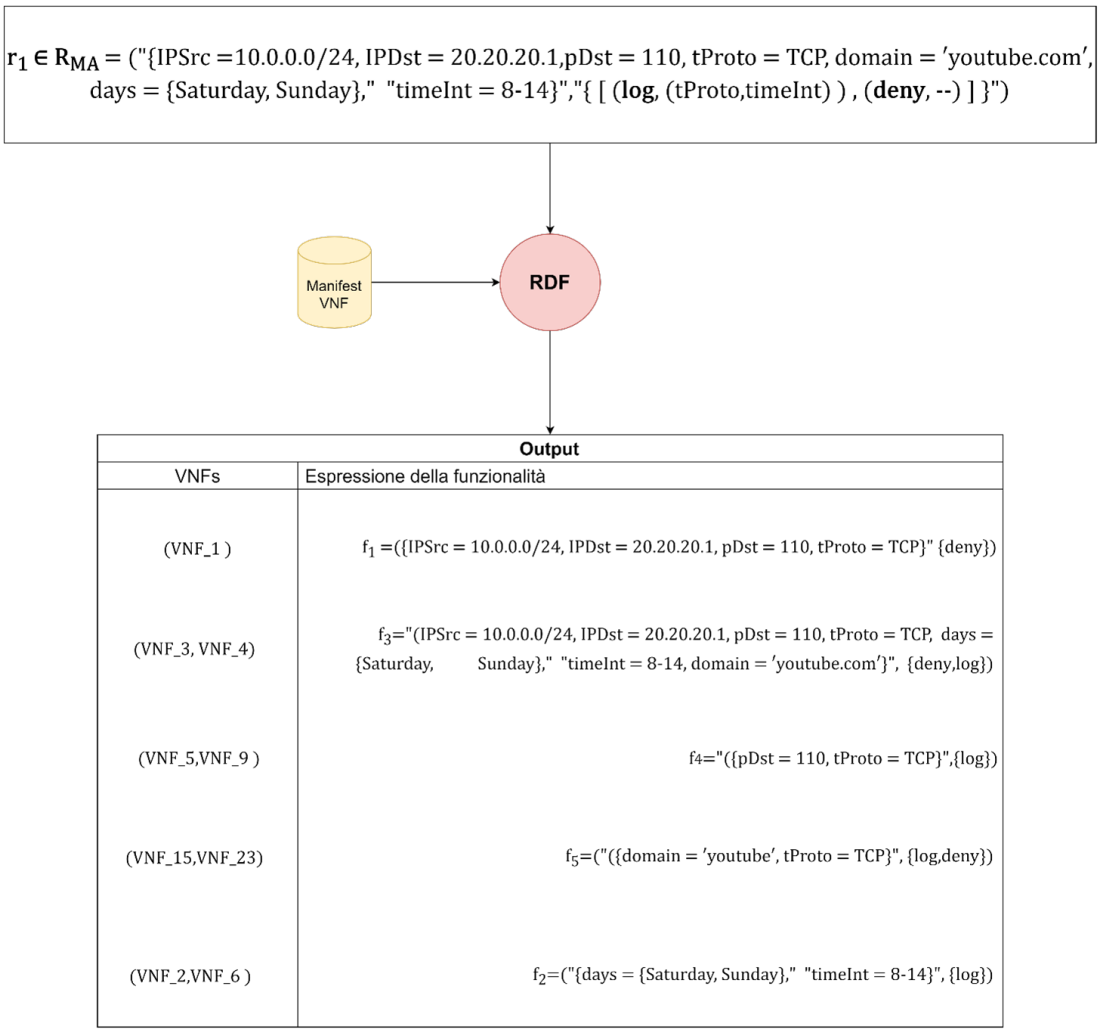

Figura 4.5: Esempio RDF in azione

4.5.2 I passi dell'algortimo

L'algoritmo può essere sintentizzato in due macro fasi:

1. Individuazione dei campi in comune tra requisito e VNF: Questa fase è una delle fasi delicate per la costruzione corretta della funzionalità, questo perché ogni singolo campo field della VNF è marcato nei 3 modi elencati precedentemente (Sezione:4.3).

   - Mapping tra field requisito e campo configurabile e decisionale.
     Caso più semplice, ovvero la vnf su quel specifico campo è in grado di

configurare il campo, pertanto il risultato del mapping sarà dato dal valore del capo seguito dal valore associato. Es: Procotol=TCP

- Mapping tra field requisito e campo non configurabile ma decisionale: La vnf sul quel specifico campo non è in grado di configurare il campo, ovvero non è in grado di saper comprendere il significato pertanto il risultato del mapping sarà dato dal nome del campo seguito dal valore associato '*' . Esempio: il campo *'domain'* per la vnf *'iptables'* . Sarà settato come Domani=*.
- Mapping tra field requisito e campo non configurabile e non decisionale: Il caso in cui per quella VNF non viene specificato il supporto di quel campo , in sostanza non compare nel manifest. Mapping vuoto per quel campo.

Prima di procedere con lo step successivo è necessario che questa operazione si concluda almeno con l'identificazione di un campo. Se così non fosse la funzionalità sarebbe priva di senso, ovvero una funzionalità che non è in grado a livello di campo di riconoscere le specifiche per ogni campo definite dal requisito.

2. Riconoscimento delle azioni in comune tra requisito e VNF.
   Questa fase è la seconda ed ultima fase, dove partendo dalle azioni richieste dal requisito verifico se nel manifest della vnf in analisi è presente. In caso di esito negativo, la funzionalità non viene creata in quanto priva di senso. Avendo a disposizioni la seguente lista si effettua un ulteriore analisi.

   - Se nel requisito sono presenti azioni marcate come 'Concorrenti' allora necessariamente tutte le azioni devo essere nell'insieme individuato al passo precedente. Se cosi non fosse quell' azione/i non può essere associata alla funzionalità corrente.
   - Se il requisito impone che determinate azioni debbano essere svolte in compartecipazione con determinati campi, allora è necessario assicurarsi che l'insieme dei field precedenti includa anche i campi dipendenti. Esempio voglio effettuare il *Log* solo per il tipo di protocollo usato per raggiungere determinate destinazioni. Il l'azione *Log* ha una stretta relazione con il field *Protocol*.

Ciò che è stato descritto rappresentano le fasi rilevanti delle operazioni svolte nel modulo RDF. Una rappresentazione grafica di questo processo è alla Figura 4.6.

Si intuisce facilmente come questo processo comporta un dispendio di risorse e di tempo se il numero di VNF a disposizione inizia ad essere significativo. Per poter svolgere questo lungo processo in tempi relativamente brevi si è scelto di

parallelizzare il processo di mapping. Tra le possibili soluzioni che si possono adottare, si è scelto di utilizzare un approccio multithread basato sul pattern ThreadPool.

Un pool di thread riutilizza i thread creati in precedenza per eseguire le attività correnti e offre una soluzione al problema dell'overhead del ciclo dei thread e del rilascio delle risorse. Poiché il thread è già esistente quando arriva la richiesta, il ritardo introdotto dalla creazione del thread viene eliminato, rendendo l'applicazione più reattiva. In Java viene usato il framework Executor. Utilizzando l'Executor, è sufficiente implementare il codice e inviarlo all'Executor per l'esecuzione.

Il pattern Thread Pool aiuta a risparmiare risorse in un'applicazione multithread e anche a contenere il parallelismo in determinati limiti predefiniti.

Quando si utilizza un pool di thread si scrive il codice concorrente sotto forma di attività parallele e le si invia per l'esecuzione a un'istanza di un pool di thread. Questa istanza controlla diversi thread e permette di riutilizzati per eseguire nuove attività.

```
ExecutorService executor=Executors.newFixedThreadPool(
   NUMBERTHREAD);
for (String nameFunction: VNFs.keySet()) {
Runnable worker = new BuildFunctionality(nameFunction,
   requirement,
         VNFs.get(nameFunction),
         mapToVnfSupport,mapToFunctionality);
      executor.execute(worker);
}
executor.shutdown();
while (!executor.isTerminated()) {}
```

Ogni thread al termine del lavoro andrà popolare il set di funzionalità utile al modulo LER dove prenderà in carico il set appena trovato e lo userà per produrre le catene di funzionalità ottime per l'enforce del requisito.

4.5.3 LER - Logic Enforce Requirement

Il modulo ler rappresenta l'ultimo step di elaborazione dei dati. Date le funzionalità messe a disposizione dal modulo precedente il blocco LER puo attivarsi cercando di trovare tutti i possibili insiemi di funzionalità - catena di funzionalità (f_{result}) tale per cui:

- Può applicare tutte le azioni richieste dal requisito:
  $B_{fresult} \subset Br$ e $A_{fresult} \subseteq A_r$.

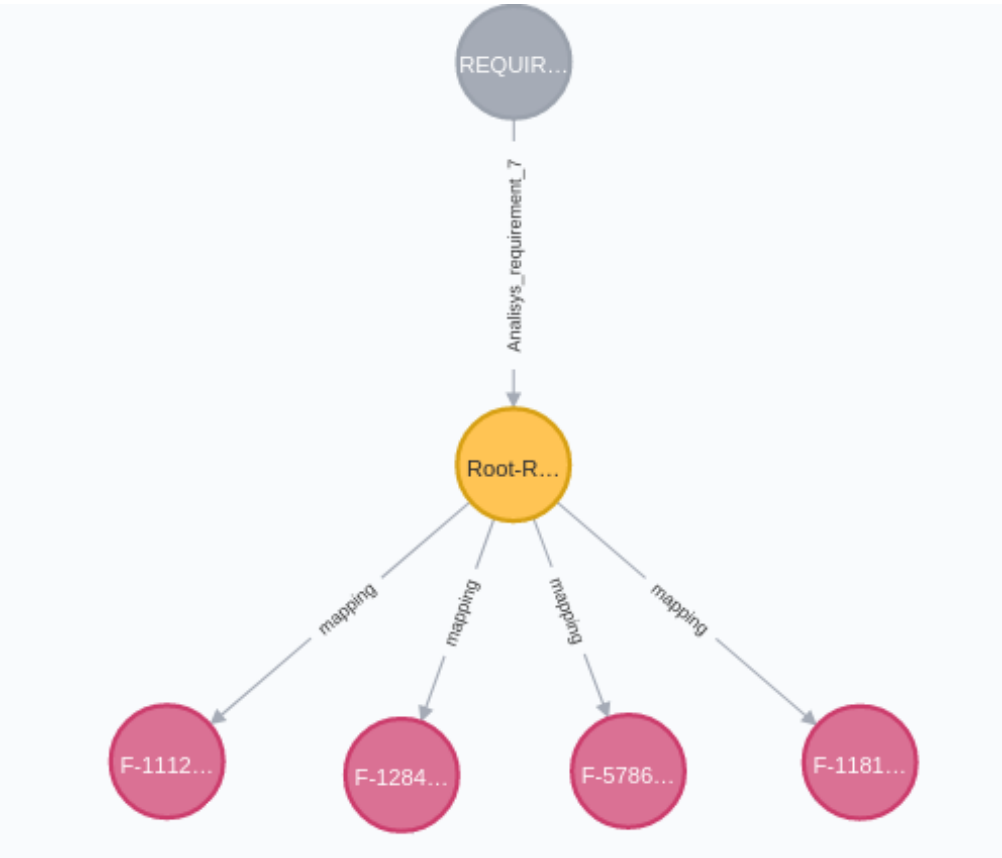

Figura 4.6: Grafico a nodi - Modulo RDF

- può applicare azioni ed essere in grado di riconoscere tutti i campi del requisito. $B_{fresult} \subseteq Br$ e $A_{fresult} \subseteq A_r$.

Il processo inizia estraendo un set dall'insieme di funzionalità. Il set estratto andrà a costruire un ulteriore funzionalità chiamata 'funzionalità risultante' ottenuta unendo gli insiemi B_f e di A_f di tutte le funzionalità del set selezionato. La nuova funzionalità f_{result} esprime un sommario di quelle che è la potenzionalità ottenuta attraverso l'unione di essi. Il processo non è finito, perchè affinchè la risultante funzionalità sia accettabile è necessario verificare se l'ordine delle azioni imposte dal requisito può essere supportato (se definito).

Queste situazioni introducono due strategie per garantire l'enforce dei requisiti:

- Replica della stessa funzionalità in cui diverse instanze si occupano di svolgere diverse azioni.
- Funzionalità in cui viene forzata l'esecuzione dell'ordine di azioni. La presenza di un ulteriore parametro come *'Priority'* è necessario. Esso serve a

discriminare quale delle tante azioni assegnate deve essere svolta per prima.

Il risultato di questa operazione è un insieme di tutte le possibili liste di eventuali funzionalità non ottimizzate configurabili. In altre parole, è l'insieme di tutte le possibili soluzioni non ottimizzate in grado di soddisfare il requisito.

Maggiori dettagli sono presenti nel capitolo 6, in cui verrano spiegati quali sono le condizioni e vincoli utilizzati dall'ottimizzatore che permettono di ottenere i risultati desiderati.
In Figura 4.7 è visibile graficamente le soluzioni di catene di funzionalità ottenute con il tool Neo4j. Per scoprire come queste sono concatenate, è sufficiente seguire gli archi partendo dal nodo colorato in giallo chiamato *Root-LER*. Le soluzioni sono i nodi colorati di rosso che si attraversano. La catena termina quando non ci sono più archi diretti verso altri nodi. Es: Supponiamo di volere estrapolare tutte le soluzioni presenti nella figura 4.7 allora :

- $Soluzione_0 : F : 5786 + F : 1284$
- $Soluzione_1 : F : 1112 + F : 1284$
- $Soluzione_2 : F : 5786 + F : 1181$
- $Soluzione_3 : F : 1112 + F : 1181$

La notazione **F:5786** indica che stiamo considerando una *Funzionalità* con identificativo pari *5786*. L'indetificativo è un numero univoco assegnato in fase di creazione ed è utilizzato per destingure tutte le funzionalità ottenute durante il processo di mapping.

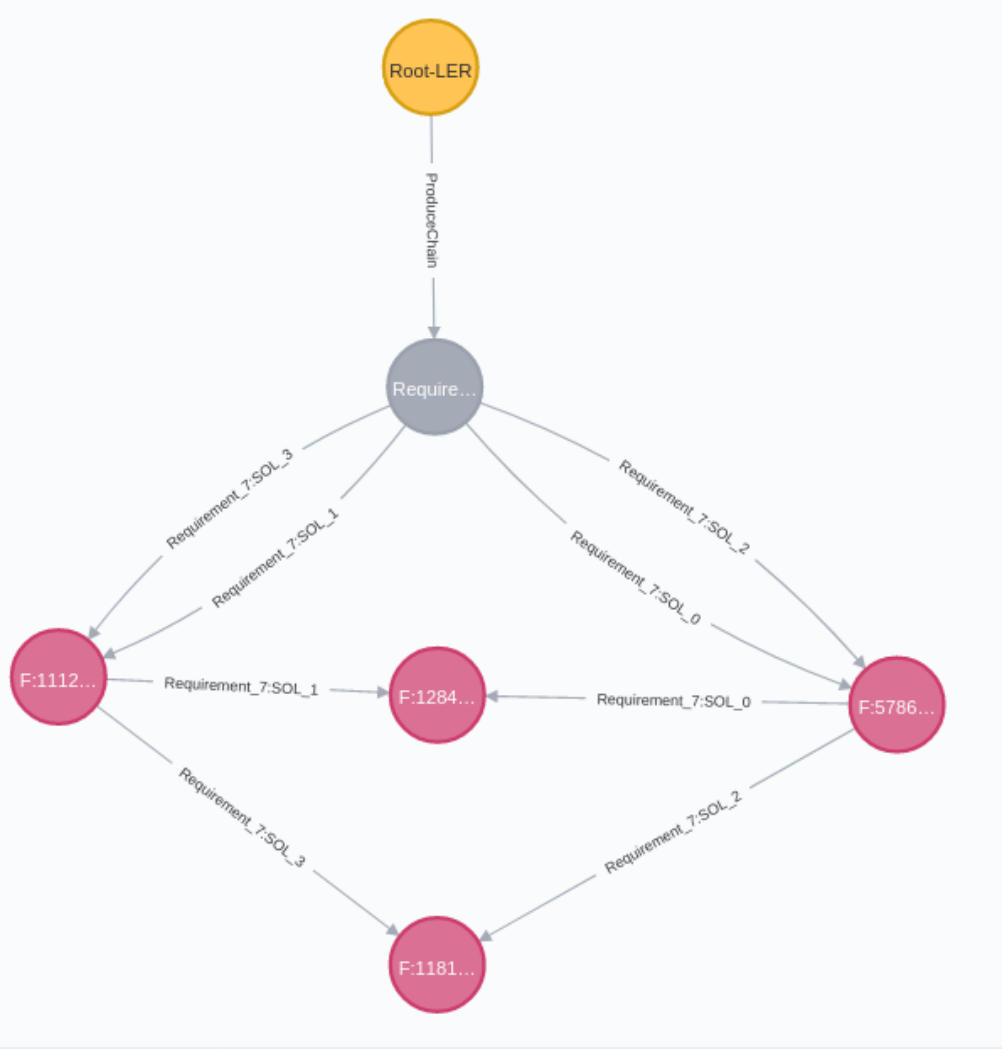

Figura 4.7: Grafico a nodi - Modulo LER

Capitolo 5

Modello del requisito e delle virtual function

In questo capitolo descriveremo e analizzeremo i file XML che servono da modello per i requisiti, per il catalogo VNF e per le funzionalità. Verranno mostrati anche alcuni casi d'uso.

5.1 Requisito

Il Requisito permette agli amministratori di rete di definire, in maniera semplice ma completa, tutte le richieste dell'amministratore di rete.

L'elemento radice del file xml è *ReuirementsList*:

```
<element name="RequirementsList">
  <complexType>
    <sequence>
       <element ref="tns:Requirement" minOccurs="1" maxOccurs="unbounded"/>
    </sequence>
  </complexType>
  ...
</element>
```

E' composto principalmente da una sequenza di entità chiamate *Requirement*. Ogni requisito ha una propria struttura interna, definita qui di seguito :

```
<element name="Requirement">
<complexType>
  <sequence>
    <element name="Info" minOccurs="1" maxOccurs="1">
       <complexType>
          <sequence>
             <element name="Description" type="string"/>
             <element name="Administrator" type="string"/>
          </sequence>
       </complexType>
    </element>
    <element ref="fd:Fields"    ></element>
    <element name="Actions"     type="tns:ActionSetType"/>
  </sequence>
<attribute name="Req_ID" type="integer" use="required"/>
</complexType>
...
</element>
```

E' composta principalmente dai seguenti elementi:

- *Description*, usato per descrivere qual'è l'intento del requisito che si sta definendo;
- *Administrator*, utile per tenere traccia chi è l'autore del requisito;
- *Fields*, l'insieme dei parametri/campi che il requisito ritiene necessario configurare;
- *Actions* l'insieme di azioni da applicare sul traffico di rete.

Le entità piu importanti sono i Fields e Actions. Esse rappresentano il core del requisito, ma data le complessità verranno analizzate uno alla volta.

5.1.1 Fields

I fieds rappresentano il modo in cui un amministratore di rete definisce i campi per un determinato traffico. La lista dei campi che il requisito può configurare sono i seguenti:

Ip

```
<element name="IpSrc"   minOccurs="0" maxOccurs="1">
<complexType>
   <attribute name="Version" type="tns:versionOfIp" use="required"/>
   <attribute name="Address" type="tns:IpAddress"   use="required"/>
</complexType>
</element>
<element name="IpDst"   minOccurs="0" maxOccurs="1">
<complexType>
   <attribute name="Version" type="tns:versionOfIp" use="required"/>
   <attribute name="Address" type="tns:IpAddress"   use="required"/>
</complexType>
</element>
```

Il campo *Ip* è composto da due attributi:

- *versionOfIp*, Versione dell'indirizzo ip, può essere di due valori ipv4 o ipv6.
- *IpAdress*, il valore dell'indirizzo di rete espresso attraverso un espressione regolare.

```
<simpleType name="IpAddress">
<restriction base="string">
  <pattern value="(((1?[0-9]?[0-9]|2[0-4][0-9]|25[0-5])\.)
    {3}(1?[0-9]?[0-9]|2[0-4][0-9]|25[0-5])\/([0-9]|[1-2][0-9]|3[0-2]))"></pattern>

<pattern value="((1?[0-9]?[0-9]|2[0-4][0-9]|25[0-5])\.)
    {3}(1?[0-9]?[0-9]|2[0-4][0-9]|25[0-5])" />
  <pattern value="([A-Fa-f0-9]{1,4}:){7}[A-Fa-f0-9]{1,4}" />
  <pattern value="[\*]" />
</restriction>
</simpleType>

<simpleType name="versionOfIp">
<restriction base="string">
   <enumeration value="ipv4"/>
   <enumeration value="ipv6"/>
</restriction>
</simpleType>
```

Port

```
<element name="PortSrc" minOccurs="0" maxOccurs="1">
  <complexType>
    <attribute name="Range"        type="tns:portvalue"       use="required"/>
    <attribute name="RangeSetting" type="tns:rangeSettings"   use="required"/
  >
  </complexType>
</element>
<element name="PortDst" minOccurs="0" maxOccurs="1">
  <complexType>
    <attribute name="Range"        type="tns:portvalue"       use="required"/>
    <attribute name="RangeSetting" type="tns:rangeSettings"   use="required"/
  >
  </complexType>
</element>
```

Il campo *Port* è composto da due attributi:

- *portValue*, il valore delle porte esprimibili attraverso valori numeri compressi tra 0 e il suo valore massimo 65536. E' espresso attravero un espressione regolare. Esempio: 10 - 1500
- *rangeSettings*, è tipo di intervallo che si vuole considerare. Se si tratta di intervallo esterno o un intervallo interno.

```
<simpleType name="portvalue">
  <restriction base="string">
    <pattern value="()([1-9]|[1-5]?[0-9]{2,4}|6[1-4][0-9]{3}|65[1-4]
    [0-9]{2}|655[1-2][0-9]|6553[1-5])"></pattern>
    <pattern value="()([1-9]|[1-5]?[0-9]{2,4}|6[1-4][0-9]{3}|65[1-4]
    [0-9]{2}|655[1-2][0-9]|6553[1-5])-()([1-9]|[1-5]?[0-9]{2,4}|6[1-4]
    [0-9]{3}|65[1-4][0-9]{2}|655[1-2][0-9]|6553[1-5])"/>
    <pattern value="[\*]" />
  </restriction>
</simpleType>
```

Protocollo

```
<element name="Proto" minOccurs="0" maxOccurs="1">
  <complexType>
    <attribute name="TypeProtocol" type="tns:protype" ></attribute>
  </complexType>
</element>
```

Il campo in questione permette di esprire quale tipo di protocollo si intende utilizzare. Esso è definito attravero un enumerazione del campo stesso in cui sono presenti tutti gli acronimi dei protocolli.

```
<simpleType name="protype">
      <restriction base="string">
        <enumeration value="TCP"/>
        <enumeration value="UDP"/>
        <enumeration value="HTTP"/>
        <enumeration value="HTTPS"/>
        <enumeration value="TLS"/>
        <enumeration value="IPSEC">
        <enumeration value="SMTP"/>
        <enumeration value="POP3"/>
        <enumeration value="IMAP"/>
        <enumeration value="ICMP"/>
        <enumeration value="IP_ESP"/>
        <enumeration value="IP_AH"/>
        <enumeration value="GRE"/>
        <enumeration value="FTP"/>
        <enumeration value="SSH"/>
        <enumeration value="SSL"/>
        ...
        <enumeration value="*"/>
      </restriction>
  </simpleType>
```

Urls

Urls è composto da una serie di valori Url, seguiti dall'attributo urlValue che rappresente il valore url stesso.

```
<element name="Urls" type="tns:listanyURI"  minOccurs="0" maxOccurs="1"/>
<complexType name="listanyURI">
   <sequence>
      <element name="Url" maxOccurs="unbounded">
         <complexType>
            <attribute name="URLvalue" type="anyURI"/>
         </complexType>
      </element>
   </sequence>
</complexType>
```

Days And Times

Rappresenta l'insieme dei dati per esprimere il concetto di tempo espresso attraverso tre campi sostanziali:

- *Day*, rappresenta il giorno della settimana
- *TimeStart, TimeEnd*, rappresenta l'intervallo di tempo.
- *TimeZone*, indica la differenza di tempo tra le varie zone nel mondo.

```
<element name="DaysAndTimes" type="tns:DaysAndTimesType" minOccurs="0"
    maxOccurs="1"/>
<complexType name="DaysAndTimesType">
   <sequence>
      <element name="DayAndTime" minOccurs="1" maxOccurs="unbounded">
         <complexType>
            <attribute name="Day"       type="tns:Day"          use="required"/>
            <attribute name="TimeStart" type="tns:regExrTime"   use="optional"/>
            <attribute name="TimeEnd"   type="tns:regExrTime"   use="optional"/>

         </complexType>
      </element>
   </sequence>
   <attribute name="TimeZone" type="string" use="required"/>
</complexType>
```

Documents

Rappresenta l'insieme dei dati per esprimere l'oggetto documento. DocumentType esprime la lista dei valori associabili al campo.

```
<element name="Documents"       type="tns:Documents"    minOccurs="0"
    maxOccurs="1"/>
<complexType name="Documents">
   <sequence>
      <element name="Document"  minOccurs="1" maxOccurs="unbounded">
         <complexType>
            <attribute name="Type" type="tns:DocumentType" use="required"/>
            <attribute name="Regex" type="string" use="optional"/>
         </complexType>
      </element>
   </sequence>
</complexType>
```

La definizione del tipo complesso *DocumentType* è il seguente:

```
<simpleType name="DocumentType">
   <restriction base="string">
      <enumeration value="PDF"/>
      <enumeration value="DOC"/>
      <enumeration value="DOCX"/>
      ...
      <enumeration value="*"/>
   </restriction>
</simpleType>
```

Scripts

Rappresenta l'insieme dei dati per andare ad esprimere l'oggetto script, inteso come file in grado di essere eseguito. ScriptType rappresenta il valori associabili al campo scripts.

```
<element name="Scripts" type="tns:Scripts" minOccurs="0" maxOccurs="1"/>
<complexType name="Scripts">
    <sequence>
        <element name="Script"  minOccurs="1" maxOccurs="unbounded">
            <complexType>
                <attribute name="Type" type="tns:ScriptType" use="required"/>
                <attribute name="Regex" type="string" use="optional"/>
            </complexType>
        </element>
    </sequence>
</complexType>
<simpleType name="ScriptType">
    <restriction base="string">
        <enumeration value="JS"/>
        <enumeration value="BASH"/>
        <enumeration value="BAT"/>
        <enumeration value="PYTHON"/>
        <enumeration value="*"/>
    </restriction>
</simpleType>
```

Web pages

Rappresenta l'insieme dei dati per esprimere il concetto pagina web. Può essere usato sia come filtro sulla singola pagina web ma anche su alcuni contenuti presenti in essa. I valori associabili al campo Webpages sono espressi dal campo WebType.

```
<element name="WebPages"        type="tns:Webpages"        minOccurs="0"
    maxOccurs="1"/>
<complexType name="Webpages">
    <sequence>
        <element name="WebPage"   minOccurs="1" maxOccurs="unbounded">
            <complexType>
                <attribute name="Regex" type="string" use="optional"/>
                <attribute name="Type" type="tns:WebType" use="required"/>
            </complexType>
        </element>
    </sequence>
</complexType>
```

```
<simpleType name="WebType">
   <restriction base="string">
      <enumeration value="HTML"/>
      <enumeration value="ASPX"/>
      <enumeration value="TAG_A"/>
      <enumeration value="TAG_HREF"/>
      <enumeration value="TAG_IMG"/>
      <enumeration value="TAG_P"/>
      <enumeration value="*"/>
   </restriction>
</simpleType>
```

Applications

Rappresenta l'insieme dei dati utili per esprimere il concetto di applicazione. Il riconoscimento del traffico generato da un applicazione come per esempio Skype, Whatsapp, Google Duo ecc. Il concetto di applicazione va inteso come traffico generato da o verso una destinazione/applicazione. AppType rappresenta una lista di possibili valori associabili.

```
<element name="Applications" type="Applications" minOccurs="0" maxOccurs="1">
<complexType name="Applications">
   <sequence>
      <element name="Application"  minOccurs="1" maxOccurs="unbounded">
         <complexType>
            <attribute name="Type" type="tns:AppType" use="required"/>
            <attribute name="Regex" type="string" use="optional"/>
         </complexType>
      </element>
   </sequence>
</complexType>
<simpleType name="AppType">
<restriction base="string">
   <enumeration value="SKYPE"/>
   <enumeration value="GOOGLE_DUO"/>
   <enumeration value="TELEGRAM"/>
   <enumeration value="WHATSAPP"/>
   ...
   <enumeration value="*"/>
</restriction>
</simpleType>
```

Email

```
<element name="Emails"  type="tns:Emails" minOccurs="0" maxOccurs="1"></
    element>
<complexType name="Emails">
   <sequence>
      <element name="Email"   minOccurs="1" maxOccurs="unbounded">
         <complexType>
            <attribute name="Type" type="string" use="required"/>
            <attribute name="Regex" type="string" use="optional"/>
         </complexType>
      </element>
   </sequence>
</complexType>
```

Possiamo notare come i campi *Documents, Media, Applications, Scripts, Email* hanno in comune l'attributo *regex*, usato per esprime vincoli sull'entità selezionata attraverso una espressione regolare. Espressione regolare puo essere definita direttamente dall'amministratore di rete nell'atto della creazione di un requisito. Il campo *regex* permette di cercare dei particolari pattern all'interno del traffico. Es. filtrare tutte le email che hanno come contenuto «"Credit Card"». Essa trova applicazione in molti casi d'uso.

Sono presenti ulteriori campi supportati. Maggiori dettagli e rispettivi modelli sono disponibili in Appendice D.

5.1.2 Actions

All'amminitratore di rete è consentito esprimere delle azioni nel requisito. Le azioni dovranno essere svolte nel caso i cui i valori associati ai fields rispecchiano i valori presenti nel flusso di dati (pacchetto). Questa struttura è espresso attraverso il tipo complesso ActionSet.

ActionSet

```
<complexType name="ActionSetType">
<sequence>
  <element name="ActionSet"  minOccurs="1" maxOccurs="unbounded">
    <complexType>
       <sequence>
          <element name="ActionRequired" type="tns:ActionsReq" minOccurs="1"
     maxOccurs="unbounded"></element>
       </sequence>
       <attribute name="Order"         type="integer" use="optional"/>
       <attribute name="Concurrent"    type="boolean" use="optional" default="false"
     />
    </complexType>
  </element>
</sequence>
</complexType>
```

Gli attributi importanti in questa struttura sono i seguenti:

- *Order*, permette di definire una classifica di azioni. E' utile nel momento i cui vogliamo discriminare un azione rispetta ad un altra in termini di ordinamento, ovvero imporre che una determinata azione venga svolta prima rispetto ad un altra.
- *Concurrent*, permette di andare ad vincolare il modo in cui le azioni posso essere svolte. Settando *Concurrent* = *true*, si indica che quelle azioni appartenti all'insieme ActionSet dovranno essere svolte dalla stessa *funzionalità*.

ActionRequired

Questa struttra XML approfondisce la struttura *Action*. Si ha la possibilità di scegliere se determinati azioni dovranno essere svolte solo un set di campi. Inoltre si protrebbe avere la necessità che alcune azioni siano svolte con precisi vincoli, ad esempio basta pensare alla cifratura del traffico. Si vuole usare una cifratura AES-128 piuttosto che AES-256.

```
<complexType name="ActionsReq">
<sequence>
  <element name="FieldsParametric" minOccurs="0" maxOccurs="1">
    <complexType>
      <sequence>
        <element name="Field" minOccurs="1" maxOccurs="unbounded">
          <complexType>
            <attribute name="Ref" type="ReferField" use="required"/>
          </complexType>
        </element>
      </sequence>
    </complexType>
  </element>
</sequence>
<attribute name="Action" type="typeAction" use="required"/>
<attribute name="Constraint" type="string" use="optional" default=""/>
<attribute name="UpdateValue" type="string" use="optional" />
</complexType>
```

Gli attributi importanti che permettono di supportare queste tipo di casistiche sono i seguenti:

- *FieldsParametric*, definisce un set di campi sui quali l'azione *Action* dovrà essere svolta.
- *Constraint*, permette di andare ad vincolare il modo in cui l' azione deve essere svolte. Esempio vogliamo imporre la cifratura del traffico AES-128 o AES-256.
- *UpdateValue*, è utile per quelle azioni che effettuano una sovrascrittura di determinati campi del pacchetto. Ad esempio l'azione di Forward, consiste nella andare a modificare il valore associato al campo IpDst.

Le azioni supportate sono definite attraverso enumerazione delle stesse. Esse sono disponibili qui sotto.

```
<simpleType name="typeAction">
<restriction base="string">
    <enumeration value="ALLOW_TRAFFIC"></enumeration>
    <enumeration value="DENY_TRAFFIC"></enumeration>
    <enumeration value="LOG_TRAFFIC"></enumeration>
    <enumeration value="ALERT"></enumeration>
    <enumeration value="COUNT"></enumeration>
    <enumeration value="ENCRYPT_TRAFFIC"></enumeration>
    <enumeration value="AUTHN_TRAFFIC"></enumeration>
    <enumeration value="PROTECT_CONFIDENTIALITY"></enumeration>
    <enumeration value="PROTECT_INTEGRITY"></enumeration>
    <enumeration value="LIMIT_BANDWIDTH"></enumeration>
    <enumeration value="FORWARD_TRAFFIC"></enumeration>
    <enumeration value="STORE_PACKET"></enumeration>
    <enumeration value="NAT_TRAFFIC"></enumeration>
    <enumeration value="REMARK_PACKET"></enumeration>
    <enumeration value="ANTIVIRUS"></enumeration>
    <enumeration value="IDS"></enumeration>
    <enumeration value="IPS"></enumeration>
    <enumeration value="PROXY"></enumeration>
    <enumeration value="PROTECT_DDOS"></enumeration>
    <enumeration value="SCANNING"></enumeration>
    <enumeration value="PROTECT_TRACKING"></enumeration>
    <enumeration value="PROTECT_IDENTITY"></enumeration>
    <enumeration value="VPN_TRAFFIC"></enumeration>
</restriction>
</simpleType>
```

5.2 Virtual Network Function

Il catalogo VNF rappresenta l'insieme delle funzioni di rete disponibili nel sistema. Questo catalogo è stato creato con l'obiettivo di fornire agli amministratori un modo semplice per elencare in dettaglia tutte le funzioni di rete disponibili nel sistema. Il catalogo è stato costruito in modo da poter contenere tutte le informazioni relative alle funzioni. La struttura è la seguente:

```
<element name="ManifestVNF" minOccurs="1" maxOccurs="unbounded">
<complexType>
<sequence>
<element name="Description" type="string" minOccurs="1" maxOccurs="1"/>
<element name="GeneralInfo" type="GeneralInfo" minOccurs="0" maxOccurs="1"/>
<element name="SoftwareInfo" type="SoftwareInfo" minOccurs="0" maxOccurs="1"/
    >
<element name="HardwareInfo" type="HardwareInfo" minOccurs="0" maxOccurs="1"
    />
<element name="ManifestInfo" minOccurs="1" maxOccurs="1">
  <complexType>
    <sequence>
       <element ref="fd:FieldsManifest" minOccurs="1" maxOccurs="1"/>
       <element ref="ac:ActionsManifest" minOccurs="1" maxOccurs="1"/>
    </sequence>
  </complexType>
</element>
</sequence>
<attribute name="Manifest_ID" type="string" use="required"></attribute>
</complexType>
</element>
```

Analizziamo nel dettaglio ciascuno elemento presente nella struttura Manifest VNF:

Description

Permette di fornire una sintesi piu o meno dettagliata della VNF. E' Utile per comprendere con una lettura le specifiche delle VNF che si sta andando a definire.

GeneralInfo

GeneralInfo rappresenta le informazioni sul venditore/fornitore e sul sito web. È composto dai seguenti elementi:

- Venditore, è una stringa che rappresenta il nome del venditore;
- WebSite, è l'URI dove è possibile trovare ulteriori informazioni sul NSF.

```
<complexType name="GeneralInfo">
   <sequence>
      <element name="WebSite" type="anyURI" minOccurs="0" maxOccurs="1"/>
   </sequence>
   <attribute name="Vendor"  type="string" use="optional"/>
</complexType>
```

SoftwareInfo

SoftwareInfo, contiene informazioni generali sul software usato dalla VNF, tra cui le seguenti specifiche.

- Developers, sviluppatori che hanno partecipato alla sviluppo;
- VersionInfo, permette dei definire l'attuale versione del software della VNF
- Repository, eventuale link al repository.
- ProgrammingLanguagesUsed, esprime il linguaggio con cui è stato sviluppato;
- OperatingSystemsSupported, eventuale compatibilità con particolari tipi di sistemi operativi;
- Licence, tipo di licenza del software.

In Appendice D sono stati analizzati nel dettaglio ciascuno punto elencato qui sopra.

Hardware Info

```
<complexType name="HardwareInfo">
   <sequence>
      <element name="CPU" type="CPU" minOccurs="0" maxOccurs="1"/>
      <element name="RAM" type="RAM" minOccurs="0" maxOccurs="1"/>
      <element name="Disk" type="Disk" minOccurs="0" maxOccurs="1"/>
      <element name="Bandwidth" type="Bandwidth" minOccurs="0" maxOccurs="
    1"/>
      <element name="Cost" type="integer" minOccurs="0" maxOccurs="1"/>
      <element name="Delay" type="Delay" minOccurs="0" maxOccurs="1"/>
   </sequence>
</complexType>
```

Informazioni di carattere generale utili per esprime specificare le risorse hardware utilizzate per eseguire il software delle VNFs, evidenziando per ciascuno componente (es. Cpu, Ram) le quantità di impiego.

- CPU, in numero di core usati dalla VNF
- RAM, la quantità di ram occuapata dalla VNF
- DISK, la quantità di spazio occupato sul disco.
- BANDWIDTH, la quantita di banda occuapata.
- COST, Costo della VNF.
- DELAY, ritardo della VNF.

E' possibile trovare ulteriori dettagli e informazioni sui campi utilizzabili per la definizione di una VNF in Appendice D.

ManifestInfo

In questa sezione troviamo informazioni rigurdanti fields e action delle VNFs. Parametri della VNF che è in grado di gestire. Particolare attenzione verrà data alla sezione fields. La struttura xml è formata prinipalamente da due elementi:

- FieldsManifest, l'insieme dei fields gestibili dalla VNF
- ActionsManifest, l'insieme delle azioni supportate dalla VNF

```
<element name="ManifestInfo" minOccurs="1" maxOccurs="1">
<complexType>
  <sequence>
    <element ref="fd:FieldsManifest" minOccurs="1" maxOccurs="1"/>
    <element ref="ac:ActionsManifest" minOccurs="1" maxOccurs="1"/>
  </sequence>
</complexType>
</element>
```

FieldsManifest

Nel struttura qui di seguito troviamo attributi significativi

- Type, indica il campo field che la VNF è in grado di gestire
- TypeConfig, in che modo la VNF gestisce il campo. Possiamo avere tre possibilità:
  - CONF_DEC, configurabile e decisionale, la vnf è in grado si saper leggere il significato del valore associato al campo e di saperne assegnare una corretta configurazione.
  - NOT_CONF_MAKE_DEC, non configurabile ma decisionale, la vnf è in grado si saper leggere il significato del valore asscoiato al campo ma non è in grado saperne applicare una configurazione, ovvero saperne assegnare un valore preciso al campo.
  - NOT_CONF_NOT_DEC, tutti i campi non presenti nell'elenco fanno parte di questo insieme.
- importSuite, permette di agevolare il supporto per una determinato campo attraverso l'insierimento di suite, ovvero un elenco predefinito di valori supportati per il singolo campo.

```
<element name="FieldsManifest">
<complexType>
  <sequence >
  <element name="Field" minOccurs="1" maxOccurs="unbounded">
    <complexType>
    <sequence>
      <element name="Supports" type="tns:SupportType" minOccurs="0"
   maxOccurs="1"/>
    </sequence>
    <attribute name="Type"  type="tns:enumField" use="required"/>
    <attribute name="TypeConfig" type="tns:configType" default="CONF_DEC"/
   >
    <attribute name="importSuite" type="string" use="optional"/>
    </complexType>
  </element>
  </sequence>
</complexType>
</element>
```

```
<simpleType name="configType">
   <restriction base="string">
      <enumeration value="CONF_DEC"/>
      <enumeration value="NOT_CONF_MAKE_DEC"/>
      <enumeration value="NOT_CONF_NOT_DEC"/>
   </restriction>
</simpleType>
```

ActionsManifest

Le azioni supportate dalle VNF vengono espresse attraverso questa struttura.

- Support, constraint sulle azioni supportate.
- importSuite, possibilità di andare ad inserire i constraint partendo da un insieme base di riferimento.

```
<element name="ActionsManifest">
<complexType>
   <sequence>
      <element name="ActionSupport" minOccurs="1" maxOccurs="unbounded">
         <complexType>
            <sequence>
               <element name="Support" type="string" minOccurs="0" maxOccurs="
    unbounded"></element>
            </sequence>
            <attribute name="Action" type="tns:typeAction" use="required">
            <attribute name="importSuite" type="string" use="optional">
         </complexType>
      </element>
   </sequence>
</complexType>
</element>
```

5.3 Functionality

La funzionalità è l'entita che esprime il modo in cui una VNF è ingrado si svolgere le operazioni richieste da un requisitio di sicurezza. Ricordiamo come l'entità funzionalità aggrega quelle che sono le informazioni in comune partendo dal mapping

tra requisito e la VNF presenti nel sistema. Queste informazioni sono espresse attraverso una struttura xml che le modella:

```
<element ref="tns:Fields" minOccurs="1" maxOccurs="1"/>
<element name="Actions" type="tns:Actions" minOccurs="1" maxOccurs="1"/>
<element name="SupportedByVNFS" minOccurs="1" maxOccurs="1">
 <complexType>
 <sequence>
  <element name="Vnf"  minOccurs="1" maxOccurs="unbounded">
  <complexType>
     <attribute name="Name" type="string" use="required"/>
  </complexType>
  </element>
 </sequence>
 </complexType>
</element>
```

E' formato principalmente dai seguenti elementi:

- Fields, l'insieme dei campi fields con il rispettivo valore associato ad essi
- Actions, l'insieme delle azioni che verrano svolte dalla funzionalità
- SupportedByVNFS, l'insieme delle VNF che durante il processo di mapping hanno come risultato gli stessi parametri di Fields e di Actions.

Questo tipo di schema verrà ripreso in seguito e verra completato con ulteriori parametri significativi.

5.4 Chain Functionality

L'output del framework consiste nell'andare a riportare liste di catene di funzionalità ovvero un insieme di funzionalità sufficiente per soddisfarte le richieste del requisito. La struttura seguente riutilizza il modello usato precedentemente per le funzionalità. L'ordine con cui queste funzionalità appaiono nella soluzione è l'ordine che è stato scelto dall'algoritmo.

```
<element name="ChainFunctionalities">
<complexType>
  <sequence>
    <element name="Chain" minOccurs="1" maxOccurs="unbounded">
    <complexType>
       <sequence>
          <element name="Solution" minOccurs="1" maxOccurs="unbounded">
             <complexType>
                <sequence>
                   <element name="Functionality" type="tns:functionExtended"
     minOccurs="0" maxOccurs="unbounded">
                   </element>
                </sequence>
                <attribute name="Sol_ID" type="string" use="required"/>
                <attribute name="Satisfiability" type="string" use="optional" default
     ="PARTIAL"></attribute>
             </complexType>
          </element>
       </sequence>
       <attribute name="Chain_ID" type="string" use="required"/>
    </complexType>
    </element>
  </sequence>
</complexType>
</element>
```

E' formato principalmente dai seguenti elementi:

- ChainID, identificativo di catena§;
- SolutionID, identificativo numerico per differenziare le diverse soluzioni per catena;
- Satisfability, un indetificativo di come questa ne è ingrado di fare l'enforce parziale o completa.

La struttura seguente evidenzia come sono state gestite le azioni per le singole funzionalità

```
<complexType name="Actions">
 <sequence>
  <element name="Action" minOccurs="1" maxOccurs="unbounded">
    <complexType>
      <sequence>
        <element name="InfoOnAction"    type="string" minOccurs="0"
   maxOccurs="unbounded"/>
      </sequence>
      <attribute name="Name_Action" type="string" use="required"/>
      <attribute name="Constraint_Action" type="string" use="optional"/>
      <attribute name="Priority_Action" type="integer" use="optional"/>
      <attribute name="Update_Value" type="string" use="optional"/>
    </complexType>
  </element>
 </sequence>
</complexType>
```

Particolare importanza hanno i seguenti attributi:

- Name_Action, nome dell'azione assegnata alla funzionalità;
- Constraint_Action, eventuale constraint da assegnare all'azione asseganta;
- Priority_Action, la priorità sull'azione indica a fronte di piu azioni assegnate quale deve essere svolta per prima. Utile soprattuto quando alla funzionalità vengono assegnate piu azioni da svolgere.

5.5 Abilitazione o Disabilitazione Tool

Questo struttura è stata costrutita per andare a gestire quelle situazioni in cui vogliamo modificare il comportamento standard del framework. Si ha la possibilità di abilitare o a disabilitare:

- *Neo4j*, per la visualizzazione grafica tramite browser delle catene di funzionalità individuate dall'algortimo;
- *MarshallOnFile*, la possibilità di andare a visualizzare solo a video le soluzione trovate senza andare a creare i rispettivi file;
- *OutputOnFile*, la possibilità di andare a produrre un output su file invece che un output sulla console.

```
<element name="Tools">
<complexType>
   <sequence>
   <element name="GraphNeo4j" >
      <complexType>
         <attribute name="Enable" type="boolean" default="false"/>
      </complexType>
   </element>
   <element name="MarshallOnFile" >
      <complexType>
         <attribute name="Enable" type="boolean" default="false"/>
      </complexType>
   </element>
   <element name="OutputOnFile" >
      <complexType>
         <attribute name="Enable" type="boolean" default="false"/>
      </complexType>
   </element>
   </sequence>
</complexType>
</element>
```

Il file è chiamato *tools.xml* ed è caricato all'avvio del framework. Lo schema di riferimento è reperibile al path *xsd/tools.xsd*. Una volta che il file è stato caricato dal framwerok, qualsiasi altra modifica al file non verrà presa in considerazione.

Capitolo 6

Selezione ottima delle funzionalità

Questo capitolo descrive come è stato possibile individuare le funzionalità in grado di garantire l'enforce del requisito, attraverso l'inserimento di constanti, vincoli e varibili binarie usate dal tool Gurobi incaricato di estrarre tutte le soluzioni valide.

6.1 Vincoli di processo

Nei capitolo precedenti abbiamo spigato quello che è il ruolo del modulo LER. Il compito principale è la ricerca di set di funzionalità ordinate in grado di fare l'enforce del requisito. Gurobi permette di fare tutto questo, andando semplicemente ad associare delle varibili binarie ai vari elementi, valore 1 (variabile binaria scelta) , 0 altrimenti.La scelta del valore associato alla varible binaria dipende dalle espressioni matematiche formulate e da eventuali constraint inseriti nel modello.

6.1.1 Prequisiti

Nel momento della definizione di questi vincoli si è partiti dall' risultato delle funzionalità. La presenza non numerosa vincoli è motivata in fase di costruzione delle funzionalità, dove si sono preventivamente escluse tutte quelle funzionalità che sicuramente non avrebbero partecipato attivamente nelle soluzioni accettabili. L'insieme uscente delle funzionalità è sicuramente un insieme di elementi che compare in almeno una soluzione. Queste decisioni vengono prese durante la fase di mapping. Alcuni esempi di esclusione di una funzionalità:

- Il requisito, richiede tre azioni concorrenti, la funzionalità f è in grado di supportarle ? No, non considero f per le azioni concorrenti.

- Il requsito richiede che un campo dipende da un azione, la funzionalità f supporta quell'azione ? No, non considero f per quel'azione dipendente.
- L'insieme dei campi in comune è vuoto, ? Si - la funzionalità f è priva di senso.
- L'insieme delle azioni in comune è vuoto ? Si - la funzionalità f è priva di senso.
- Il requisito impone un vincolo sull'azione, la funzionalità f supporta il vincolo ? No - non considero f per quel tipo di azione con il relativo vincolo.

C'è un altra esclusione che nasce da alcuni ragionamenti e situazioni al limite che possono presentarsi. Quando in un requisito viene richiesto l'azione di *deny*, bisogna effettuare un ulteriore considerazione, ovvero è necessario che tutte le funzionalità individuate supportano l'azione di *allow*. Il motivo è semplice, senza questo supporto per questa azione non sarebbe possibile effetture l'inoltro del traffico da una funzionalità ad un altra ma rimarrebbe confinato in una di essa. Per questo motivo è necessario che tutte le funzionalità che entreranno a far parte della soluzione supportino l'azione *allow*.

6.1.2 Vincoli nel modello

I vincoli imposti durante il processo di calcolo sono i seguenti:

1. Un singola azione del requisito puo essere associata ad una ed una sola funzionalità;
2. Tutte le azioni richieste dal requisito devono essere assciate ad almeno una funzionalità;
3. Se l'azione presenta un costraint anche quest'ultimo deve essere supportata dalla funzionalità
4. Tutte le azioni etichettate come concorrenti devono essere assegante tutte alla stessa funzionalità;
5. Le azioni concorrenti se assegnate devono essere supportate dalla medesima funzionalità
6. Le azioni dei requisiti devo essere assegnate in ordine alle funzionalità;
7. La stessa azione puo essere associata piu volte alla stessa funzionalità se e solo se agisce su campi parametrici differenti;
8. Alla stessa funzionalità possono essere associate piu azioni so se diverse diverse;

9. Una funzionalita che ha ricevuto gia l'asseganzione dell'azione puo essere nuovamente cadidata all'asseganzione di un ulteriore azione purchè diversa dalla precedente.

Nomenclatura usata per esprimere i vincoli citati sopra :

- A1:F1, l'azione A1 del requisito è assegnata alla funzionalità F1;
- A1:P1:F1, l'azione A1 del requisito svolto sul campo parametrico P1 è associato alla funzionalità F1;
- A1:C1:P1:F1, l'azione A1 del requisito con rispettivo vincolo C1 sull'azione viene svolto sul campo parametrico P1 è assegnato alla funzionalità F1;
- AC1:F1 l'azione concorrente AC1 è associata alla funzionalità F1.

6.2 Formulazione della notazioni Gurobi

Questa sezione descrive le varibili binarie e le formule matematiche usate da Gurobi per ottimizzare la selezione delle funzionalità ottenute dal modulo RDF. Questi step vengono svolti all'interno del modulo LER (Logic Enforce Requirement).

6.3 Gurobi formulation problem

In questa sezione verranno mostrate tutte le formule e le notazioni usate da gurobi per il modello LER. Queste formulazioni permetto la selezione ottima di funzionalità.

6.3.1 Simboli

Qui di seguito vengono riportati i simboli utilizzati per rappresentare i vari insieme dei dati su cui vengono forumlate equazioni matematiche [6.3.2] e su i quali vengono applicati vincoli come restrizioni sugli insiemi o dipendenze da altre varibili.

- a_i : i-esima azione $a \in A_r$
- F : l'insieme delle funzionalità
- f_j : la j-esima funzionalità $f \in F$

- a_i^j : azione i-esima della funzionalità j-esima
- x_i^j : varibile binaria a_i^j , 1 se è assegnata, 0 altrimenti
- C : l'insieme delle azioni concorrenti imposte dal requisito
- c_s : s-esima azione concocorrente $c \in C$
- c_s^j : s-esima azione concorrente svolta dalla j-esima funzionalità
- r_s^j : varibile binaria associata a c_s^j, 1 se assegnata, 0 altrimenti
- Z : insieme dei field parametrici
- z_h : h-esimo campo parametrico $z \in Z$
- $a_i^{h,j}$: azione i-esima eseguita sul campo h-esimo, svolta dalla funzionalità j-esima
- $x_i^{h,j}$: variabile binaria associata a $a_i^{h,j}$, 1 se è assegnato, 0 altrimenti.

6.3.2 Vincoli

Queste notazioni rappresentano i vincoli che verranno testati da Gurobi ad ogni soluzione identificata. I vincoli fanno si di considerare solo soluzioni utili e che abbiano in comune la capacita di *enfocability* di un requisito. I vincoli forniti qui di seguito lavorano sulle varibili binarie definite prima.[6.3.1]. In caso di mancanza di soluzioni valide il tool fornirà un segnale noto di UNSAT (Unsatisfiability - Non Soddisfacibilità).

1.
$$\sum_{i|j=1}^{m} x_i^j = 1 \tag{6.1}$$
$$\forall i \in A, \forall j \in F \tag{6.2}$$

Un azione puo essere associata ad una ed una sola j-esima funzionalità. La stessa funzionalita puo essere associata a piu azioni.

2.
$$\sum_{i=1}^{n} \left(\sum_{j=1}^{m} x_i^j = 1\right) = A \tag{6.3}$$
$$\forall i \in A, \forall j \in F \tag{6.4}$$

La somma di tutte le azioni associate alle singole funzionalità deve essere apri alla dimensione dell'insime delle azioni del requisiti (A).

3.
$$r_k^j(k-1) \bigwedge r_n^j(n-1) \tag{6.5}$$
$$\forall (n,k)|j \ , \ k \neq n, \forall j \in F \tag{6.6}$$
$$1 < k \leq C, 1 < n \leq C \tag{6.7}$$

Se ad una funzionalità viene asseganta un azione concorrente , allora necessariamente deve essere asseganta anche le altre dell'insieme.

4.
$$\sum_{(i,j)|h=1}^{h=n} x_i^{h,j} \leq Z \tag{6.8}$$
$$\forall i \in A, \forall j \in F, \forall h \in Z \tag{6.9}$$

La funzionalità può svolgere la stessa azione ma su campi parametrici diversi. Il massimo valore è pari alla dimensione dell'insieme dei campi parametrici Z.

5.

$$1 \leq \sum_{(i,j)|h=1}^{h=n} x_i^{h,j} \tag{6.10}$$

$$\forall i \in A, \forall j \in F, \forall h \in Z \tag{6.11}$$

Limite inferiore sul numero sulla dimensione dell'insieme da poter considerare.

Alcuni vincoli descritti in precedenza nel sezione 6.1.2 non appaiono nella lista delle fomulazioni dei vincoli poichè alcuni di questi come il punto 6 che recita: *"Le azioni dei requisiti devo essere assegnate in ordine alle funzionalità"* viene soddisfatta nella fase di costruzione del modello. Gurobi assegna ad ogni variabile binaria/continua inserita nel modello, un valore numerico intero che ne rappresenta l'ordinamento. Questo valore viene sfruttato per indicare l'ordinamento delle azioni. Il valore può essere modifcato in seguito se necesssario. In presenza di un requisito con azioni non ordinate, questa situazione non è un problema dato che verrano considerate tutte le sue permutazioni. In presenza invece di un insieme parzialmente ordinato di azioni, alle azioni non ordinate li vengono assegnate un valore di ordinamento negativo (es. -1, -2, -3), per azioni ordinate un valore di ordinamento positivo (+1 , +2, +3). Per riassumere tutti casi :

- A1 + A2 + A3 , insieme ordinato di azioni;
- A-2 + A-1 + A1 + A2 , insieme parzialmente ordinato;
- A-3 + A-2 + A-1 , insieme non ordinato di azioni.

Un altro vincolo non presente nella formulazione è il punto 3, che recita *"Se l'azione presenta un costraint anche quest'ultimo deve essere supportata dalla funzionalità"*. Questo vincolo viene testato in fase di assegnazione. Se la funzionalità non supporta il constraint associato all'azione, essa non viene considerata. Infine un altro vincolo non presente nella formulazione è il punto 5, che recita *"Le azioni concorrenti se assegnate devono essere supportate dalla medesima funzionalità"*. Questo vincolo viene testato in fase di assegnazione. Se la funzionalità non supporta tutte le azioni conconcorrenti allora non viene effettuata nessuna assegnazione ad essa.

6.4 Operazioni di post Processing

Il modello visitato poco prima ci permette di avere subito una visione netta di quelle che sono le catene ma nonostante ciò, necessitano di un ulteriore elaborazione dei dati. Si tratta di un processo di aggregazione chiamato *Reduction process*. Consiste nell'aggregare tutte quelle repliche della stessa funzionalità se e solo se quest'ultime sono adiacenti.

Una serie di possibili catene individuate si presentano in questa forma:

1. $A1 : F1 + A2 : F2 + A3 : F4$
2. $AC1 : F2 + AC2 : F2 + A3 : F4$
3. $A1P1 : F1 + A2P2 : F1 + A3 : F1$
4. $A1 : F2 + A2 : F3 + A3 : F2$

Quelle enumerate qui sopra sono alcune delle situazioni in cui potremmo trovarci. Alcune di queste non necessitano di ulteriori manipolazioni, altri invece si .Analizziamoli caso per caso:

Caso 1

Il caso 1, è il caso piu fortunato ovvero non necessita di alcuna modifica. E' evidente che in questo caso le azioni del requisito sono assegante a diverse funzionalità e **non** è necessario eseguire il processo di riduzione in quanto non vi sono le condizioni minime per applicarlo

Caso 2

Il caso 2, è il caso in cui nel requisito sono presenti azioni concorrenti. Sappiamo che queste azioni devo essere svolte dalla singola funzionalità . Effettivamente lo è. In questo caso nella notazione riportata appare la replica di un istanza di una funzionalità F2, in cui, nella prima istanza applica l'azione AC1 e in un seconda instanza un AC2. Queste possono essere viste come una singola istanza in cui vengono applicate entrambe dalla funzionalità F2. Il risultato di questa operazione produce il seguente output: $(AC1 + AC2) : F2 + A3 : F4$

Caso 3

Il caso 3, è il caso in cui nel requisito sono presenti azioni che agiscono su campi parametrici. Notiamo la presenza di 3 instanze della funzionalità F1 in cui in ciascuna istanza di funzionalità vengono svolte azioni diverse. Queste possono essere viste come una singola istanza in cui vengono applicate tutte le azioni (A1P1

e A1P2 e A3) dalla funzionalità F1. Il risultato di questa operazione produce il seguente output: $(A1P1 + A2P2 + A3) : F1$

Caso 4

Il caso 4, è il caso in cui il requisito è soddifatto solo con l'uso di due funzionalità. Quello che è importante notare in questo caso è la replica di un istanza di una stessa funzionalità in cui li vengono assegnate azioni differenti. In una prima istanza di F2 è asseganta l'azione A1 e in una seconda istanza l'azione A3. Non viene svolto nessun processo di aggregazione perchè non è presente la concatenazione contigua della stessa funzionalità F2.

Per avere un quadro completo del nuovo output in seguito all'applicazione del processo di riduzione alle catene di funzionalità presentate prima, vengono riportate qui sotto con la nuova notazione:

1. $A1 : F1 + A2 : F2 + A3 : F4$
2. $(AC1 + AC2) : F2 + A3 : F4$
3. $(A1P1 + A2P2 + A3) : F1$
4. $A1 : F2 + A2 : F3 + A3 : F2$

Capitolo 7

Validazione

Per valutare la scalabilità e il tempo di esecuzione del tool , sono stati eseguiti una serie di test. I test sono stati eseguiti sul flusso completo di esecuzione. Sono stati eseguti anche i test sui singoli moduli per osservare come questi reagiscono ai vari input che è in grado di accettare il tool. Le prove sono state eseguite su una macchina avente le seguenti caratteristiche:

- Sistema operativo: Windows 10 Pro;
- CPU: CPU Intel® Core™ i7-6700MQ a 3,40 GHz;
- Architettura: x64;
- RAM: 32,00 GB Dual-Channel DDR4.

I test avvengono aumentando periodicamente uno delle varibili alla volta in maniera progressiva al fine di monitorare il tempo di esecuzione. I test hanno dimostrato che il tempo di esecuzione è più sensibile al numero dei requisiti rispetto al numero di VNFs, ma non solo, parte di queste tempistiche dipendono da alcuni fattori intreseci per i quali non è possibile controllare il suo comportamento in quanto non sono parametri di input del tool.

7.1 Test di Scalabilità

Sul workflow completo sono stati eseguiti una serie di test per testarne la scalabilità. I seguenti test dipendo da una serie di fattori tra cui:

- Numero di requisiti;
- Numero di VNFs presenti nel catalogo;
- Numero di thread usati;

- Numero delle funzionalità ottenute ;
- Presenza di azioni ordinate nel requisito;

Tra tutti i fattori evidenziati prima ,solo tre sono sotto il controllo dell' utente, ovvero il numero di requisiti, il numero di vnfs e il numero di thread. I restanti sono parametri intriseci che dipendo dal workflow.

7.1.1 VNFs=constanti, Requisiti=crescenti, Threads=8

L'impatto del numero dei requisiti è mostrato nella Fig. 7.1. Questo grafico è stato ottenuto variando il numero di requisiti in input al modello e mantenendo constante il numero di VNFs uguale a 10, e il numero di thread a 8. È importante notare che sono stati eseguiti dei test incrementando le copie degli stessi requisiti, questo permette di valutare al meglio le prestazioni del tool in quanto lo strumento è costretto a lavorare nel situazione peggiore. Esaminando il grafico 7.1 che rappresenta un ingrandimento della primi intervalli di tempo, possiamo distinguere le seguenti situazioni:

- Tra 0 e 100 requisiti, il tempo di esecuzione è inferiore a 1 s;
- Tra 100 e 400 requisiti, il tempo di esecuzione è massimo di 2 s ;
- Tra 400 e 1.000 requisiti, il tempo di esecuzione è inferiore a 10 s;
- Tra 1.000 e 6.000 requisiti, il tempo di esecuzione è inferiore a 25 s;
- Sino 100.000 requisiti, il tempo di esecuzione è inferiore a 2.5 ore.
- Dopo 100.000 requisiti, il tempo di esecuzione cresce sempre più velocemente. Durante la fase finale del test, sono state analizzate oltre un milione di requisiti in quasi 5 ore.

7.1.2 VNFs=constanti, Requisiti=crescenti, Threads=8

Quello che lo differenzia il grafico 7.2 dal test precedente è la presenza di azioni non ordinate in tutti i requisiti. Anche questo grafico è stato ottenuto variando il numero di requisiti in input al modello e mantenendo constante il numero di VNFs uguale a 10, e il numero di thread a 8. È importante notare che sono stati eseguiti dei test incrementando le copie degli stessi requisiti, questo permette di valutare al meglio le prestazioni del tool in quanto lo strumento è costretto a lavorare nel situazione peggiore. Esaminando il grafico 7.2 che rappresenta un ingrandimento della primi intervalli di tempo, possiamo distinguere le seguenti situazioni:

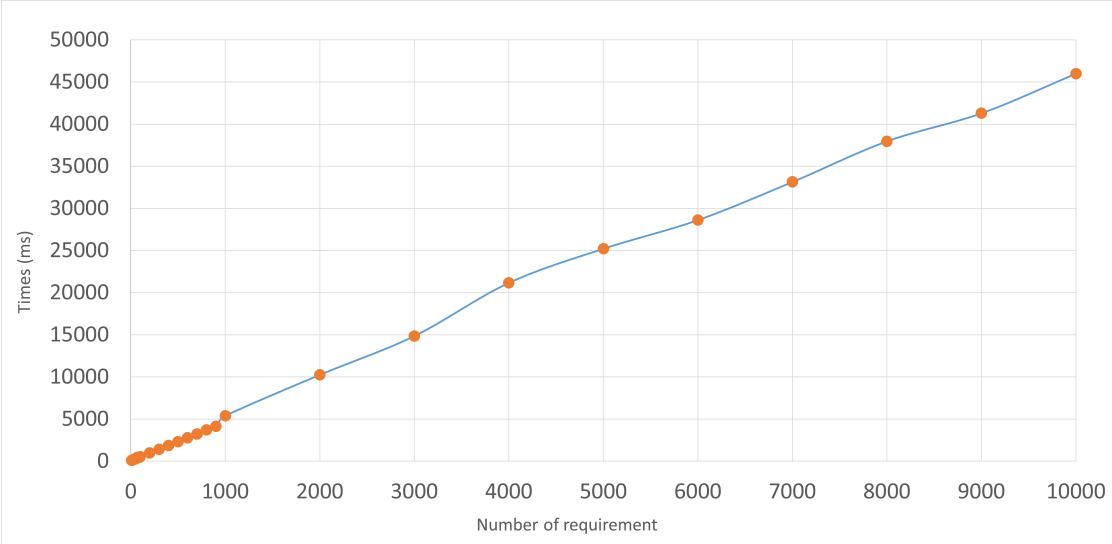

Figura 7.1: Andamento nel tempo del tempo impiegato per analizzare tutti i requisiti

- Tra 0 e 50 requisiti, il tempo di esecuzione è inferiore a 1 s;
- Tra 50 e 100 requisiti, il tempo di esecuzione è massimo di 2 s ;
- Tra 100 e 800 requisiti, il tempo di esecuzione è inferiore a 10 s;
- Tra 800 e 2000 requisiti, il tempo di esecuzione è inferiore a 30s;
- Sino 7.000 requisiti, il tempo di esecuzione è inferiore a 1 m;
- Sino 10.000 requisiti, il tempo di esecuzione è inferiore a 3 m;
- Dopo 100.000 requisiti, il tempo di esecuzione cresce sempre più velocemente. Durante la fase finale del test, sono state analizzate oltre un milione di requisiti in oltre 10 ore.

7.1.3 Confronto dei risultati effettuati sui requisiti

Quello che differenzia il grafico 7.2 dal grafico 7.1 è uno dei parametri intrinseci dei requisiti. Il confronto è rappresentato nel grafico 7.3, dove si da subito si nota un certo scostamento della linea arancione (tempo di esecuzioni di requisiti con azioni ordinati) da quella blu (tempo di esecuzione di requisiti con azioni non ordinate). Un risultato che sicuramente ci aspettavamo, questo perchè ne introduce un ulteriore complessità per ciascuna soluzione trovata dal tool. L'andamento lineare è sicuramente atteso. Se provassimo a calcolare la retta di tendenza otteremo:

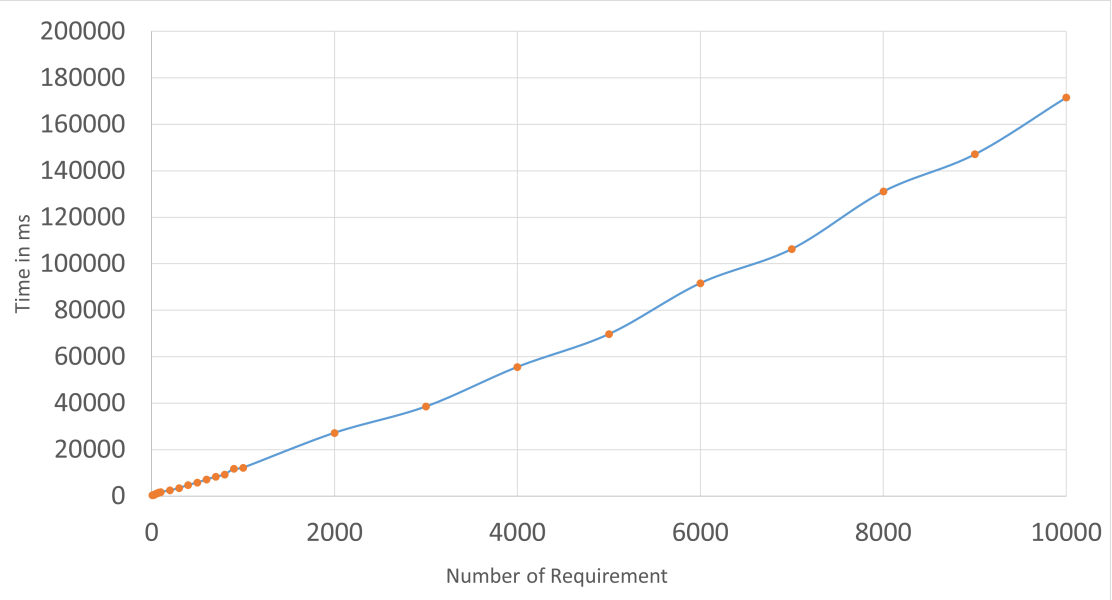

Figura 7.2: Andamento nel tempo del tempo impiegato per analizzare tutti i requisiti

- linea blu : $y = 16{,}236x - 2120{,}7$
- linea arancio: $y = 4{,}6883x + 268{,}26$

Calcolando il rapporto dei due coefficenti angolari della rette di queste due funzioni otteremo un valore pari a circa tre. Questo valore rappresenta la differenza di pendenza delle due rette. Possiamo dire in maniera elementare che la linea blu è 3 volte piu lenta (quindi impiega meno tempo) rispetto alla linea arancione. Questo è un evidente dimostrazione di come dei requisiti con azioni non ordinate influiscono nei tempi di esecuzioni di un fattore significativo.

7.1.4 VNFs=crescenti, Requisiti = costanti, Threads=3-5-8-10

L'impatto del numero di VNFs è mostrato nella Fig. 7.4 . Questo grafico è stato ottenuto variando il numero di VNFs e mantenendo il numero di requisiti costanti uguale a 10. Il numero di thread varia da tre, poi variando a cinque, a otto e infine a dieci. È importante notare che sono stati eseguiti dei test incrementando le copie della stesse VNFs, come è accaduto nel test precedente. Esaminando il grafico in Figura 7.4 che rappresenta un ingrandimento della primi intervalli di tempo, possiamo distinguere le seguenti situazioni:

- Tra 0 e 1000 VNFs, il tempo di esecuzione è inferiore a 1 s;
- Tra 1000 e 4.000 VNFs, il tempo di esecuzione è massimo di 2 s;
- Tra 4.000 e 7.000 VNFs, il tempo di esecuzione è massimo di 10 s;
- Tra 4.000 e 10.000 VNFs, il tempo di esecuzione si mantiene costante intorno ai 6 s;
- Dopo 100.000 funzioni, il tempo di esecuzione cresce sempre più velocemente. Durante la fase finale del test, sono state analizzate oltre 1 milione di VNFs in poco piu 4 ore;

Si puo notare come l'aumento del numero di threads non implica necessariamente un diminuzione delle tempistiche, addirittura otteniamo prestazione peggiori. Guardando l'andamento del grafico si intuisce anche in questo caso un andamento lineare nel tempo, e se tentassimo di calcolare anche in questo caso le linea di tendenza per ciascun caso otteremo le seguenti funzioni:

- Linea t10 - $y = 0{,}5237x + 146{,}11$
- Linea t8 - $y = 0{,}5203x + 55{,}691$
- Linea t5 - $y = 0{,}6192x + 77{,}637$
- Linea t3 - $y = 0{,}713x + 70{,}486$

E' possibile osservare come partendo da queste funzione e esaminando i vari coefficenti angolari della retta è facilmente dimostrabile come la linea in giallo abbia mediamente tempi di esecuzioni migliori(coefficente pari a 0.5203). Da questa analisi nasce il valore di threads pari a 8 usato nei precedenti test, frutto di una analisi nel tempo delle performance. Esso è un parametro che dipende molto dall' architettura hardware sui cui sono stati eseguiti i test di scalabilità.

7.1.5 VNFs=crescenti, Requisiti = crescenti, Threads =8

Questo grafico è stato ottenuto variando il numero di funzioni e il numero di requisiti in maniera proporzionale. Importatnte notare l'andamento sempre crescente del tempo di esecuzione. Esaminando il grafico 7.5 che rappresenta un ingrandimento della primi intervalli di tempo, possiamo distinguere le seguenti situazioni:

- Tra 0 e 200 (Requisiti e VNFs, il tempo di esecuzione è inferiore a 1 s;
- Tra 200 e 400 (Requisiti e VNFs), il tempo di esecuzione è massimo di 2.5 s;
- Tra 400 e 600 (Requisiti e VNFs), il tempo di esecuzione è massimo di 7 s;

- Tra 600 e 5000 (Requisiti e VNFs), il tempo di esecuzione si mantiene costante intorno ai 1 m;
- Dopo 100.000 (Requisiti e VNFs) il tempo di esecuzione cresce sempre più velocemente (2.5 ore). Nella fase finale sono state analizzate oltre 1 milioni di dati(requisiti e VNFs) stimando come tempi di esecuzione del'ordine dei giorni.

7.1.6 VNFs=crescenti, Requisiti = crescenti, Threads =8

Questo grafico è stato ottenuto variando il numero di funzioni e il numero di requisiti in maniera proporzionale. Quello che distingue questo test da quello precedente è il modo in cui il requisito impone gli ordinamenti delle azioni. Notiamo anche in questo caso un andamento crescente dei tempi di esecuzione. Esaminando il grafico 7.6 possiamo distinguere le seguenti situazioni:

- Tra 0 e 200 (Requisiti e VNFs), il tempo di esecuzione è inferiore a 1 s;
- Tra 200 e 400 (Requisiti e VNFs), il tempo di esecuzione è massimo di 2.5 s;
- Tra 400 e 600 (Requisiti e VNFs) funzioni, il tempo di esecuzione è massimo di 7 s;
- Tra 600 e 5000 (Requisiti e VNFs), il tempo di esecuzione è inferiore ai 3 m;;
- Dopo 100.000 (Requisiti e VNFs), il tempo di esecuzione cresce sempre più velocemente (circa 5 ore) . Infine sono state analizzate oltre due milioni di dati tra requisiti e VNFs stimando come tempi di esecuzione dell'ordine dei giorni.

7.1.7 Confronto fra test progressivi

In Figura 7.7 appare chiaramente come la scelta di un ordinamento delle azioni incide sulle tempistiche di esecuzione anche nei test progressivi. Con un numero di dati relativamente piccoli la differenza dei tempi di esecuzione è minima o in alcuni casi quasi impercettibile. Ma è abbastanza semplice immaginare come questo fattore puo incidere sulle tempistiche di esecuzione sopratutto su grosse dimensioni di dati. Per avere un idea di come variano nel tempo le due casisitiche è stato calcolato la linea di tendenza per i due test:

- Linea blue - $y = 0{,}0016x2 + 8{,}0891x - 230{,}21$ - andamento del tempo di esecuzione per i requisiti con azioni non ordinate.

- Linea arancio - $y = 0{,}0009x2 + 5{,}008x + 63{,}402$ - andamento del tempo di esecuzione per i requisiti con azioni ordinate.

Si nota si da subito un divario tra i due coefficenti angolari delle rette di grado due. Questo dimostrano come la linea arancio ha un tempo di esecuzione minore del 56% rispetto alla linea blu. Questa è una chiara dimostrazione di come i tempi di esecuzioni possano aumentare rapidamente. L'andamento quadratico è piu che inevitabile in quanto diversi requisiti mappano diversamente la stessa VNFs producendo diverse funzionalità. Il processo di rivisita delle stesse VNFs per ogni requisito è inevitabile

7.2 Casi d'uso

In quest'ultima parte andremo ad osservare alcuni casi d'uso di requisiti, andando ad evidenziare come il tool è in grado di soddisfare gran parte delle richieste dell'utente. Si partirà con un esempio molto semplice fino ad raggiungerne uno più complesso. Verrano mostrati esempi di requisiti in formato XML, e verranno discussi i dettagli piu significativi inseriti.

7.2.1 Caso semplice

Supponiamo di essere all'interno di una società e l'amministratore di rete vuole evitare che durante la Domenica (solo in una determinata fascia oraria) i servizi presenti sulle porte comprese tra (1024-10500) non siano raggiungibili. Supponiamo di restrignere il campo ad una determinata rete 10.0.3.21/24 e che gli orari sono compresi da mezzanotte a mezzogiorno.

Il requisito viene tradotto nel seguente modo:

```
<Requirement Req_ID="1">
<Info>
   <Description>Questa è una descrizione REQ1 − </Description>
   <Administrator>MARIO ROSSI</Administrator>
</Info>
<Fields>
   <IpSrc Version="ipv4" Address="10.0.3.21/24"/>
   <PortDst Range="1024−10500" RangeSetting="inside"/>
   <Proto TypeProtocol="∗"/>
   <DaysAndTimes TimeZone="TMZ">
    <DayAndTime Day="Sunday" TimeStart="00:00:00" TimeEnd="12:00:00"/>
   </DaysAndTimes>
</Fields>
<Actions>
   <ActionSet Order="1">
      <ActionRequired Action="LOG_TRAFFIC"/>
   </ActionSet >
   <ActionSet Order="2">
      <ActionRequired Action="DENY_TRAFFIC"/>
   </ActionSet>
</Actions>
</Requirement>
```

Tempo medio di esecuzione <45 ms.

7.2.2 Caso medio

Un caso meno semplice ma non complesso, puo essere riassunto nel seguente modo: L'amministratore di rete intende eseguire le seguenti azioni:

- Memorizzare tutti i pacchetti che fanno match con i valori sui fields. In futuro l'utente vuole effettuarne delle statistiche su di esso,(esempio monitorare in quale fascia orario si ha più match, o addirittura tenere traccia degli ip di destinazioni raggiunginti ecc..) .
- Contarli ed emettere un alert di notifca,
- Loggare il traffico , in particolare si vuole tenere traccia solo dei valori di ip destinazione e porta.

Il modo con cui le azioni devono essere svolte non è casuale ma segue un ordine ben preciso, inoltre l'utente ha richiesto esplicitamente che le operazioni di

alert e count siano fatte dalla stessa funzionalità. Questa tipo di comportamento viene inserito aggiungendo l'attributo *Concurrent=true*, ovvero si forza a cercare funzionalità che sappiano svolgere entrambe le azioni. L'utente più esperto ha la possibilità di esprimere particolari match attravero l'uso di espressioni regolari su alcuni campi come per il tipo di documento nel nostro caso. E' importante notare la notazione '*' utile per indicare che su quel campo non è imposto nessun vincolo, qualsiasi valore è accettato.

```
<Requirement Req_ID="2">
...
<fd:Fields>
   <fd:IpSrc Version="ipv4" Address="10.0.3.21/24"/>
   <fd:IpDst Version="ipv4" Address="*"/>
   <fd:PortDst Range="1024-10500" RangeSetting="inside"/>
   <fd:Proto TypeProtocol="TCP"/>
   <fd:Domains>
      <fd:Domain DomainValue="*"/>
   </fd:Domains>
   <fd:DaysAndTimes TimeZone="TMZ">
      <fd:DayAndTime Day="Sunday" TimeStart="00:10:33" TimeEnd="12:15:22"/>
   </fd:DaysAndTimes>
   <fd:Documents>
      <fd:Document Type="PDF" Regex="myregex"/>
   </fd:Documents>
</fd:Fields>
<Actions>
   <ActionSet Order="1">
      <ActionRequired Action="LOG_TRAFFIC">
         <FieldsParametric>
            <Field Ref="IpDstVer4"/>
            <Field Ref="PortDst"/>
         </FieldsParametric>
      </ActionRequired>
   </ActionSet >
   <ActionSet Order="2" Concurrent="true">
      <ActionRequired Action="ALERT"/>
      <ActionRequired Action="COUNT"/>
   </ActionSet>
   <ActionSet Order="3">
      <ActionRequired Action="STORE_PACKET"/>
   </ActionSet>
</Actions>
</Requirement>
```

Tempo medio impiegato <180 ms.

7.2.3 Caso complesso

Questo particolare requisito formulato in questo modo, vuole dimostrare come il tool è robusto a richiste molto complesse ed è in grado di gestire in tempi ragionevoli situazioni di questo calibro. Situazioni non frequenti ma possibili. E' importate sottolineare come le azioni in questo caso non hanno un ordinamento preciso. Questo influisce come già spiegato in precedenza sui tempi di esecuzione. Il tempo di esecuzione medio è inferiore ai 800 ms. Se considerassimo tutte queste azioni ordinate e supponiamo che l'ordinamento proposto qui sotto è quello che desidera l'utente, il tempo medio di esecuzione scende sotto agli 80 ms. Diminuzione di un fattore 10. Tempo che su un numero di dati molto grande (per esempio un milione) porterebbe a tempi di esecuzione dell'ordine dei giorni. Questo vuole essere una testimonianza di quanti fattori possono andare ad influire sul tempo medio di esecuzione. Per comodità andremo ad osservare prima l'insieme delle condizioni da verificare su i campi e in seguito le azioni coinvolte.

```
<Requirement Req_ID="8">
<Info>
    <Description>Questa è una descrizione REQ8</Description>
    <Administrator>MARIO ROSSI</Administrator>
</Info>
<fd:Fields>
    <fd:IpSrc Version="ipv6" Address="2001:db8:3333:4444:5555:6666:7777:8888"/>
    <fd:IpDst Version="ipv6" Address="2001:db8:3333:4444:5555:6666:7777:9999"/>
    <fd:PortSrc Range="15000-35000" RangeSetting="inside"/>
    <fd:PortDst Range="1-10500"     RangeSetting="outside"/>
    <fd:Proto TypeProtocol="*"/>
    <fd:Domains>
        <fd:Domain DomainValue="www.sitedomain.com"/>
    </fd:Domains>
    <fd:VlanTag>tagTraffic</fd:VlanTag>
    <fd:Applications>
        <fd:Application Type="MESSENGER"/>
    </fd:Applications>
    <fd:Body>Specific content inside the body</fd:Body>
    <fd:GeoPosition Country="Italia"/>
</fd:Fields>
```

```
<Actions>
   <ActionSet>
      <ActionRequired Action="ENCRYPT_TRAFFIC"
      Constraint="AES-128"/>
   </ActionSet>
   <ActionSet>
      <ActionRequired Action="FORWARD_TRAFFIC"
      Constraint="IpDstVer4" UpdateValue="10.2.2.3/24"/>
   </ActionSet>
   <ActionSet>
      <ActionRequired Action="REMARK_PACKET"
      Constraint="VlanTag" UpdateValue="NewTagForTraffic"/>
   </ActionSet>
   <ActionSet>
      <ActionRequired Action="LIMIT_BANDWIDTH"
      Constraint="10Mbit/s"/>
   </ActionSet>
   <ActionSet Concurrent="true">
      <ActionRequired Action="LOG_TRAFFIC">
         <FieldsParametric>
            <Field Ref="IpDstVer6"/>
            <Field Ref="PortDst"/>
         </FieldsParametric>
      </ActionRequired>
      <ActionRequired Action="ALERT"/>
      <ActionRequired Action="COUNT"/>
   </ActionSet>
   <ActionSet>
      <ActionRequired Action="ANTIVIRUS"/>
   </ActionSet>
   <ActionSet>
      <ActionRequired Action="STORE_PACKET"/>
   </ActionSet>
</Actions>
</Requirement>
```

Questo esempio si puo tranquillamente complicare a piacere aggiungendo valori sui campi, aggiungendo vincoli sulle azioni, imponendo un eventuale ordinamento parziale delle azioni. Aggiungendo ulteriori specifiche, il tempo medio di esecuzione continua a crescere sempre di più.

7.3 Soluzioni di un caso d'uso

Concluderemo questo capitolo con la discussione di alcune soluzione di catene di funzionalità individuate dal tool. Faremo riferimento al caso più semplici spiegato precedentemente per ragioni di complessità e per il numero di possibili soluzioni. Verranno presentate due casistiche:

1. Soluzione etichettata come parziale.
2. Soluzione etichettata come completa.

Notare la differenza tra le due soluzione proposte andando ad osservare come sono configurati i campi e quale è l'azione coinvolta, e sopratutto quali sono le VNFs che supportano la stessa configurazione. Tutte le soluzioni riguardanti un determinato requisito vengono salvate tutte su un file xml.

A)

Nella soluzione con id **SOL_0** notiamo la presenza di due funzionalita concatenate, la prima ha una configurazione diversa dalla seconda sia in intermini di campi che di azioni. In questo specifico caso si hanno le due azioni che vengono eseguite da due VNFs diverse. La prima funzionalità svolge l'azione di "LOG" la seconda esegue l'azione di "DENY".
Caso diverso è per la soluzione con id **SOL_1**, in cui si è riusciti a trovare una funzionalità capace di fare l'enforce del requisito senza un ulteriore funzionalità da affiancare. La presenza del campo *Priority_Action* sull'azione coinvolta permette alla VNFs di imporre un ordinamento delle azioni utile nel momento in cui quella stessa funzionalità (come in questo esempio) deve svolgere piu azioni sul traffico. Se volessimo elencare le VNFs utili per l'enforce del requisito basterà andare a leggere le VNFs presenti nel tag *SupportedByVNFS*. Possiamo elencarle qui di seguito:

1. VNF6 +VNF9
2. VNF6 +VNF11
3. VNF10 +VNF9
4. VNF10 +VNF11
5. VNF2

```
<Solution Sol_ID="SOL_0" Satisfiability="PARTIAL">
<Functionality>
    <Fields>
    <IpSrc Version="IPV_4" Address="10.0.3.21/24" ConfigField="CONF_DEC"/>
    <PortDst Range="1024-10500" RangeSetting="INSIDE" ConfigField="
     CONF_DEC"/>
    <Proto TypeProtocol="*" ConfigField="CONF_DEC"/>
    </Fields>
    <Actions>
        <Action Name_Action="LOG_TRAFFIC" Priority_Action="1"/>
    </Actions>
    <SupportedByVNFS>
        <Vnf Name="VNF6"/>
        <Vnf Name="VNF10"/>
    </SupportedByVNFS>
</Functionality>
<Functionality>
    <Fields>
    <IpSrc Version="IPV_4" Address="10.0.3.21/24" ConfigField="CONF_DEC"/>
    <Proto TypeProtocol="*" ConfigField="CONF_DEC"/>
    </Fields>
    <Actions>
        <Action Name_Action="DENY_TRAFFIC" Priority_Action="1"/>
    </Actions>
    <SupportedByVNFS>
        <Vnf Name="VNF9"/>
        <Vnf Name="VNF11"/>
    </SupportedByVNFS>
</Functionality>
</Solution>
```

```
<Solution Sol_ID="SOL_1" Satisfiability="PARTIAL">
<Functionality>
   <Fields>
   <PortDst Range="1024-10500" RangeSetting="INSIDE" ConfigField="
    CONF_DEC"/>
   <Proto TypeProtocol="*" ConfigField="CONF_DEC"/>
   <DaysAndTimes TimeZone="TMZ" ConfigField="CONF_DEC">
      <DayAndTime Day="Sunday" TimeStart="00:10:33" TimeEnd="12:15:22"/>
   </DaysAndTimes>
   </Fields>
   <Actions>
      <Action Name_Action="LOG_TRAFFIC" Priority_Action="1"/>
      <Action Name_Action="DENY_TRAFFIC" Priority_Action="2"/>
   </Actions>
   <SupportedByVNFS>
      <Vnf Name="VNF2"/>
   </SupportedByVNFS>
</Functionality>
</Solution>
```

B)

Questa soluzione è diversa dalle precedenti in termini soddisfacibilità, rileviamo sin da subito l'attributo *Satisfiability* pari a *COMPLETE*.
Ricordiamo che per soddisfacibilità completa, si intende aderenza completa con il requisito sia in termini di campi supportati(ogni campo è supportato completamente) e sia in termini di azioni (ciascuna azione è supportata con gli eventuali vincoli) .
In questo caso le funzionalità concatenate permettono di avere un'aderenza completa con il requisito. Per scegliere questa soluzione basterà prendere la **VNF8** seguita da **VNF2** e assegnarle i seguenti parametri.

```
<Solution Sol_ID="SOL_2" Satisfiability="COMPLETE">
<Functionality>
   <Fields>
   <IpSrc Version="IPV_4" Address="10.0.3.21/24" ConfigField="CONF_DEC"/>
   <Proto TypeProtocol="*" ConfigField="CONF_DEC"/>
   </Fields>
   <Actions>
      <Action Name_Action="LOG_TRAFFIC" Priority_Action="1"/>
   </Actions>
   <SupportedByVNFS>
      <Vnf Name="VNF8"/>
   </SupportedByVNFS>
</Functionality>
<Functionality>
   <Fields>
   <PortDst Range="1024-10500" RangeSetting="INSIDE" ConfigField="
    CONF_DEC">
   <Proto TypeProtocol="*" ConfigField="CONF_DEC"/>
   <DaysAndTimes TimeZone="TMZ" ConfigField="CONF_DEC">
      <DayAndTime Day="Sunday" TimeStart="00:10:33" TimeEnd="12:15:22"/>
   </DaysAndTimes>
   </Fields>
   <Actions>
      <Action Name_Action="DENY_TRAFFIC" Priority_Action="1"/>
   </Actions>
   <SupportedByVNFS>
      <Vnf Name="VNF2"/>
   </SupportedByVNFS>
</Functionality>
</Solution>
```

Questa situazione ottenuta può apparire come la soluzione più comoda dal punto di vista del numero di VNFs utili per fare l'enforce del requisito. Se considerassimo come priorità la quantita di risorse impiegate (esempio RAM o CPU occupata) per fare il dispiegamento delle VNFs sulla rete piuttosto che la quantità di VNFs impiegate, probabilmente questa alternativa può essere messa in discussione.

7.3.1 Considerazioni finali

L'utilizzo di una soluzione di tipo A piuttosto che di una soluzione tipo B è una scelta che viene fatta a posteriori. Il motivo principale è l'assenza della visibilità del grafo di rete che si vuole amministrare. Si è scelto di posticipare quanto più possibile l'eliminazione di eventuali soluzioni alla fase finale, ovvero alla fondo della

catena dei blocchi dell'architteura di VEREFUSE. Questo tipo di comportamento è stato adottato per una serie di ragioni tra cui:

- Mantenere la modularità, punto chiave sin dall'inizio nella fase di design del framework;
- Complessità dell'algoritmo;
- Ridurre la complessita sul core del framework di VEREFUSE.

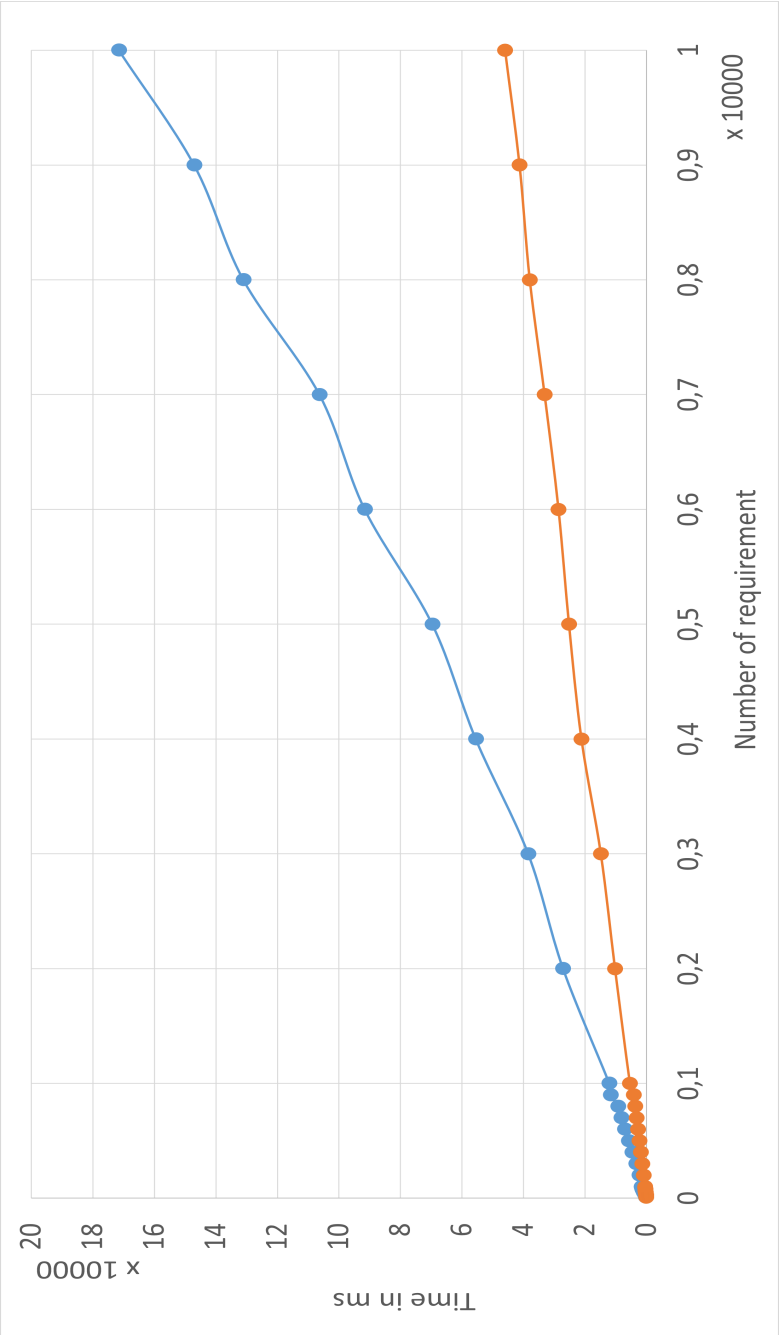

Figura 7.3: Confronto del tempo di esecuzione impiegato per analizzare i requisiti - Linea arancione è l'andamento nel tempo dei tempi esecuzione eseguiti su requisiti con azioni ordinate
-Linea blu è l'andamento nel tempo dei tempi esecuzione eseguiti su requisiti con azioni non ordinate

Figura 7.4: Andamento nel tempo del tempo impiegato per analizzare un numero fissato di requisiti, alternado il numero di VNFs - Thread Variabili

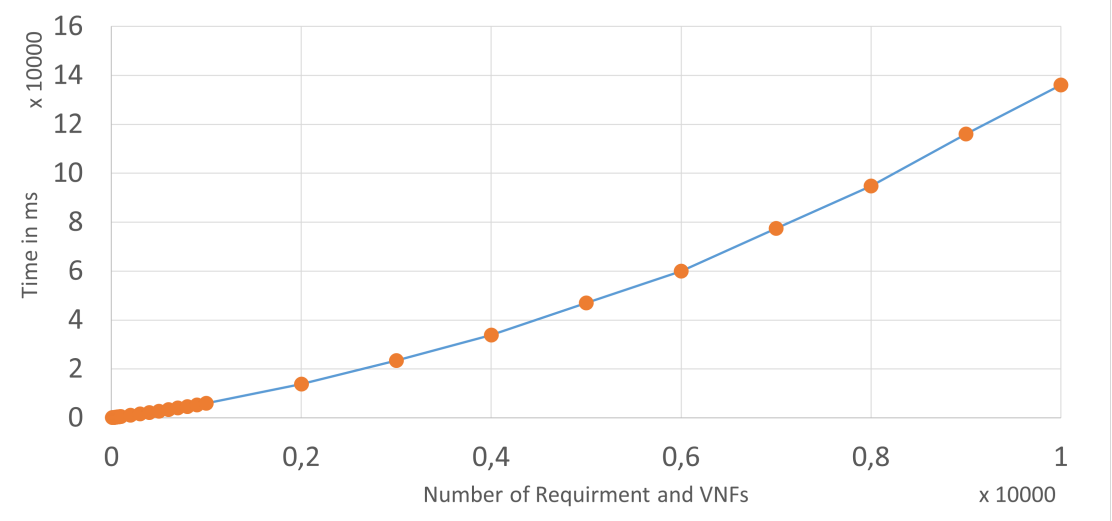

Figura 7.5: Andamento nel tempo del tempo impiegato per elaborare un numero di requisiti pari al numero di VNFs.

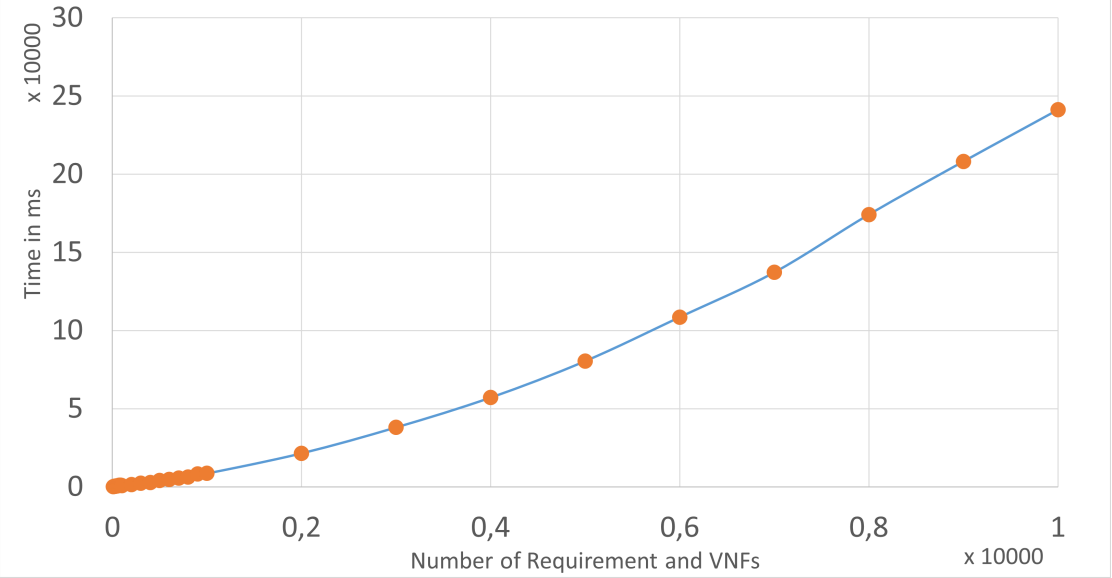

Figura 7.6: Andamento nel tempo impiegato per analizzare un numero sempre crescente di requisiti e VNFs.

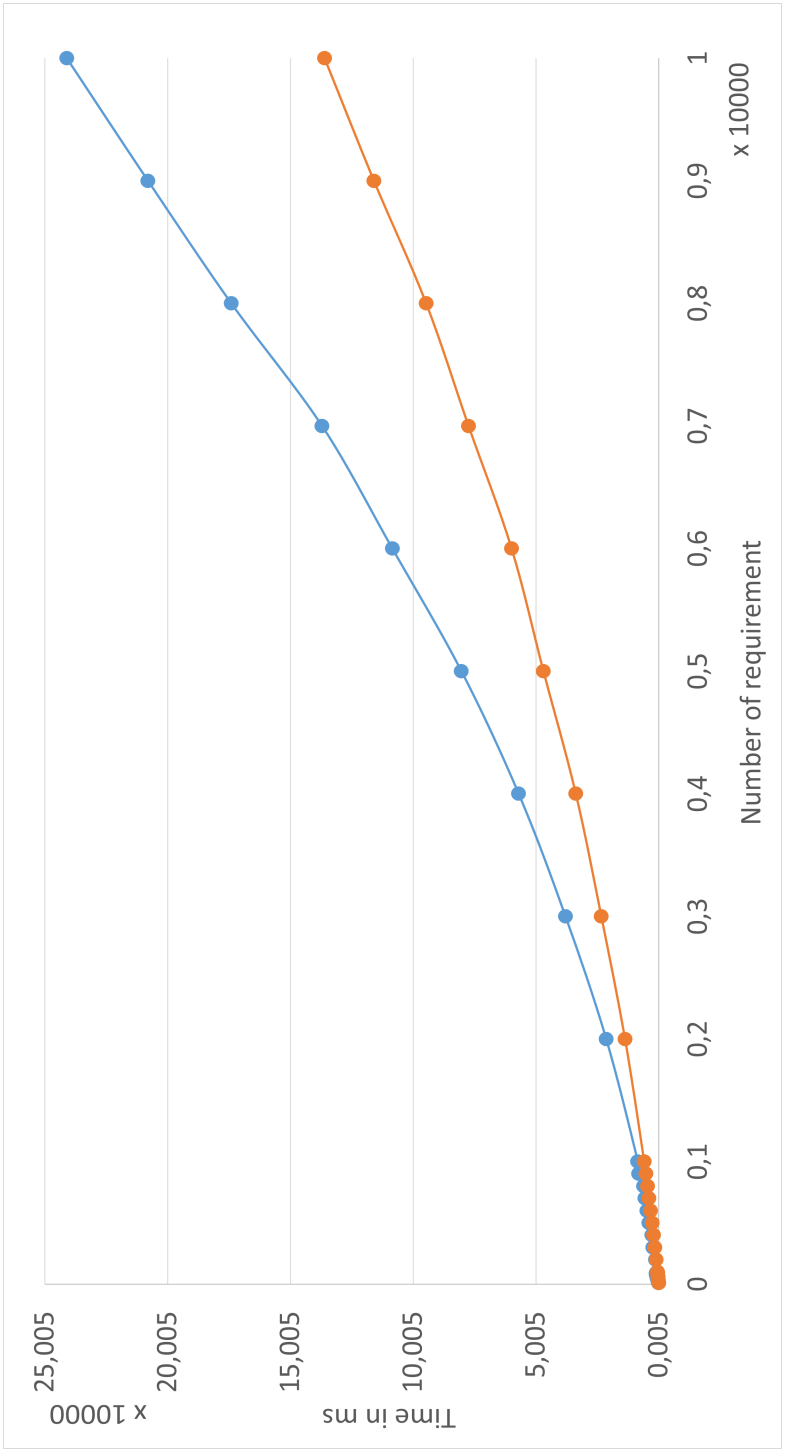

Figura 7.7: Confronto fra test progressivi con numero di requisiti direttamente proporzionale al numero di VNFs

Capitolo 8

Conclusione

Il lavoro di tesi ha dato la possibilità di migliorare uno strumento, estendendo quello che è attualmente lo stato attuale di *Verefuse*. Lo sviluppo ha richiesto dapprima un studio teorico durante la prima fase. Uno studio in cui molta attenzione è stata dedicata ai lavori in letteratura riguardanti reti, sicurezza e policy, e lo studio di alcuni moduli del framework *Verefuse*. Studio che ha portato ad estendere il framework utilizzando come concetto chiave ***Functionality*** attorno al quale è stata sviluppata l'estenzione di cui abbiamo ampiemante discusso nei capitoli precedenti.

Concetto nuovo, menzionato in letteratura ma che non è mai stato usato in questo forma. L'idea di usare le funzionalità come mezzo per rappresentare la capacita di *Enforceability* tra una funzioni di rete e un requisito di sicurezza sembra essere un'idea vincente e gli ottimi risultati dei test lo confermano.

L'idea di funzionalità permette di identificare le funzioni di rete indipendentemente dal venditore, dalle specifiche di sistema, dal utilizzo di risorse di sistema, dal complessita della funzione di rete stessa ecc... . Quindi, partendo dal concetto di funzionalità è stato possibile modellare le Virtual Network Function (VNF) e i Network Security Requirement (NSR). Il catalogo è completo di ogni tipo di informazione sulle funzioni disponibili come hardware, software ed informazioni specifiche per quella funzione di rete mentre il repository dei requisiti contiene informazioni generali come la descrizione della volontà del requisito, l'autore della scrittura della policy e specifiche per descrivere l'intento di una policy di sicurezza.

La base del lavoro è stata lo sviluppo di una serie di schemi XML che convoglia le varie informazioni necessarie per il calcolo interno e per la rappresentare degli input e dei due moduli conivolti nell'estenzione. La sua struttura presenta un alto grado di flessibilità e modularità che permette futuri ampliamenti con svariate

aggiunte. Nella sua versione attuale, lo schema XML garantisce già la possibilità di esprimere un'ampia gamma di scenari riproducibili, tra cui requisiti di rete e servizi di rete realistici,

Il linguaggio XML, permette di creare modelli molto precisi ed estesi che danno anche la possibilità di esprimere vincoli. Questo linguaggio è ben supportato da Java grazie all'utilizzo del framework JAXB[1] che permette un facile marshal e unmarshal, da e verso le classi java.

È stata aggiunta un'integrazione con Neo4j. Attualmente serve come strumento di debug o semplicemente per avere una versione grafica, basata su grafi, delle soluzioni indivituate.

Tuttavia, *Verefuse* è da considerarsi ancora in fase di sviluppo e non è esente da limitazioni. I lavori futuri possono migliorare ulteriormente il software perfezionando alcuni dei problemi delineati e migliorando la sua interoperabilità con gli altri prodotti *NFV* presenti sul mercato.

[1]JAXB (Java Architecture for XML Binding) è un framework a supporto dell'utilizzo dell'XML in Java

Bibliografia

[1] Manuel Cheminod, Luca Durante, Lucia Seno, Fulvio Valenza, Adriano Valenzano e Claudio Zunino. «Leveraging SDN to improve security in industrial networks». In: *IEEE 13th International Workshop on Factory Communication Systems, WFCS 2017, Trondheim, Norway, May 31 - June 2, 2017*. IEEE, 2017, pp. 1–7. DOI: 10.1109/WFCS.2017.7991960. URL: https://doi.org/10.1109/WFCS.2017.7991960 (cit. a p. 1).

[2] Fulvio Valenza, Serena Spinoso e Riccardo Sisto. «Formally specifying and checking policies and anomalies in service function chaining». In: *J. Netw. Comput. Appl.* 146 (2019). DOI: 10.1016/j.jnca.2019.102419. URL: https://doi.org/10.1016/j.jnca.2019.102419 (cit. alle pp. 2, 10).

[3] Guido Marchetto, Riccardo Sisto, Fulvio Valenza e Jalolliddin Yusupov. «A Framework for Verification-Oriented User-Friendly Network Function Modeling». In: *IEEE Access* 7 (2019), pp. 99349–99359. DOI: 10.1109/ACCESS.2019.2929325. URL: https://doi.org/10.1109/ACCESS.2019.2929325 (cit. alle pp. 2, 14).

[4] Thi-Thuy-Lien Nguyen e Tuan-Minh Pham. «Efficient Traffic Engineering in an NFV Enabled IoT System». In: *Sensors* 20.11 (2020), p. 3198. DOI: 10.3390/s20113198. URL: https://doi.org/10.3390/s20113198 (cit. a p. 5).

[5] «ETSI-NFV». In: (2017). URL: %7BHTTP://WWW.ETSI.ORG/TECHNOLOGIES-CLUSTERS/TECHNOLOGIES/NFV%7D (cit. a p. 6).

[6] Diego Kreutz, Fernando M. V. Ramos, Paulo Jorge Esteves Verıssimo, Christian Esteve Rothenberg, Siamak Azodolmolky e Steve Uhlig. «Software-Defined Networking: A Comprehensive Survey». In: *Proc. IEEE* 103.1 (2015), pp. 14–76. DOI: 10.1109/JPROC.2014.2371999. URL: https://doi.org/10.1109/JPROC.2014.2371999 (cit. a p. 7).

[7] Service Function Chaining Technology for Future Networks. «THE DEFINITION OF OSS AND BSS | OSS LINE.» In: (2014). URL: %7Bhttps://www.ntt-review.jp/archive/ntttechnical.php?contents=ntr201408fa2.pdf&mode=show_pd%7D (cit. alle pp. 10, 12).

[8] Cataldo Basile, Fulvio Valenza, Antonio Lioy, Diego R. López e Antonio Pastor Perales. «Adding Support for Automatic Enforcement of Security Policies in NFV Networks». In: *IEEE/ACM Trans. Netw.* 27.2 (2019), pp. 707–720. DOI: 10.1109/TNET.2019.2895278. URL: https://doi.org/10.1109/TNET.2019.2895278 (cit. a p. 10).

[9] Hiroyuki Kitada, Hisashi Kojima, Naoki Takaya,and Masao Aihara. «Service Function Chaining Technology for Future Networks». In: (2014) (cit. a p. 10).

[10] Luca Durante, Lucia Seno, Fulvio Valenza e Adriano Valenzano. «A model for the analysis of security policies in service function chains». In: *2017 IEEE Conference on Network Softwarization, NetSoft 2017, Bologna, Italy, July 3-7, 2017*. IEEE, 2017, pp. 1–6. DOI: 10.1109/NETSOFT.2017.8004230. URL: https://doi.org/10.1109/NETSOFT.2017.8004230 (cit. alle pp. 10, 13, 16).

[11] Guido Marchetto, Riccardo Sisto, Fulvio Valenza, Daniele Bringhetti. «Short Paper: Automatic Configuration for an Optimal Channel Protection in Virtualized Networks». In: (2020) (cit. a p. 11).

[12] Ignazio Pedone, Antonio Lioy e Fulvio Valenza. «Towards an Efficient Management and Orchestration Framework for Virtual Network Security Functions». In: *Secur. Commun. Networks* 2019 (2019), 2425983:1–2425983:11. DOI: 10.1155/2019/2425983. URL: https://doi.org/10.1155/2019/2425983 (cit. a p. 11).

[13] Daniele Bringhenti, Guido Marchetto, Riccardo Sisto, Fulvio Valenza e Jalolliddin Yusupov. «Automated optimal firewall orchestration and configuration in virtualized networks». In: *NOMS 2020 - IEEE/IFIP Network Operations and Management Symposium, Budapest, Hungary, April 20-24, 2020*. IEEE, 2020, pp. 1–7. DOI: 10.1109/NOMS47738.2020.9110402. URL: https://doi.org/10.1109/NOMS47738.2020.9110402 (cit. a p. 13).

[14] Erisa Karafili e Fulvio Valenza. «Automatic Firewalls' Configuration Using Argumentation Reasoning». In: *Emerging Technologies for Authorization and Authentication - Third International Workshop, ETAA 2020, Guildford, UK, September 18, 2020, Proceedings*. A cura di Andrea Saracino e Paolo Mori. Vol. 12515. Lecture Notes in Computer Science. Springer, 2020, pp. 124–140. DOI: 10.1007/978-3-030-64455-0_8. URL: https://doi.org/10.1007/978-3-030-64455-0%5C_8 (cit. a p. 13).

[15] Daniele Bringhenti, Guido Marchetto, Riccardo Sisto, Fulvio Valenza e Jalolliddin Yusupov. «Towards a fully automated and optimized network security functions orchestration». In: *2019 4th International Conference on Computing, Communications and Security (ICCCS), Rome, Italy, October 10-12,*

2019. IEEE, 2019, pp. 1–7. DOI: 10.1109/CCCS.2019.8888130. URL: https://doi.org/10.1109/CCCS.2019.8888130 (cit. a p. 13).

[16] Daniele Bringhenti, Guido Marchetto, Riccardo Sisto, Serena Spinoso, Fulvio Valenza e Jalolliddin Yusupov. «Improving the Formal Verification of Reachability Policies in Virtualized Networks». In: *IEEE Trans. Netw. Serv. Manag.* 18.1 (2021), pp. 713–728. DOI: 10.1109/TNSM.2020.3045781. URL: https://doi.org/10.1109/TNSM.2020.3045781 (cit. a p. 13).

[17] Manuel Cheminod, Luca Durante, Lucia Seno, Fulvio Valenza e Adriano Valenzano. «A comprehensive approach to the automatic refinement and verification of access control policies». In: *Comput. Secur.* 80 (2019), pp. 186–199. DOI: 10.1016/j.cose.2018.09.013. URL: https://doi.org/10.1016/j.cose.2018.09.013 (cit. alle pp. 13, 14).

[18] Daniele Bringhenti, Guido Marchetto, Riccardo Sisto, Fulvio Valenza e Jalolliddin Yusupov. «Introducing programmability and automation in the synthesis of virtual firewall rules». In: *6th IEEE Conference on Network Softwarization, NetSoft 2020, Ghent, Belgium, June 29 - July 3, 2020*. A cura di Filip De Turck, Prosper Chemouil, Tim Wauters, Mohamed Faten Zhani, Walter Cerroni, Rafael Pasquini e Zuqing Zhu. IEEE, 2020, pp. 473–478. DOI: 10.1109/NetSoft48620.2020.9165434. URL: https://doi.org/10.1109/NetSoft48620.2020.9165434 (cit. a p. 13).

[19] Fulvio Valenza. «Modelling and Analysis of Network Security Policies». In: (2020) (cit. a p. 18).

Appendice A

Ordinamento automatico di azioni coinvolte nel requisito

Nel modulo **Functionalities Selection** si fa riferimento più volte come un requisito possa imporre un ordinamento nello svolgimento delle *azioni*. Questo argomento è molto delicato, perché una cattiva gestione delle azioni provoca malfunzionamenti o interruzioni del traffico che fluisce da una sorgente ad una destinazione. Si è voluto lasciare questo parte del processo separato dal blocco analizzato in dettaglio nella tesi, per evitare di appesantire il modulo corrente ma soprattutto ha una affinità maggiore con il blocco **H-To-M translator**. Il modello di requisito permette di poter esprimere un ordinamento con cui le azioni devo essere svolte.L'ordinamento con cui vengono svolte le operazioni non è indifferente e sappiamo essere uno dei motivi per cui si possono allungare i tempi di esecuzione sopratutto per numero di dati molto ampio .

Se volessi negare un determinato traffico raggiunga una destinazione o un semplice servizio all'interno della rete e contemporaneamente memorizzare tutti i pacchetti in arrivo (magari è utile per svolgere a posteriori delle statistiche sul traffico in quesitone) , la scelta di quale azione deve essere svolte prima o per ultima è importante. Se facessimo prima l'operazione di *deny* e poi l'operazione di *store*, probabilmente dopo il deny non si è più in grado di recupere informazioni del traffico. Viceversa non accadrebbe. Questo è un esempio banale, ma basta poco per complicare l'esempio e vedere quanta importanza ha un ordinamento in una policy di sicurezza.
Non si esclude anche la possibilità di trovarci tranquillamente nei casi in cui comunque si facciano le operazioni i risultati non cambiano. Casi meno frequenti ma comunque possibili. Questo permette all'amministratore di rete di poter usufruire di una certa elasticità ma allo stesso tempo è incaricato di una grossa responsabilità.

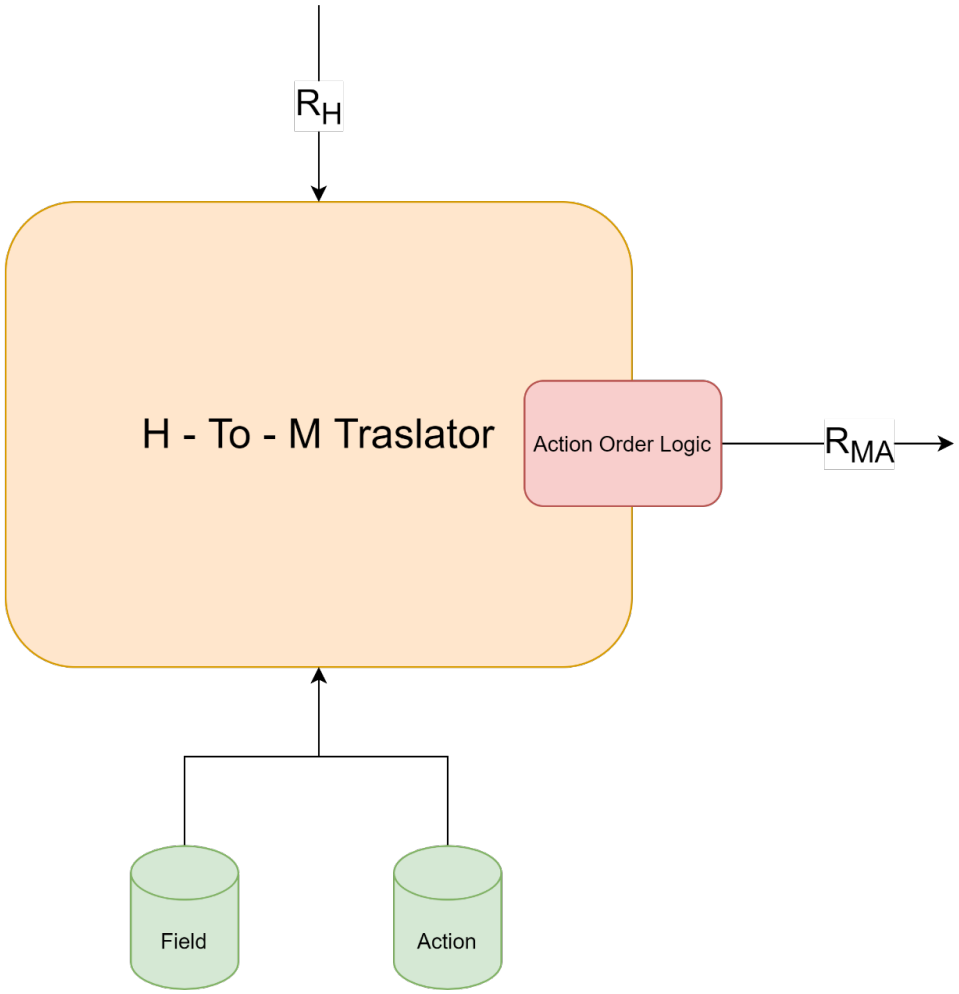

Figura A.1: Estenzione del modulo H-To-M translator

Motivazioni all'introduzione del modulo di ordinamento

Come Futura implementazione si potrebbe provvedere un *modulo* aggiuntivo presente nella processo di trasformazione H-To-M Traslator la possibilità di introdurre meccanismi automatici, utili per permettere all'amministratore di rete di spendere meno energie e tempo nella definizione di un ordinamento corretto. Tipicamente sono due le ragioni per cui vale la pena un introdurre questa specifica:

1. L'amministratopre potrebbe non essere in grado di farlo. Tipicamente accade per un utente con poche conoscenze in materia o semplicemente è un nuovo utente che si affaccia per le prime volte in una gestione più complessa di un requisito di rete.

2. È coinvolto in numero così elevato di azioni che non è in grado in tempi brevi di raggiungere una soluzione corretta, cadendo in intoppi che provocano un malfunzionamento della rete. Questo permette all'utente di focalizzarsi ancora di più quella che è il requisito, evitando di preoccuparsi di alcune informazioni di contorno.

Questo futura implementazione comporta una conoscenza piu appofondita dei vari linguaggi HLP; MLP; LPL usati nel workflow completo di Verefuse.

Appendice B

Manuale dello sviluppatore

Il seguente capitolo mostra come effeuttare il setup del framework agganciando in maniera corretta i vari tool di supporto usati per il corretto svolgimento delle operazioni di calcolo. Per questi tool verranno forniti una guida su come installarli e configurarli in maniera corretta.

B.1 Linee Guida

B.1.1 JAVA JDK 8 SE

JDK (Java Development Kit)è un ambiente di sviluppo per la creazione di applicazioni, applet e componenti utilizzando il linguaggio di programmazione Java.

Il JDK include strumenti utili per lo sviluppo e il test di programmi scritti nel linguaggio di programmazione Java in esecuzione sulla JVM.
Ai suegenti link puoi trovare informazioni ulteriori su come installare JAVA sui vari sistemi operativi , la documentazione JDK , link per il download di JDK.

- `https://docs.oracle.com/javase/8/` documentazione generale su JAVA JDK 8.
- `https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html` pagina di download di JAVA JDK 8. Per ragioni di compatibilità è consigliabile scegliere JAVA JDK SE 8.
- `https://docs.oracle.com/javase/8/docs/technotes/guides/install/install_overview.html` puoi trovare tutta la documentazione su come installare java su tutti i sistemi operativi.

Dopo aver scaricato e installato java è necessario andare a configurare la variabile di ambiente sul sistema operativo

In Linux:
JAVA_HOME="the/complete/path/to/your/java/installation/jdk1.8.0/" In Windows: JAVA_HOME="C:/ProgramFiles/Java/jdk1.8.0/"

Per verificare la corretta installazione di Java JDK 8 SE, in linux puoi digitare

$ java –version.

B.1.2 Apache Ant

Apache Ant è una libreria Java, strumento da riga di comando usato sopratutto per guidare i processi build con estensione dipendenti l'uno dall'altro. Il principale utilizzo noto di Ant è la creazione di applicazioni Java. Ant integra una serie di attività che consentono di compilare, assemblare, testare ed eseguire applicazioni Java. Nel framework questo file va sotto il nome di build.xml.

Ant non impone convenzioni di codifica o layout di directory ai progetti Java che lo adottano come strumento di compilazione.

- `https://ant.apache.org/manual/index.html` trovi il manuale di riferimento per l'ultima versione di Ant.
- `https://ant.apache.org/bindownload.cgi` trovi la pagina download di Ant. E' consigliabile usare la versione di ant 1.10.x o superiore per il corretto funzionamento e integrazione con Java JDK 8.

Dopo il download è necessario aggiungere un nuova variabile di ambiente, come nello step precedente.

In Linux: ANT_HOME="the/complete/path/to/your/ant/installation/bin" In Windows: ANT_HOME="C:/ProgramFiles/Ant/bin/"

Per verificare la corretta installazione di Apache Ant, su linux puoi digitare

$ ant -version

B.1.3 Neo4J

Neo4j (Network Exploration and Optimization 4 Java) è un sistema di gestione di database a grafo sviluppato da Neo4j, Inc.

Neo4j funziona con una struttura di rete flessibile di nodi e relazioni piuttosto che con tabelle statiche, ma gode di tutti i vantaggi di un database. Per molte applicazioni, Neo4j offre vantaggi in termini di prestazioni di ordini di grandezza rispetto ai DB relazionali.

I seguenti link forniscono informazioni dettagliate sul tool:

- `https://neo4j.com/sandbox/` puoi provare un versione onlinee del tool.
- `https://neo4j.com/download/` puoi scaricare la versione desktop per il tuo sistema operativo.
- `https://neo4j.com/docs/` documentazione completa su tutte le versioni ed editioni del tool
- `https://neo4j.com/download-center/#community` pagina del download a Neo4j Community edition utile per il corretto funzionamento del tool. E' consigliabile installare la versione Neo4j 3.5.x .

Dopo il download è necessario aggiungere un nuova variabile di ambiente, come nello step precedente.

In Linux: NEO4J_HOME="the/complete/path/to/your/neo4j/installation/neo4j-community-3.5.x"
In Windows: NEO4J_HOME="C:/ProgramFiles/neo4j-community-3.5.x/"

Per testare il corretto funzionamento di Neo4j puoi recarti in 'the/complete/path/ neo4j/installation/neo4j-community-3.5.x/bin' ed eseguire il file 'neo4j'. Il framework al momento del build provvederà in maniera automatica a lanciare il Neo4jServer.

B.1.4 Gurobi

Gurobi Optimizer è un ottimizzatore per la programmazione lineare (LP), la programmazione quadratica (QP), la programmazione quadratica vincolata (QCP), la programmazione lineare mista intera (MILP), la programmazione quadratica mista intera (MIQP) e la programmazione quadratica mista intera quadratica programmazione vincolata (MIQCP).

Gurobi Optimizer supporta una varietà di linguaggi di programmazione tra cui anche JAVA (liguaggio di programmazione usato per lo sviluppo del framework)
I seguenti link fornisco informazioni dettagliate sul tool:

- `https://www.gurobi.com/wp-content/plugins/hd_documentations /documentation/9.1/refman.pdf` puoi trovare il manuale di riferimento su l'utilizzo di Gurobi.

- `https://www.gurobi.com/documentation/quickstart.html` puoi trovare la documentazione su come installare il tool nei vari sistemi operativi.

- `https://www.gurobi.com/downloads/` pagina di download di Gurobi.

Dopo il download è necessario eseguire i seguenti step al fine di portare a termine la corretta configurazione di Gurobi.

In linux :

- *sudo nano /etc/environment* - Inserisci la seguente riga: *GUROBI_HOME = /opt/gurobi911/linux64*
- *sudo nano /.bashrc* - Inserisci le seguenti righe:
  - *export GUROBI_HOME = /opt/gurobi911/linux64*
  - *export LD_LIBRARY_PATH = LD_LIBRARY_PATH : GUROBI_HOME/lib*
  - *export PATH = PATH :GUROBI_HOME/bin*

In fine è necessario copiare la licenza di Gurobi nella cartella */usr/lib/*

In Windows :

- *GUROBI_HOME = C:/gurobi911/win64*
- *PATH = C:/gurobi911/win64/bin*

Al termine della procedura il tool è installato e configurato correttamente per interagire con il framework.

B.2 Strutta delle cartelle del framework

In questo sezione viende descritta l'organizzazione delle cartelle utilizzate dal framework. Inoltre è stata fornita una documentazione java doc utile per descrive parametri, valori di ritorni e comportamenti per ciascun metodo utilizzabili dalle varie classi.

L'organizzazione del progetto è impostata nel seguente modo :

- La cartella **src** contiene tutti i package riguardanti il framework, in particolare:
  - Il package **it.polito.verifuse.main** contiene il main principale del framework;
  - Il package **it.polito.verifuse.moudles** contiene le classi per i due moduli RDF - LER;
  - Il package **it.polito.verifuse.neo4jClient** contiene le classi necessarie per insierimento dati nel db ;
  - Il package **it.polito.verifuse.utility** contiene le classi per la gestione dei dati; Esempio: la gestione degli input;
  - Il package **it.polito.verifuse.framework.test** contiene classi e metodi per svolgere i test su ciascun modulo;
    - **test** include le classi java per il testing dei singoli moduli;
    - **test_scalability** contiene tutti i test svolti per analizzare le prestazioni del tool con un numero di dati elevato (fino un milione).
  - Il package **it.polito.verifuse.functionality** contiene le classi per gestire l'entità 'funzionalità';
- La cartella **xsd** contiene tutti gli schemi delle strutture dati usati
- La cartella **gen-src** contiene tutti le strutture dati generate automaticate durante la fase di build del framework. Esso sono generate da JAXB framework , partendo dagli schema xsd presenti nella cartella *src*.
- La cartella **lib** include tutte le librerie usate dal tool.
- La cartella **test** contiene le classi per lo svolgimento dei test JUnit.
- La cartella **outputChian** contiene file xml generati dal framework . Esse rappresentano le soluzioni trovate dal tool.
- La cartella **test** contiene le classi per lo svolgimento dei test Junit.

- La cartella **resultTest** contiene tutti i risultati dei test Junit.
- La cartella **xmlInputFile** contiene file xml che vengono caricate all'avvio del main.
- La cartella **UseCases** contiene file xml dove sono mostrati dei casi d'uso significativi.
- La cartella **docs** documentazione delle funzioni/metodi usati nel framework.

File per il build del progetto:

- Il **build.xml** file permette il build del tool;
- Il **service-build.xml** file permette della gestione del dispiegamento del framework.
- Il **neo4j-build.xml** file permette di gestire il tool Neo4j;
- Il **tools.xml** file permette di gestire l'attivazione/disattivazione di alcuni servizi durante il processo di esecuzione. Esempio attivare l'output sul file , attivazione salvataggio su Neo4j , attivazione marshalling delle soluzioni torvate dal tool.

Appendice C

Estensione codice

Questo capito vuole essere una documentazione di rapido di utilizzo, utile per permettere allo svilupparotre di poter estendere ulteriormente il supporto a diverse VNFs o estendere l'espressività di un requisito di rete.

C.1 Inserimento nuova VNF

Per aggiungere un ulteriore VNFs al catalogo corrente, è necessario andare a modificare i seguenti file

- */xmlInputFile/ManifestVNFs.xml* : file xml che racchiude tutte le VNF presenti nel catalogo. L'inserimento di una nuova VNF con tutte le sue specifiche deve essere necessarimanete aggiunta all'interno di questo file, compilando un numero minimo di campi e rispettando i vincoli imposti nel rispettivo schema xsd (*/xsd/ManifestVNFs.xsd*).

- */xsd/ActionManifest.xsd* : schema xml usato per andare a inserire il supporto per nuove azioni. E' sufficente andare ad aggiungere il valore all'interno dell'elemento *typeAction*.
  Es. *<enumeration value="NewActionSupported"></enumeration>*

- */xsd/FieldManifest.xsd*: schema xml usato per andare a estendere il supporto per nuovi campi. E' sufficente andare ad aggiungere il valore all'interno dell'elemento *enumField*.
  Es. *<enumeration value="NewFieldSupported"/>*

Per agevolare il supporto per quei campi che possono assumere un set valori ampi, è possibile inserire la lista questi valori all'interno di un file di supporto: *FileSupport.xml*. Il file menzionato puo essere usato sia per andare a enumerare

tutti i valori per determinati campi , ma anche per elencare tutti i constraints associabili alle azioni di una VNF.

```
<Suite NameSuite="MySuite">
   <SupportValue value="value1"/>
   <SupportValue value="value2"/>
   ...
</Suite>
```

C.2 Inserimento nuovo Requisito

Per aggiungere un nuovo requisito all'interno del repository, è necessario inserirlo nel file */xmlInputFile/RequirementsList.xml* completando tutto i campi obbligatori. Per modificare l'epressività di un requisito è necessario modifcare alcuni file tra cui:

- */xsd/Actions.xsd* : schema xml usato per aggiungere il supporto di nuove azioni. E' sufficente andare ad aggiungere il valore all'interno dell'elemento. *typeAction*.
  Es. *<enumeration value="NewActionSupportedRequirement"/>*

- */xsd/Fields.xsd*: schema xml usato per andare a estendere il supporto per nuovi campi. E' sufficente andare ad aggiungere il valore all'interno dell'elemento *enumField*.
  Es. *<enumeration value="NewFieldSupportedRequirement"/>*

Con il file xsd forniti è possibile modificare anche la strutta di base sia di un requisito che di una VNF. La modifica della struttura xsd implica un necessario rebuild del framework. Il rebuild avvine attravero il file Ant *Build.xml* , selezionando il target *"Build"*.

C.3 Classi Java coinvolte nelle modifiche

L'aggiunta o la rimozione di parametri (azioni o campi, che siano di un requisito o di una VNF) devono essere riportate anche nella classe */src/it/polito/verifuse/utiliy/classes/GetDataInput.java* adattado il codice alle nuove esigenze. Il file *GetDataInput.java* è stato creato al fine di separare gli input del framework dalla logica algoritmica usata. Questa struttura ne agevola la modifica e la manutenzione del codice. La classe che effettua l'operazione unmarshalling non necessita in alcun modo modifiche.

Supponiamo di voler inserire come campo l'header del pacchetto IP. Dove l'intento è quello di verificare dei particolari tipi di pattern attraverso l'applicazione delle regular expression. Il codice seguente deve essere inserito all'interno di questa classe java : */src/it/polito/verifuse/utiliy/classes/GetDataInput.java*. Un codice di esempio può essere il seguente.

```
if( f.getHeaderIP() != null){
    hs=new HashSet<>();
    hs.add(f.getHeaderIP());
    fields.put(Consts.HEADERIP, hs);
}
```

Fields è una mappa contente come chiave l'identificativo del campo, e come valore il valore del campo o i valori associati ad esso. La variabile *Consts.HEADERIP* è una costante stringa definita nella classe java, Consts.java, presente al path */src/it/polito/verifuse/utiliy/classes*.

Si tratta di un aggiunta di un semplice campo, di conseguenza la quantità di codice da aggiungere è minima. L'aggiunta di campi piu complessi con caratteristiche dettagliate necessità del codice aggiuntivo per adattare il nuovo campo al framework.

E' possibile estendere anche l'algoritmo responsabile dell individuazione delle catene di funzionalità. Nel caso in cui si volesse aggiungere ulteriori vincoli , variabili binarie o condizioni che ne vincolino la deteminazione delle soluzione , è sufficiente modificare la classe */src/it/polito/verifuse/utiliy/functionality/SolverChain.java*. Nella prima parte sono state costruite strutture dati necessarie per andare a contenere i vincoli per poter esprimere equazione e disequazioni lineari. Il risolutore inizia l'eleborazione chiamando il metodo *model.optimize()*. Al termine, in caso di esito positivo, vengono salvate tutte le soluzioni generate in apposite strutture dati. Inoltre viene effettuato un ulteriore verifica con granularità di dettaglio molto fine (attivando quello che è il reduction process). Per effettuare modifiche sulla logica risolutiva e sulla definizioni di vincoli come, equazioni, varibili binarie, variabili continue, disequazioni, è richiesta la conoscenza delle strutture dati e delle notazione proprie di Gurobi.

Appendice D

Modelli requisiti e VNFs

Questo capitolo vuole completare la descrizione dei parametri modellati in xml, già precedentmente analizzato nel capitolo 5. Si fa riferimento ai parametri caratterizzanti di un requisito e sia a parametri specifichi per una VNFs.

D.1 Parametri Requisito

Domains

Ogni Domains è composto da una serie di Domain, seguiti dall'attributo domainValue che rappresente il valore dominio stesso.

```
<element name="Domains" type="Domainlist" minOccurs="0" maxOccurs="1"/>
<complexType name="Domainlist">
   <sequence>
      <element name="Domain"  maxOccurs="unbounded">
         <complexType>
            <attribute name="DomainValue" type="anyURI"/>
         </complexType>
      </element>
   </sequence>
</complexType>
```

Vlan tag

Rappresenta il valore dal tag vlan.

```
<element name="VlanTag" type="string" minOccurs="0" maxOccurs="1"/>
```

Media

Rappresenta l'insieme dei dati per andare ad esprimere il concetto Media il campo MediaType. MediaType associabili al campo

```
<element name="Medias" type="tns:Medias" minOccurs="0" maxOccurs="1"/>
<complexType name="Medias">
   <sequence>
      <element name="Media"   minOccurs="1" maxOccurs="unbounded">
         <complexType>
            <attribute name="Type" type="tns:MediaType" use="required"/>
            <attribute name="Regex" type="string" use="optional"/>
         </complexType>
      </element>
   </sequence>
</complexType>
<simpleType name="MediaType">
   <restriction base="string">
      <enumeration value="AUDIO.MP3"/>
      <enumeration value="VIDEO.MP4">
      <enumeration value="IMAGES.PNG"/>
      <enumeration value="IMAGES.JPG"/>
      <enumeration value="*"/>
   </restriction>
</simpleType>
```

Advertising

```
<element name="Advertisings" type="Advertisings" minOccurs="0" maxOccurs="1">
<complexType name="Advertisings">
   <sequence>
      <element name="Advertising" minOccurs="1" maxOccurs="unbounded">
      <complexType> <attribute name="Type" type="string" use="required"/>
                  <attribute name="Regex" type="string" use="optional"/>
      </complexType>
      </element>
   </sequence>
</complexType>
```

Priority

Rappresenta il campo priorià di un pacchetto usato per andare a gestire la QoS.

```
<element name="Priority" type="integer" minOccurs="0" maxOccurs="1">
```

GeoPosition

Rappresenta la possbilità di geolocalizzare la sorgente o destinazione del traffico dati.

```
<element name="GeoPosition" type="tns:Position" minOccurs="0" maxOccurs="1"/>
<complexType name="Position">
    <attribute name="Country" type="tns:countrytype" use="required"/>
</complexType>
```

Bandwidth

Il parametro bandiwidth permette di definire un limite massimo di carico sul traffico dati passante per un canale di comunicazione. Esso è espresso nella forma *valore:unità*. Es 10Mbit/s

```
<element name="Bandwidth" type="BandType" minOccurs="0" maxOccurs="1"/>
<complexType name="DaysAndTimesType">
<simpleType name="BandType">
    <restriction base="string">
        <pattern value="[0-9]*[KMG]bit\/s"/>
    </restriction>
</simpleType>
```

Body

Utili per andare ad esprimere vincoli sul contenuto del pacchetto.

```
<element name="Body" type="string"  minOccurs="0" maxOccurs="1"/>
```

D.2 Parametri VNF

Developers

L'insieme degli sviluppatori che hanno partecipato alla sviluppo del software;

```
<complexType name="GeneralInfo">
<element name="Developers"  minOccurs="0" maxOccurs="1">
<complexType>
   <sequence>
      <element name="Developer" type="string" minOccurs="1" maxOccurs="
    unbounded"/>
   </sequence>
</complexType>
</element>
```

VersionInfo

L'insieme degli sviluppatori che hanno partecipato alla sviluppo del software. Gli attributi sono espressi tutti tramite espressioni regolari.

- Version, il numero di versione
- Release, il numero di release
- releaseDate, la data di rilascio della release.

```
<element name="VersionInfo" minOccurs="0" maxOccurs="1">
   <complexType>
      <attribute name="version" type="tns:regExrVersion" use="required"/>
      <attribute name="release" type="tns:regExrVersion" use="required"/>
      <attribute name="releaseDate" type="tns:regExrDate" use="required"/>
   </complexType>
</element>
```

```
<!-- Regular expression  -->
<simpleType name="regExrVersion">
    <restriction base="string">
        <pattern value="((\d+)(\.\d+)*)"/>
    </restriction>
</simpleType>

<simpleType name="regExrDate">
    <restriction base="string">
        <pattern value="(\d{4})(\.|:|\\)(([0]{0,1}[1-9])|(1{1}[0-2]{1}))(\.|:|\\)(([0-2])
    {0,1}[0-9]{1}|(3{1}[0-1]{1}))"/>
    </restriction>
</simpleType>
```

Repository

L'eventuale presenza di repository da cui scaricare le utlime versioni del software

```
<element name="Repository"  type="anyURI"  minOccurs="0"  maxOccurs="1"/>
```

ProgrammingLanguagesUsed

L'insieme dei linguaggi di programmazione utilizzati per sviluppare il software delle VNFs. L'insieme completo è espresso attraverso delle enumerazioni, tra cui troviamo:

```
<simpleType name="ProgrammingLanguage">
    <restriction base="string">
        <enumeration value="C"/>
        <enumeration value="C++"/>
        <enumeration value="Java"/>
        <enumeration value="JavaScript"/>
        <enumeration value="Matlab"/>
        <enumeration value="Python"/>
        <enumeration value="R"/>
        <enumeration value="SQL"/>
        <enumeration value="Visual Basic"/>
        ...
    </restriction>
</simpleType>
```

OperatingSystemsSupported

L'insieme dei sistemi operativi supporti dal software delle VNFs espressi attraverso attributi di supporto quali:

- OS_TYPE, permette di definire il tipo di sistema operativo
- OS_VERSION, la versione del sistema operativo
- OS_Architecture, l'architettura hardware del sistema operativo

Questi parametri permettono di specificare la compatibilità con i sottosistemi. Utili quando queste VNFs dovranno essere instaziate su dei server.

```
<element name="OperatingSystemsSupported" minOccurs="0" maxOccurs="1">
<complexType>
   <sequence>
   <element name="OperatingSystem" minOccurs="1"   maxOccurs="unbounded">
      <complexType>
      <attribute name="OS_Type"          type="OS_Type"/>
      <attribute name="OS_Version"       type="string"/>
      <attribute name="OS_Architecture"  type="Architecture"/>
      </complexType>
   </element>
   </sequence>
</complexType>
</element>
```

```
<!-- Sistemi operativi supportati -->
<simpleType name="OS_Type">
  <restriction base="string">
    <enumeration value="ARM"/>
    <enumeration value="Android"/>
    <enumeration value="BSD"/>
    <enumeration value="CentOS"/>
    <enumeration value="Debian"/>
    <enumeration value="Fedora"/>
    <enumeration value="Fink"/>
    <enumeration value="FreeBSD"/>
    <enumeration value="OpenBSD"/>
    <enumeration value="Gentoo"/>
    <enumeration value="IBM"/>
    <enumeration value="iOS"/>
    <enumeration value="Linux"/>
    <enumeration value="macOS"/>
    <enumeration value="Maemo"/>
    <enumeration value="Mandriva"/>
    <enumeration value="NetBSD"/>
    <enumeration value="Slackware"/>
    <enumeration value="SLES"/>
    <enumeration value="Solaris"/>
    <enumeration value="Ubuntu"/>
    <enumeration value="Unix"/>
    <enumeration value="Windows"/>
    <enumeration value="Other"/>
    <enumeration value="Unspecified"/>
  </restriction>
</simpleType>
```

L' architetture hardware supportata dalla VNFs.

```
<!-- Definizione delle Achitetture -->
<simpleType name="Architecture">
   <restriction base="string">
      <enumeration value="x86"/>
      <enumeration value="x64"/>
      <enumeration value="ARM"/>
      <enumeration value="Unspecified"/>
   </restriction>
</simpleType>
```

Licence

Lincenza d'uso del software usato dalla VNF. Esse spaziano da licenza "open source" a licenze commerciali.

```
<element name="Licence" type="tns:Licence"  minOccurs="0" maxOccurs="1"/>
<simpleType name="Licence">
   <restriction base="string">
      <enumeration value="BSD"/>
      <enumeration value="Commercial"/>
      <enumeration value="Freeware"/>
      <enumeration value="GPL"/>
      <enumeration value="MIT"/>
      <enumeration value="Other"/>
      <enumeration value="Unspecified"/>
   </restriction>
</simpleType>
```

CPU

Informaztioni riguardanti la quantità di CPU occupato durante l'esecuzione del software, espressa in un valore decimale seguito dall'unità di misura in Ghz o Mhz.

```
<!-- Definition of the CPU Type element -->
<complexType name="CPU">
<attribute name="value" type="decimal"      use="required"/>
<attribute name="unit"  type="tns:CPU_Unit" use="required"/>
</complexType>
```

```
<!-- Definition of the CPU Unit -->
<simpleType name="CPU_Unit">
  <restriction base="string">
    <enumeration value="Ghz"/>
    <enumeration value="Mhz"/>
  </restriction>
</simpleType>
```

RAM

La quantità dui ram occupata espressa in un valore numerico seguito da l'unita di misura KB,MB,GB.

```
<!-- Definition of the RAM Type element -->
<complexType name="RAM">
    <attribute name="value" type="integer"      use="required"/>
    <attribute name="unit"  type="tns:RAM_Unit" use="required"/>
</complexType>

<!-- Definition of the RAM Unit -->
<simpleType name="RAM_Unit">
<restriction base="string">
  <enumeration value="KB"/>
  <enumeration value="MB"/>
  <enumeration value="GB"/>
</restriction>
</simpleType>
```

DiskUnit

Lo spazio occuapto sul disco rigiso espresso da un valore numerico seguito dall'unita di misura KB,MB,GB,TB.

```
<!-- Definition of the Disk Type element -->
<complexType name="Disk">
    <attribute name="value" type="integer"      use="required"/>
    <attribute name="unit"  type="tns:Disk_Unit" use="required"/>
</complexType>
```

```
<!-- Definition of the Disk Unit -->
<simpleType name="Disk_Unit">
  <restriction base="string">
    <enumeration value="KB"/>
    <enumeration value="MB"/>
    <enumeration value="GB"/>
    <enumeration value="TB"/>
  </restriction>
</simpleType>
```

Bandwidth

LA banda occuapta sul canale di comunicazione rappresentatp da un valore intero seguito dall'unita di misura Kb/s,Mb/s,Gb/s.

```
<!-- Definition of the Bandwidth_Type element -->
<complexType name="Bandwidth">
    <attribute name="value" type="integer"            use="required"/>
    <attribute name="unit"  type="tns:Bandwidth_Unit" use="required"/>
</complexType>
<simpleType name="Bandwidth_Unit">
  <restriction base="string">
    <enumeration value="Kb/s"/>
    <enumeration value="Mb/s"/>
    <enumeration value="Gb/s"/>
  </restriction>
</simpleType>
```

Cost

Il costo di acquisto del software della VNF. Esso è rappresentato da un valore intero.

```
<element name="Cost" type="integer" minOccurs="0" maxOccurs="1"/>
```

Delay

Ritardo di comunicazione sulla rete della VNF espressa in millisecondi, microsecondi o secondi.

```
<!-- Definition of the Delay_Type element -->
<complexType name="Delay">
   <attribute name="value" type="integer" use="required"/>
   <attribute name="unit" type="tns:MaxDelay_Unit" use="required"/>
</complexType>
```

```
<simpleType name="MaxDelay Unit">
  <restriction base="string">
    <enumeration value="ms"/>
    <enumeration value="micros"/>
    <enumeration value="s"/>
  </restriction>
</simpleType>
```

D.3 Suite

La lista di suite che è mostrato qui di seguito, rappresentano un insieme valori definibili dall'utente e richiamabili nelle VNFs in modo da rendere piu semplice la definizioni del supporto dei parametri per le VNF.

```
<Suite NameSuite="AUTHN_TRAFFIC_base">
   <SupportValue value="PSK"/>
   <SupportValue value="KERBEROS"/>
   <SupportValue value="SRP"/>
</Suite>
<Suite NameSuite="PROTOCOL_base">
   <SupportValue value="TCP"/>
   <SupportValue value="UDP"/>
   <SupportValue value="HTTP"/>
   <SupportValue value="HTTPs"/>
   <SupportValue value="SMTP"/>
   <SupportValue value="SMNP"/>
</Suite>
```

```
<Suite NameSuite="VPN_TRAFFIC_base">
    <SupportValue value="SSL_NULL_WITH_NULL_NULL"/>
    <SupportValue value="SSL_RSA_WITH_NULL_SHA"/>
    <SupportValue value="SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5"/>
    <SupportValue value="SSL_RSA_WITH_3DES_EDE_CBC_SHA"/>
    <SupportValue value="TLS_DHE_DSS_WHIT_3DES_EDE_CBC_SHA"/>
</Suite>
<Suite NameSuite="ENCRYPT_TRAFFIC_base">
    <SupportValue value="AES-128-CBC"/>
    <SupportValue value="AES-128-CBC"/>
    <SupportValue value="AES-192-CBC"/>
    <SupportValue value="AES-192-CBC"/>
    <SupportValue value="AES-256-CBC"/>
    <SupportValue value="AES-256-CBC"/>
</Suite>
<Suite NameSuite="PROTECT_INTEGRITY_base">
    <SupportValue value="SHA1"/>
    <SupportValue value="SHA224"/>
    <SupportValue value="SHA256"/>
    <SupportValue value="SHA512"/>
    <SupportValue value="SHA3"/>
    <SupportValue value="RIPEMD-160"/>
    <SupportValue value="MD5"/>
    <SupportValue value="MD4"/>
    <SupportValue value="MD2"/>
</Suite>
<Suite NameSuite="LIMIT_BANDWIDTH_v1">
    <SupportValue value="10Mbit/s"/>
    <SupportValue value="20Mbit/s"/>
    <SupportValue value="30Mbit/s"/>
</Suite>
```